

Clojure In Small Pieces

Rich Hickey

Juozas Baliuka	Eric Bruneton	Chas Emerick
Tom Faulhaber	Stephen C. Gilardi	Daniel Solano Gmez
Christophe Grand	Stuart Halloway	Mike Hinckey
Shawn Hoover	Chris Houser	Rajendra Inamdar
Eugene Kuleshov	Robert Lachlan	J. McConnell
Chris Nokleberg	Allen Rohner	Michel Salim
Jason Sankey	Stuart Sierra	Frantisek Sodomka
Vishal Vishnoi	Jean Niklas Lóränge	Kyle Kingsbury
Kai Wu		

Timothy Daly (Editor)

Based on Version 1.3.0-alpha4

November 14, 2013

Contents

Preface: Why Literate Programming	i
Steps to build Clojure	i
Steps to change Clojure	i
Why Bother?	ii
1 From The Ground Up (Kyle Kingsbury)	1
1.1 Clojure from the ground up	1
1.1.1 Getting set up	1
1.1.2 The structure of programs	2
1.1.3 Review	6
1.2 Clojure from the ground up: basic types	6
1.2.1 Types	6
1.2.2 Integers	7
1.2.3 Fractional numbers	9
1.2.4 Mathematical operations	9
1.2.5 Strings	11
1.2.6 Booleans and logic	13
1.2.7 Symbols	14
1.2.8 Keywords	14
1.2.9 Lists	15
1.2.10 Vectors	16
1.2.11 Sets	18
1.2.12 Maps	19
1.2.13 Putting it all together	20
2 From ideas to implementation	23
2.1 Starting from Java	23
2.2 Extends and Implements	23
2.3 Understanding Persistent Vectors (Jean Lorange)	23
2.3.1 Insertion	26
2.3.2 Popping	28
2.3.3 $O(1) \neq O(\log n)$	31
2.3.4 Persistence	32
2.4 Red Black Trees	39

2.4.1	Persistence	39
2.4.2	Persistent Red Black Trees	39
2.4.3	Node Structure	40
2.4.4	The Black Node implementation	42
2.4.5	Constructing a Black Node	45
2.4.6	The Red Node implementation	46
2.4.7	Constructing a Red Node	49
2.4.8	Okasaki's balance cases	50
2.4.9	Deleting a Node	52
2.4.10	Clojure's balance cases	53
2.4.11	Replacing a Node	55
2.5	Immutable Data Structures	55
2.6	Bit-Partitioned Hash Tries	55
2.6.1	Lists	59
2.6.2	Vectors	59
2.6.3	Hashmaps	59
2.6.4	Seqs	60
2.7	Records	61
2.8	Java Interoperability	62
2.8.1	Language primitives	62
2.8.2	Clojure calling Java	64
2.9	Recursion	65
2.10	Argument Deconstruction	65
2.11	The Read-Eval-Print Loop	65
2.12	Special Forms	65
2.13	Reader Macros	65
2.14	Namespaces	65
2.15	Dynamic Comopilation	66
2.16	Ahead-Of-Time Comopilation	66
2.17	Lazy Evaluation	66
2.18	Metadata	66
2.19	Concurrency	66
2.20	Identity and State	66
2.21	Software Transactional Memory	67
2.22	Symbols	67
2.23	The Lisp Reader	69
2.23.1	Reader Syntax Macros	71
2.23.2	The String reader macro	72
2.23.3	The Comment reader macro	74
2.23.4	The Wrapping reader macro	74
2.23.5	The Meta reader macro	75
2.23.6	The SyntaxQuote reader macro	76
2.23.7	The Unquote reader macro	79
2.23.8	The List reader macro	80
2.23.9	The Unmatched Delimiter reader macro	80
2.23.10	The Vector reader macro	80

2.23.11 The Map reader macro	81
2.23.12 The Character reader macro	81
2.23.13 The Arg reader macro	82
2.24 Reader Dispatch Macros	83
2.24.1 The Var reader macro	84
2.24.2 The Regular Expression reader macro	85
2.24.3 The Function reader macro	86
2.24.4 The Set reader macro	87
2.24.5 The Eval reader macro	87
2.24.6 The Unreadable reader macro	89
2.24.7 The Discard reader macro	89
2.25 Vars	89
2.26 Transients	89
2.27 Atoms	94
2.28 Refs	94
2.29 Agents	94
2.30 Promise and Deliver	94
2.31 Futures	96
2.32 MultiMethods	99
2.33 Deftype	99
2.34 Defrecord	99
2.35 Protocols	101
2.36 Prototypes	102
2.37 Genclass	102
2.37.1 Overriding protected methods	102
2.38 Proxies	103
2.39 Macros	103
2.40 Iterators	103
2.41 Generating Java Classes	105
2.42 Type Hints	105
2.43 Native Arithmetic	105
3 Writing Idiomatic Clojure	107
4 The Ants Demo (Kai Wu)	109
4.1 The Simulation World	109
4.1.1 Initial setup of constants/magic-numbers	109
4.1.2 The board: ready to mutate via transactions	110
4.1.3 Ants as agents - doing asynchronous uncoordinated changes	112
4.1.4 Setting up the home, and ants	113
4.1.5 Orientation and moving around the world	115
4.1.6 Ant-agent behavior functions	116
4.1.7 World behavior: pheromone evaporation	124
4.2 The UI	125
4.2.1 Using the Java AWT	125
4.2.2 Functions to render the board and the ants	126

4.2.3	Setting the scene, then updating it continually	128
4.3	Running the Program	130
4.3.1	Running the simulator	130
5	Parallel Processing	133
5.1	Natural Graphs	133
5.2	Steele's parallel ideas	133
5.3	Sequential mapreduce	137
5.3.1	Parallel mapreduce	139
5.4	MPI Message Passing	141
5.4.1	The N-colony Ant Demo	141
5.5	MapReduce	141
6	The ant build sequence	143
6.1	ant.build	143
6.2	The Execution	149
7	jvm/clojure/asm/	163
7.1	AnnotationVisitor.java	163
7.2	AnnotationWriter.java	165
7.3	Attribute.java	171
7.4	ByteVector.java	176
7.5	ClassAdapter.java	183
7.6	ClassReader.java	185
7.7	ClassVisitor.java	229
7.8	ClassWriter.java	233
7.9	Edge.java	262
7.10	FieldVisitor.java	263
7.11	FieldWriter.java	264
7.12	Frame.java	269
7.13	Handler.java	300
7.14	Item.java	301
7.15	Label.java	305
7.16	MethodAdapter.java	314
7.17	MethodVisitor.java	317
7.18	MethodWriter.java	326
7.19	Opcodes.java	387
7.20	Type.java	394
8	jvm/clojure/asm/commons	413
8.1	AdviceAdapter.java	413
8.2	AnalyzerAdapter.java	426
8.3	CodeSizeEvaluator.java	445
8.4	EmptyVisitor.java	449
8.5	GeneratorAdapter.java	453
8.6	LocalVariablesSorter.java	484

8.7	Method.java	491
8.8	SerialVersionUIDAdder.java	496
8.9	StaticInitMerger.java	506
8.10	TableSwitchGenerator.java	508
9	jvm/clojure/lang/	509
9.1	AFn.java	509
9.2	AFunction.java	519
9.3	Agent.java	520
9.4	AMapEntry.java	527
9.5	APersistentMap.java	530
9.6	APersistentSet.java	538
9.7	APersistentVector.java	541
9.8	AReference.java	552
9.9	ARef.java	553
9.10	ArityException.java	556
9.11	ArrayChunk.java	556
9.12	ArraySeq.java	558
9.13	ASeq.java	571
9.14	Associative.java	576
9.15	Atom.java	577
9.16	ATransientMap.java	579
9.17	ATransientSet.java	581
9.18	BigInt.java	582
9.19	Binding.java	585
9.20	Box.java	585
9.21	ChunkBuffer.java	586
9.22	ChunkedCons.java	586
9.23	Compile.java	588
9.24	Compiler.java	590
9.25	Cons.java	766
9.26	Counted.java	768
9.27	Delay.java	768
9.28	DynamicClassLoader.java	769
9.29	EnumerationSeq.java	770
9.30	Fn.java	772
9.31	IChunk.java	772
9.32	IChunkedSeq.java	773
9.33	IDeref.java	773
9.34	IEditableCollection.java	774
9.35	IFn.java	774
9.36	IKeywordLookup.java	796
9.37	ILookup.java	797
9.38	ILookupSite.java	797
9.39	ILookupThunk.java	798
9.40	IMapEntry.java	798

9.41 IMeta.java	799
9.42 Indexed.java	799
9.43 IndexedSeq.java	799
9.44 IObj.java	800
9.45 IPersistentCollection.java	800
9.46 IPersistentList.java	801
9.47 IPersistentMap.java	801
9.48 IPersistentSet.java	802
9.49 IPersistentStack.java	802
9.50 IPersistentVector.java	802
9.51 IPromiseImpl.java	803
9.52 IProxy.java	803
9.53 IReduce.java	804
9.54 IReference.java	804
9.55 IRef.java	805
9.56 ISeq.java	805
9.57 IteratorSeq.java	806
9.58 ITransientAssociative.java	807
9.59 ITransientCollection.java	808
9.60 ITransientMap.java	808
9.61 ITransientSet.java	809
9.62 ITransientVector.java	809
9.63 Keyword.java	810
9.64 KeywordLookupSite.java	816
9.65 LazilyPersistentVector.java	817
9.66 LazySeq.java	818
9.67 LineNumberingPushbackReader.java	823
9.68 LispReader.java	825
9.69 LockingTransaction.java	836
9.70 MapEntry.java	850
9.71 MapEquivalence.java	850
9.72 MethodImplCache.java	851
9.73 MultiFn.java	852
9.74 Named.java	861
9.75 Namespace.java	861
9.76 Numbers.java	867
9.77 Obj.java	947
9.78 PersistentArrayMap.java	948
9.79 PersistentHashMap.java	955
9.80 PersistentHashSet.java	980
9.81 PersistentList.java	982
9.82 PersistentQueue.java	989
9.83 PersistentStructMap.java	995
9.84 PersistentTreeMap.java	1000
9.85 PersistentTreeSet.java	1012
9.86 PersistentVector.java	1014

9.87 ProxyHandler.java	1030
9.88 Range.java	1031
9.89 Ratio.java	1033
9.90 Ref.java	1034
9.91 Reflector.java	1044
9.92 Repl.java	1054
9.93 RestFn.java	1055
9.94 Reversible.java	1094
9.95 RT.java	1094
9.96 Script.java	1138
9.97 SeqEnumeration.java	1138
9.98 SeqIterator.java	1139
9.99 Seqable.java	1140
9.100Sequential.java	1140
9.101Settable.java	1140
9.102Sorted.java	1141
9.103StringSeq.java	1141
9.104Symbol.java	1143
9.105TransactionalHashMap.java	1145
9.106Util.java	1149
9.107Var.java	1152
9.108XMLHandler.java	1164
10 jvm/clojure	1167
10.1 main.java	1167
11 clj/clojure/	1169
11.1 core.clj	1169
11.2 protocols.clj	1296
11.3 core'deftype.clj	1298
11.4 core'print.clj	1315
11.5 core'proxy.clj	1323
11.6 data.clj	1332
11.7 genclass.clj	1334
11.8 gvec.clj	1351
11.9 inspector.clj	1361
11.10browse.clj	1365
11.11browse'ui.clj	1366
11.12io.clj	1367
11.13javadoc.clj	1376
11.14shell.clj	1378
11.15main.clj	1381
11.16parallel.clj	1389
11.17cl'format.clj	1394
11.18column'writer.clj	1436
11.19dispatch.clj	1438

11.20pprint`base.clj	1448
11.21pretty`writer.clj	1456
11.22print`table.clj	1467
11.23utilities.clj	1467
11.24pprint.clj	1470
11.24.1java.clj	1471
11.25reflect.clj	1476
11.26repl.clj	1479
11.27set.clj	1485
11.28stacktrace.clj	1489
11.29string.clj	1490
11.30template.clj	1496
11.31junit.clj	1497
11.32tap.clj	1501
11.33test.clj	1503
11.34version.properties	1519
11.35walk.clj	1519
11.36xml.clj	1522
11.37zip.clj	1524
11.38pom-template.xml	1531
12 test/clojure	1533
12.1 test/test`clojure.clj	1533
12.2 test/test`helper.clj	1535
12.3 test/agents.clj	1537
12.4 test/annotations.clj	1541
12.5 test/atoms.clj	1541
12.6 test/clojure`set.clj	1542
12.7 test/clojure`xml.clj	1546
12.8 test/clojure`zip.clj	1547
12.9 test/compilation.clj	1548
12.10test/control.clj	1550
12.11test/data.clj	1556
12.12test/data`structures.clj	1557
12.13test/def.clj	1574
12.14test/errors.clj	1575
12.15test/evaluation.clj	1576
12.16test/for.clj	1580
12.17test/genclass.clj	1583
12.18test/java`interop.clj	1585
12.19test/keywords.clj	1594
12.20test/logic.clj	1594
12.21test/macros.clj	1599
12.22test/main.clj	1599
12.23test/metadata.clj	1600
12.24test/multimethods.clj	1602

12.25test/ns'libs.clj	1605
12.26test/numbers.clj	1607
12.27test/other'functions.clj	1618
12.28test/parallel.clj	1620
12.29test/pprint.clj	1621
12.30test/predicates.clj	1621
12.31test/printer.clj	1624
12.32test/protocols.clj	1626
12.33test/reader.clj	1633
12.34test/reflect.clj	1640
12.35test/refs.clj	1640
12.36test/repl.clj	1641
12.37test/rt.clj	1642
12.38test/sequences.clj	1644
12.39test/serialization.clj	1668
12.40test/special.clj	1671
12.41test/string.clj	1672
12.42test/test.clj	1674
12.43test/test'fixtures.clj	1677
12.44test/transients.clj	1678
12.45test/vars.clj	1678
12.46test/vectors.clj	1680
12.47test/java'5.clj	1686
12.48test/java'6'and'later.clj	1688
12.49test/examples.clj	1690
12.50test/io.clj	1691
12.51test/javadoc.clj	1696
12.52test/shell.clj	1696
12.53test/test'cl/format.clj	1697
12.54test1/test'helper.clj	1714
12.55test/test'pretty.clj	1715
12.56test1/examples.clj	1721
12.57test/more'examples.clj	1722
12.58test/example.clj	1722
A External Java References	1723
B Copyright and Licenses	1725
C Building Clojure from this document	1733
C.1 The basic idea	1733
C.2 The tangle function in C	1733
C.3 Makefile	1737
C.4 The tangle function in Clojure	1744
C.4.1 Author and License	1744
C.4.2 Abstract and Use Cases	1744

C.4.3	The Latex Support Code	1746
C.4.4	Imports	1747
C.4.5	The Tangle Command	1748
C.4.6	The say function	1748
C.4.7	The read-file function	1748
C.4.8	The ischunk function	1749
C.4.9	The haschunks function	1750
C.4.10	The expand function	1751
C.4.11	The tangle function	1751
Bibliography		1753
Index		1755

Foreword

Rich Hickey invented Clojure. This is a fork of the project to experiment with literate programming as a development and documentation technology.

Every effort is made to give credit for any and all contributions.

Clojure is a break with the past traditions of Lisp. This literate fork is a break with the past traditions of code development. As such it is intended as an experiment, not a replacement or competition with the official version of Clojure.

Most programmers are still locked into the idea of making a program out of a large pile of tiny files containing pieces of programs. They do not realize that this organization was forced by the fact that machines like the PDP 11 only had 8k of memory and a limit of 4k buffers in the editor. Thus there was a lot of machinery built up, such as overlay linkers, to try to reconstruct the whole program.

The time has come to move into a more rational means of creating and maintaining programs. Knuth suggested we write programs like we write literature, with the idea that we are trying to communicate the ideas to other people. The fact that the machine can also run the programs is a useful side-effect but not important.

Very few people have seen a literate program so this is intended as a complete working example, published in book form. The intent is that you can sit and read this book like any other novel. At the end of it you will be familiar with the ideas and how those ideas are actually expressed in the code.

If programmers can read it and understand it then they can maintain and modify it. The ideas will have been communicated. The code will be changed to match changes in the idea. We will all benefit.

I've tried to make it as simple as possible. Try it once, you might like it.

Tim Daly
December 28, 2010 ((iHy))

Preface: Why Literate Programming

This is a literate program, inspired by Donald Knuth [Knu84]. It is intended to be read like a novel from cover to cover. The ideas are expressed clearly but they are grounded in the actual source code.

The code in this document is the executable source. The appendix gives the procedure for building a running system from the enclosed sources.

Steps to build Clojure

Step 1

Basically you need the C program [1733] tangle.c which you can clip using a text editor and save it as tangle.c.

Step 2

Compile tangle.c to create a function called tangle.

```
gcc -o tangle tangle.c
```

Step 3 Run tangle to extract the [1737] Makefile from this document.

```
tangle clojure.pamphlet Makefile >Makefile
```

Step 4

```
make
```

This will

- create a new subdirectory called “tpd” containing all of the source code
- test Clojure
- create a running copy of Clojure
- create the pdf
- start a Clojure REPL

Steps to change Clojure

If you make changes to this document and want the new changes just type:

```
rm -rf tpd  
tangle clojure.pamphlet Makefile >Makefile  
make
```

or you can combine them into one line:

```
rm -rf tpd && tangle clojure.pamphlet Makefile >Makefile && make
```

This will destroy the old source, extract the Makefile, and rebuild the system. On a fast processor this takes about a minute.

Resist the urge to edit the files in the tpd directory. They are only there for the compiler. Edit this file directly.

You can change where Clojure is built. In the [1737] Makefile there is a line which defines the root of the directory build. You can change this or override it on the command line to build Clojure elsewhere.

`WHERE=tpd`

To build a second copy of Clojure, or to work in some other directory, just type

`make WHERE=newplace`

Why Bother?

Why bother with such a difficult method of programming? Because worthwhile programs should “live”.

Programs “live” because people maintain them. Maintaining and modifying code correctly requires that you understand why the program is written as it is, what the key ideas that make the program work, and why certain, not very obvious, pieces of code exist. Programmers almost never write this information down anywhere. Great ideas are invented, the code is written, a man page of documentation is created, and the job is done.

Well, almost. What does it mean for a program to “live”? How does a program survive once the original developers leave the project? There are many sources of information but almost no source of knowledge. New programmers don’t know what the “elders” know. In order to “live” and continue to grow there has to be a way to transmit this knowledge.

Literate programming is Knuth’s proposed idea for moving from the world of ideas to the world of information. This is not simply another documentation format. This is meant to be **Literature**. The ideas are presented, the implications are explored, the tradeoffs are discussed, and the code is “motivated”, like characters in a novel.

You are encouraged to write or rewrite sections of this document to improve the communication with the readers.

“But I have to learn latex!” . Well, for this document you do. But L^AT_EX is just more than a document markup language like HTML and it is no harder to learn. It gives you the added advantage that you have a real language for publishing real documents. Most books are typeset with this technology and a lot of conferences and Journals require it. If you can learn Clojure, you can learn L^AT_EX. If you’re a programmer you will always need to continue to learn, at least until you retire into management.

Having used literate programming for years I have collected some key quotes that might stimulate your interest.

I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title “Literate Programming”. Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate on explaining to human beings what we want a computer to do.

—Donald Knuth “Literate Programming (1984)”

Step away from the machine. Literate programming has nothing to do with tools or style. It has very little to do with programming. One of the hard transitions to literate programming is “literate thinking”.

—Timothy Daly in Lambda the Ultimate (2010)

The effect of this simple shift of emphasis can be so profound as to change one’s whole approach to programming. Under the literate programming paradigm, the central activity of programming becomes that of conveying meaning to other intelligent beings rather than merely convincing the computer to behave in a particular way. It is the difference between performing and exposing a magic trick.

—Ross Williams, FunnelWeb Tutorial Manual

Another thing I’ve been enjoying lately is literate programming. Amazingly it turns out to be faster to write a literate program than an ordinary program because debugging takes almost no time.

—Bill Hart, SAGE Mailing list, May 3, 2010

The conversation is much more direct if the Design Concept per se, rather than derivative representatives or partial details, is the focus.

—Fred Brooks, “The Design of Design”

We are banning the old notion of literate programming that I used when developing TeX82 because documentation has proven to be too much of a pain.

—Donald Knuth TUG 2010

Once upon a time I took great care to ensure that TeX82 would be truly archival so that results obtainable today would produce the same output 50 years from now but that was manifestly foolish. Let's face it, who is going to care one whit for what I do today after even 5 years have elapsed, let alone 50. Life is too short to re-read anything anymore in the internet age. Nothing over 30 months old is trustworthy or interesting.

—Donald Knuth TUG 2010

Chapter 1

From The Ground Up (Kyle Kingsbury)

This is quoted from Kingsbury. [King13]

1.1 Clojure from the ground up

1.1.1 Getting set up

When you have a JDK, youll need Leiningen, the Clojure build tool. If youre on a Linux or OS X computer, these instructions should get you going right away. If youre on Windows, see the Leiningen page for an installer. If you get stuck, you might want to start with a primer on command line basics.

```
mkdir -p ~/bin
cd ~/bin
wget https://raw.github.com/technomancy/leiningen/stable/bin/lein
chmod a+x lein
```

Leiningen automatically handles installing Clojure, finding libraries from the internet, and building and running your programs. Well create a new Leiningen project to play around in:

```
cd
lein new scratch
```

This creates a new directory in your homedir, called scratch. If you see command not found instead, it means the directory /bin isnt registered with your terminal as a place to search for programs. To fix this, add the line

```
export PATH="$PATH":~/bin
```

to the file .bash_profile in your home directory, then run

```
source ~/.bash_profile.
```

Re-running

```
lein new scratch
```

should work.

Lets enter that directory, and start using Clojure itself:

```
cd scratch
lein repl
```

1.1.2 The structure of programs

```
boa$ lein repl
```

which responds with

```
nREPL server started on port 45413
REPL-y 0.2.0
Clojure 1.5.1
Docs: (doc function-name-here)
      (find-doc "part-of-name-here")
Source: (source function-name-here)
Javadoc: (javadoc java-object-or-class-here)
Exit: Control+D or (exit) or (quit)
user=>
```

This is an interactive Clojure environment called a REPL, for Read, Execute, Print Loop. Its going to read a program we enter, run that program, and print the results. REPLs give you quick feedback, so theyre a great way to explore a program interactively, run tests, and prototype new ideas.

Lets write a simple program. The simplest, in fact. Type nil, and hit enter.

```
user=> nil
nil
```

nil is the most basic value in Clojure. It represents emptiness, nothing-doing, not-a-thing. The absence of information.

```
user=> true
true
user=> false
false
```

true and **false** are a pair of special values called Booleans. They mean exactly what you think: whether a statement is true or false. **true**, **false**, and **nil** form the three poles of the Lisp logical system.

```
user=> 0
0
```

This is the number zero. Its numeric friends are 1, -47, 1.2e-4, 1/3, and so on. We might also talk about strings, which are chunks of text surrounded by double quotes:

```
user=> "hi there!"
"hi there!"
```

nil, **true**, **0**, and "hi there!" are all different types of values; the nouns of programming. Just as one could say House. in English, we can write a program like "hello, world" and it evaluates to itself: the string "hello world". But most sentences aren't just about stating the existence of a thing; they involve action. We need verbs.

```
user=> inc
#<core$inc clojure.core$inc@6f7ef41c>
```

This is a verb called `inc`short for increment. Specifically, `inc` is a symbol which points to a verb: `#<core$inc clojure.core$inc@6f7ef41c>` just like the word `run` is a name for the concept of running.

Theres a key distinction herethat a signifier, a reference, a label, is not the same as the signified, the referent, the concept itself. If you write the word `run` on paper, the ink means nothing by itself. Its just a symbol. But in the mind of a reader, that symbol takes on meaning; the idea of running.

Unlike the number `0`, or the string `hi`, symbols are references to other values. when Clojure evaluates a symbol, it looks up that symbols meaning. Look up `inc`, and you get `#<core$inc clojure.core$inc@6f7ef41c>`.

Can we refer to the symbol itself, without looking up its meaning?

```
user=> 'inc
inc
```

Yes. The single quote '`'` escapes an expression. It says Rather than evaluating this text, simply return the text itself, unchanged. Quote a symbol, get a symbol. Quote a number, get a number. Quote anything, and get it back exactly as it came in.

```
user=> '123
```

```
123
user=> '"foo"
"foo"
user=> '(1 2 3)
(1 2 3)
```

A new kind of value, surrounded by parentheses: the list. LISP originally stood for LISt Processing, and lists are still at the core of the language. This list contains three elements: the numbers 1, 2, and 3. Lists can contain anything: numbers, strings, even other lists:

```
user=> '(nil "hi")
(nil "hi")
```

A list containing two elements: the number 1, and a second list. That list contains two elements: the number 2, and another list. That list contains two elements: 3, and an empty list.

```
user=> '(1 (2 (3 ())))
(1 (2 (3)))
```

You could think of this structure as a tree which is a provocative idea, because languages are like trees too: sentences are comprised of clauses, which can be nested, and each clause may have subjects modified by adjectives, and verbs modified by adverbs, and so on. "Lindsay, my best friend, took the dog which we found together at the pound on fourth street, for a walk with her mother Michelle.

```
Took
  Lindsay
    my best friend
    the dog
      which we found together
        at the pound
          on fourth street
        for a walk
        with her mother
      Michelle
```

But lets try something simpler. Something we know how to talk about. Increment the number zero. As a tree:

```
Increment
  the number zero
```

We have a symbol for incrementing, and we know how to write the number zero. Lets combine them in a list:

```
clj=> '(inc 0)
(inc 0)
```

A basic sentence. Remember, since its quoted, were talking about the tree, the text, the expression, by itself. Absent interpretation. If we remove the single-quote, Clojure will interpret the expression:

```
user=> (inc 0)
1
```

Incrementing zero yields one. And if we wanted to increment that value? Increment increment the number zero

```
user=> (inc (inc 0))
2
```

A sentence in Lisp is a list. It starts with a verb, and is followed by zero or more objects for that verb to act on. Each part of the list can itself be another list, in which case that nested list is evaluated first, just like a nested clause in a sentence. When we type

```
(inc (inc 0))
```

Clojure first looks up the meanings for the symbols in the code:

```
(#<core$inc clojure.core$inc@6f7ef41c>
 (#<core$inc clojure.core$inc@6f7ef41c>
  0))
```

Then evaluates the innermost list (**inc 0**), which becomes the number 1:

```
(#<core$inc clojure.core$inc@6f7ef41c>
 1)
```

Finally, it evaluates the outer list, incrementing the number 1:

```
2
```

Every list starts with a verb. Parts of a list are evaluated from left to right. Innermost lists are evaluated before outer lists.

```
(+ 1 (- 5 2) (+ 3 4))
(+ 1 3           (+ 3 4))
(+ 1 3           7)
11
```

Thats it.

The entire grammar of Lisp: the structure for every expression in the language. We transform expressions by substituting meanings for symbols, and obtain some result. This is the core of the Lambda Calculus, and it is the theoretical basis for almost all computer languages. Ruby, Javascript, C, Haskell; all languages express the text of their programs in different ways, but internally all construct a tree of expressions. Lisp simply makes it explicit.

1.1.3 Review

We started by learning a few basic nouns: numbers like 5, strings like "cat", and symbols like inc and +. We saw how quoting makes the difference between an expression itself and the thing it evaluates to. We discovered symbols as names for other values, just like how words represent concepts in any other language. Finally, we combined lists to make trees, and used those trees to represent a program.

With these basic elements of syntax in place, its time to expand our vocabulary with new verbs and nouns; learning to represent more complex values and transform them in different ways.

1.2 Clojure from the ground up: basic types

Weve learned the basics of Clojures syntax and evaluation model. Now well take a tour of the basic nouns in the language.

1.2.1 Types

Weve seen a few different values alreadyfor instance, nil, true, false, 1, 2.34, and "meow". Clearly all these things are different values, but some of them seem more alike than others.

For instance, 1 and 2 are very similar numbers; both can be added, divided, multiplied, and subtracted. 2.34 is also a number, and acts very much like 1 and 2, but its not quite the same. Its got decimal points. Its not an integer. And clearly true is not very much like a number. What is true plus one? Or false divided by 5.3? These questions are poorly defined.

We say that a type is a group of values which work in the same way. Its a property that some values share, which allows us to organize the world into sets of similar things. $1 + 1$ and $1 + 2$ use the same addition, which adds together integers. Types also help us verify that a program makes sense: that you can only add together numbers, instead of adding numbers to porcupines.

Types can overlap and intersect each other. Cats are animals, and cats are fuzzy

too. You could say that a cat is a member (or sometimes instance), of the fuzzy and animal types. But there are fuzzy things like moss which aren't animals, and animals like alligators that aren't fuzzy in the slightest.

Other types completely subsume one another. All tabbies are housecats, and all housecats are felidae, and all felidae are animals. Everything which is true of an animal is automatically true of a housecat. Hierarchical types make it easier to write programs which don't need to know all the specifics of every value; and conversely, to create new types in terms of others. But they can also get in the way of the programmer, because not every useful classification (like fuzziness) is purely hierarchical. Expressing overlapping types in a hierarchy can be tricky.

Every language has a type system; a particular way of organizing nouns into types, figuring out which verbs make sense on which types, and relating types to one another. Some languages are strict, and others more relaxed. Some emphasize hierarchy, and others a more ad-hoc view of the world. We call Clojure's type system strong in that operations on improper types are simply not allowed: the program will explode if asked to subtract a dandelion. We also say that Clojure's types are dynamic because they are enforced when the program is run, instead of when the program is first read by the computer.

We'll learn more about the formal relationships between types later, but for now, keep this in the back of your head. It'll start to hook in to other concepts later.

1.2.2 Integers

Lets find the type of the number 3:

```
user=> (type 3)
java.lang.Long
```

So 3 is a `java.lang.Long`, or a `Long`, for short. Because Clojure is built on top of Java, many of its types are plain old Java types.

`Longs`, internally, are represented as a group of sixty-four binary digits (ones and zeroes), written down in a particular pattern called signed twos complement representation. You don't need to worry about the specifics—there are only two things to remember about longs. First, longs use one bit to store the sign: whether the number is positive or negative. Second, the other 63 bits store the size of the number. That means the biggest number you can represent with a long is $2^{63} - 1$ (the minus one is because of the number 0), and the smallest long is -2^{63} .

How big is $2^{63} - 1$?

```
user=> Long/MAX_VALUE
9223372036854775807
```

Thats a reasonably big number. Most of the time, you wont need anything bigger, but what if you did? What happens if you add one to the biggest Long?

```
user=> (inc Long/MAX_VALUE)
ArithmException integer overflow clojure.lang.Numbers.throwIntOverflow
(Numbers.java:1388)
```

An error occurs! This is Clojure telling us that something went wrong. The type of error was an ArithmException, and its message was integer overflow, meaning this type of number cant hold a number that big. The error came from a specific place in the source code of the program: Numbers.java, on line 1388. Thats a part of the Clojure source code. Later, well learn more about how to unravel error messages and find out what went wrong.

The important thing is that Clojures type system protected us from doing something dangerous; instead of returning a corrupt value, it aborted evaluation and returned an error.

If you do need to talk about really big numbers, you can use a BigInt: an arbitrary-precision integer. Lets convert the biggest Long into a BigInt, then increment it:

```
user=> (inc (bigint Long/MAX_VALUE)) 9223372036854775808N
```

Notice the N at the end? Thats how Clojure writes arbitrary-precision integers.

```
user=> (type 5N)
clojure.lang.BigInt
```

There are also smaller numbers.

```
user=> (type (int 0))
java.lang.Integer
user=> (type (short 0))
java.lang.Short
user=> (type (byte 0))
java.lang.Byte
```

Integers are half the size of Longs; they store values in 32 bits. Shorts are 16 bits, and Bytes are 8. That means their biggest values are 2^{31-1} , 2^{15-1} , and 2^{7-1} , respectively.

```
user=> Integer/MAX_VALUE
2147483647
user=> Short/MAX_VALUE
32767
user=> Byte/MAX_VALUE
127
```

1.2.3 Fractional numbers

To represent numbers between integers, we often use floating-point numbers, which can represent small numbers with fine precision, and large numbers with coarse precision. Floats use 32 bits, and Doubles use 64. Doubles are the default in Clojure.

```
user=> (type 1.23)
java.lang.Double
user=> (type (float 1.23))
java.lang.Float
```

Floating point math is complicated, and we wont get bogged down in the details just yet. The important thing to know is floats and doubles are approximations. There are limits to their correctness:

```
user=> 0.9999999999999999
1.0
```

To represent fractions exactly, we can use the ratio type:

```
user=> (type 1/3)
clojure.lang.Ratio
```

1.2.4 Mathematical operations

The exact behavior of mathematical operations in Clojure depends on their types. In general, though, Clojure aims to preserve information. Adding two longs returns a long; adding a double and a long returns a double.

```
user=> (+ 1 2)
3
user=> (+ 1 2.0)
3.0
```

3 and 3.0 are not the same number; one is a long, and the other a double. But for most purposes, theyre equivalent, and Clojure will tell you so:

```
user=> (= 3 3.0)
false
user=> (== 3 3.0)
true
```

`=` asks whether all the things that follow are equal. Since floats are approximations, `=` considers them different from integers. `==` also compares things,

but a little more loosely: it considers integers equivalent to their floating-point representations.

We can also subtract with `-`, multiply with `*`, and divide with `/`.

```
user=> (- 3 1)
2
user=> (* 1.5 3)
4.5
user=> (/ 1 2)
1/2
```

Putting the verb first in each list allows us to add or multiply more than one number in the same step:

```
user=> (+ 1 2 3)
6
user=> (* 2 3 1/5)
6/5
```

Subtraction with more than 2 numbers subtracts all later numbers from the first. Division divides the first number by all the rest.

```
user=> (- 5 1 1 1)
2
user=> (/ 24 2 3)
4
```

By extension, we can define useful interpretations for numeric operations with just a single number:

```
user=> (+ 2)
2
user=> (- 2)
-2
user=> (* 4)
4
user=> (/ 4)
1/4
```

We can also add or multiply a list of no numbers at all, obtaining the additive and multiplicative identities, respectively. This might seem odd, especially coming from other languages, but well see later that these generalizations make it easier to reason about higher-level numeric operations.

```
user=> (+)
0
user=> (*)
1
```

Often, we want to ask which number is bigger, or if one number falls between two others. `<=` means less than or equal to, and asserts that all following values are in order from smallest to biggest.

```
user=> (<= 1 2 3)
true
user=> (<= 1 3 2)
false
```

`<` means strictly less than, and works just like `<=`, except that no two values may be equal.

```
user=> (<= 1 1 2)
true
user=> (< 1 1 2)
false
```

Their friends `>` and `>=` mean greater than and greater than or equal to, respectively, and assert that numbers are in descending order.

```
user=> (> 3 2 1)
true
user=> (> 1 2 3)
false
```

Also commonly used are `inc` and `dec`, which add and subtract one to a number, respectively:

```
user=> (inc 5)
6
user=> (dec 5)
4
```

One final note: equality tests can take more than 2 numbers as well.

```
user=> (= 2 2 2)
true
user=> (= 2 2 3)
false
```

1.2.5 Strings

We saw that strings are text, surrounded by double quotes, like `"foo"`. Strings in Clojure are, like Longs, Doubles, and company, backed by a Java type:

```
user=> (type "cat")
java.lang.String
```

We can make almost anything into a string with str. Strings, symbols, numbers, booleans; every value in Clojure has a string representation. Note that nils string representation is ""; an empty string.

```
user=> (str "cat")
"cat"
user=> (str 'cat)
"cat"
user=> (str 1)
"1"
user=> (str true)
"true"
user=> (str '(1 2 3))
"(1 2 3)"
user=> (str nil)
""
```

str can also combine things together into a single string, which we call concatenation.

```
user=> (str "meow " 3 " times")
"meow 3 times"
```

To look for patterns in text, we can use a regular expression, which is a tiny language for describing particular arrangements of text. re-find and re-matches look for occurrences of a regular expression in a string. To find a cat:

```
user=> (re-find #"\cat" "mystic cat mouse")
"cat"
user=> (re-find #"\cat" "only dogs here")
nil
```

That #"\..." is Clojures way of writing a regular expression.

With re-matches, you can extract particular parts of a string which match an expression. Here we find two strings, separated by a ::. The parentheses mean that the regular expression should capture that part of the match. We get back a list containing the part of the string that matched the first parentheses, followed by the part that matched the second parentheses.

```
user=> (rest (re-matches #"\(.+):(.+)" "mouse:treat"))
("mouse" "treat")
```

Regular expressions are a powerful tool for searching and matching text, especially when working with data files. Since regexes work the same in most languages, you can use any guide online to learn more. Its not something you have to master right away; just learn specific tricks as you find you need them.

1.2.6 Booleans and logic

Everything in Clojure has a sort of charge, a truth value, sometimes called truthiness. true is positive and false is negative. nil is negative, too.

```
user=> (boolean true)
true
user=> (boolean false)
false
user=> (boolean nil)
false
```

Every other value in Clojure is positive.

```
user=> (boolean 0)
true
user=> (boolean 1)
true
user=> (boolean "hi there")
true
user=> (boolean str)
true
```

If you're coming from a C-inspired language, where 0 is considered false, this might be a bit surprising. Likewise, in much of POSIX, 0 is considered success and nonzero values are failures. Lisp allows no such confusion: the only negative values are false and nil.

We can reason about truth values using **and**, **or**, and **not**. **and** returns the first negative value, or the last value if all are truthy.

```
user=> (and true false true)
false
user=> (and true true true)
true
user=> (and 1 2 3)
3
```

Similarly, **or** returns the first positive value.

```
user=> (or false 2 3)
2
user=> (or false nil)
nil
```

And **not** inverts the logical sense of a value:

```
user=> (not 2)
false
user=> (not nil)
true
```

We'll learn more about Boolean logic when we start talking about control flow; the way we alter evaluation of a program and express ideas like if I'm a cat, then meow incessantly.

1.2.7 Symbols

We saw symbols in the previous chapter; they're bare strings of characters, like foo or +.

```
user=> (class 'str)
clojure.lang.Symbol
```

Every symbol actually has two names: one, a short name, is used to refer to things locally. Another is the fully qualified name, which is used to refer unambiguously to a symbol from anywhere. If I were a symbol, my name would be Kyle, and my full name Kyle Kingsbury.

Symbol names are separated with a /. For instance, the symbol str actually comes from a family called clojure.core, which means that its full name is clojure.core/str

```
user=> (= str clojure.core/str)
true
user=> (name 'clojure.core/str)
"str"
```

When we talked about the maximum size of an integer, that was a fully-qualified symbol, too.

```
user=> (type 'Integer/MAX_VALUE)
clojure.lang.Symbol
```

The job of symbols is to refer to things, to point to other values. When evaluating a program, symbols are looked up and replaced by their corresponding values. That's not the only use of symbols, but it's the most common.

1.2.8 Keywords

Closely related to symbols and strings are keywords, which begin with a :. Keywords are like strings in that they're made up of text, but are specifically

intended for use as labels or identifiers. These aren't labels in the sense of symbols: keywords aren't replaced by any other value. They're just names, by themselves.

```
user=> (type :cat)
clojure.lang.Keyword
user=> (str :cat)
":cat"
user=> (name :cat)
"cat"
```

As labels, keywords are most useful when paired with other values in a collection, like a map. We'll come back to keywords shortly.

1.2.9 Lists

A collection is a group of values. It's a container which provides some structure, some framework, for the things that it holds. We say that a collection contains elements, or members. We saw one kind of collection—a list—in the previous chapter.

```
user=> '(1 2 3)
(1 2 3)
user=> (type '(1 2 3))
clojure.lang.PersistentList
```

Remember, we quote lists with a '`'` to prevent them from being evaluated. You can also construct a list using `list`:

```
user=> (list 1 2 3)
(1 2 3)
```

Lists are comparable just like every other value:

```
user=> (= (list 1 2) (list 1 2))
true
```

You can modify a list by conjoining an element onto it:

```
user=> (conj '(1 2 3) 4)
(4 1 2 3)
```

We added 4 to the list—but it appeared at the front. Why? Internally, lists are stored as a chain of values: each link in the chain is a tiny box which holds the value and a connection to the next link. This data structure, called a linked list, offers immediate access to the first element.

```
user=> (first (list 1 2 3))
1
```

But getting to the second element requires an extra hop down the chain

```
user=> (second (list 1 2 3))
2
```

and the third element a hop after that, and so on.

```
user=> (nth (list 1 2 3) 2)
3
```

nth gets the element of an ordered collection at a particular index. The first element is index 0, the second is index 1, and so on.

This means that lists are well-suited for small collections, or collections which are read in linear order, but are slow when you want to get arbitrary elements from later in the list. For fast access to every element, we use a vector.

1.2.10 Vectors

Vectors are surrounded by square brackets, just like lists are surrounded by parentheses. Because vectors aren't evaluated like lists are, there's no need to quote them:

```
user=> [1 2 3]
[1 2 3]
user=> (type [1 2 3])
clojure.lang.PersistentVector
```

You can also create vectors with `vector`, or change other structures into vectors with `vec`:

```
user=> (vector 1 2 3)
[1 2 3]
user=> (vec (list 1 2 3))
[1 2 3]
```

`conj` on a vector adds to the end, not the start:

```
user=> (conj [1 2 3] 4) [1 2 3 4]
```

Our friends **first**, **second**, and **nth** work here too; but unlike lists, **nth** is fast on vectors. That's because internally, vectors are represented as a very broad tree

of elements, where each part of the tree branches into 32 smaller trees. Even very large vectors are only a few layers deep, which means getting to elements only takes a few hops.

In addition to first, youll often want to get the remaining elements in a collection. There are two ways to do this:

```
user=> (rest [1 2 3])
(2 3)
user=> (next [1 2 3])
(2 3)
```

rest and **next** both return everything but the first element. They differ only by what happens when there are no remaining elements:

```
user=> (rest [1])
()
user=> (next [1])
nil
```

rest returns logical **true**, **next** returns logical **false**. Each has their uses, but in almost every case theyre equivalentI interchange them freely.

We can get the final element of any collection with last:

```
user=> (last [1 2 3])
3
```

And figure out how big the vector is with count:

```
user=> (count [1 2 3])
3
```

Because vectors are intended for looking up elements by index, we can also use them directly as verbs:

```
user=> ([:a :b :c] 1)
:b
```

So we took the vector containing three keywords, and asked Whats the element at index 1? Index 1 is the second element, so this evaluates to :b.

Finally, note that vectors and lists containing the same elements are considered equal in Clojure:

```
user=> (= '(1 2 3) [1 2 3])
true
```

In almost all contexts, you can consider vectors, lists, and other sequences as interchangeable. They only differ in their performance characteristics, and in a few data-structure-specific operations.

1.2.11 Sets

Sometimes you want an unordered collection of values; especially when you plan to ask questions like does the collection have the number 3 in it? Clojure, like most languages, calls these collections sets.

```
user=> #{:a :b :c}
#{:a :c :b}
```

Sets are surrounded by `#{:...}`. Notice that though we gave the elements `:a`, `:b`, and `:c`, they came out in a different order. In general, the order of sets can shift at any time. If you want a particular order, you can ask for it as a list or vector:

```
user=> (vec #{:a :b :c})
[:a :c :b]
```

Or ask for the elements in sorted order:

```
user=> (sort #{:a :b :c})
(:a :b :c)
```

`conj` on a set adds an element:

```
user=> (conj #{:a :b :c} :d)
#{:a :b :c :d}
user=> (conj #{:a :b :c} :a)
#{:a :b :c :a}
```

Sets never contain an element more than once, so conjing an element which is already present does nothing. Conversely, one removes elements with `disj`:

```
user=> (disj #{"hornet" "hummingbird"} "hummingbird")
#{"hornet"}
```

The most common operation with a set is to check whether something is inside it. For this we use `contains?`.

```
user=> (contains? #{1 2 3} 3)
true
user=> (contains? #{1 2 3} 5)
false
```

Like vectors, you can use the set itself as a verb. Unlike `contains?`, this expression returns the element itself (if it was present), or `nil`.

```
user=> (#{}{1 2 3} 3)
3
user=> (#{}{1 2 3} 4)
nil
```

You can make a set out of any other collection with `set`.

```
user=> (set [:a :b :c])
#{:a :c :b}
```

1.2.12 Maps

The last collection on our tour is the map: a data structure which associates keys with values. In a dictionary, the keys are words and the definitions are the values. In a library, keys are call signs, and the books are values. Maps are indexes for looking things up, and for representing different pieces of named information together.

```
user=> {:name "spook" :weight 2 :color "black"}
{:weight 2, :name "spook", :color "black"}
```

Maps are surrounded by braces `{...}`, filled by alternating keys and values. In this map, the three keys are `:name`, `:color`, and `:weight`, and their values are `"spook"`, `"black"`, and `2`, respectively. We can look up the corresponding value for a key with `get`:

```
user=> (get {"cat" "meow" "dog" "woof"} "cat")
"meow"
user=> (get {:a 1 :b 2} :c)
nil
```

`get` can also take a default value to return instead of `nil`, if the key doesn't exist in that map.

```
user=> (get {:glinda :good} :wicked :not-here)
:not-here
```

Since lookups are so important for maps, we can use a map as a verb directly:

```
user=> ({"amlodipine" 12 "ibuprofin" 50} "ibuprofin")
50
```

And conversely, keywords can also be used as verbs, which look themselves up in maps:

```
user=> (:raccoon {:weasel "queen" :raccoon "king"})
"king"
```

You can add a value for a given key to a map with **assoc**.

```
user=> (assoc {:bolts 1088} :camshafts 3)
{:camshafts 3 :bolts 1088}
user=> (assoc {:camshafts 3} :camshafts 2)
{:camshafts 2}
```

assoc adds keys if they aren't present, and replaces values if they're already there. If you associate a value onto nil, it creates a new map.

```
user=> (assoc nil 5 2)
{5 2}
```

You can combine maps together using **merge**, which yields a map containing all the elements of all given maps, preferring the values from later ones.

```
user=> (merge {:a 1 :b 2} {:b 3 :c 4})
{:c 4, :a 1, :b 3}
```

Finally, to remove a value, use **dissoc**.

```
user=> (dissoc {:potatoes 5 :mushrooms 2} :mushrooms)
{:potatoes 5}
```

1.2.13 Putting it all together

All these collections and types can be combined freely. As software engineers, we model the world by creating a particular representation of the problem in the program. Having a rich set of values at our disposal allows us to talk about complex problems. We might describe a person:

```
{:name "Amelia Earhart"
:birth 1897
:death 1939
:awards {"US" #{"Distinguished Flying Cross"
"National Women's Hall of Fame"}
"World" #{"Altitude record for Autogyro"
"First to cross Atlantic twice"{}}}
```

Or a recipe:

```
{:title "Chocolate chip cookies"
:ingredients {"flour" [(+ 2 1/4) :cup]
              "baking soda" [1 :teaspoon]
              "salt" [1 :teaspoon]
              "butter" [1 :cup]
              "sugar" [3/4 :cup]
              "brown sugar" [3/4 :cup]
              "vanilla" [1 :teaspoon]
              "eggs" 2
              "chocolate chips" [12 :ounce]}
```

Or the Gini coefficients of nations, as measured over time:

```
{"Afghanistan" {2008 27.8}
 "Indonesia" {2008 34.1 2010 35.6 2011 38.1}
 "Uruguay" {2008 46.3 2009 46.3 2010 45.3}}
```

In Clojure, we compose data structures to form more complex values; to talk about bigger ideas. We use operations like `first`, `nth`, `get`, and `contains?` to extract specific information from these structures, and modify them using `conj`, `disj`, `assoc`, `dissoc`, and so on.

We started this chapter with a discussion of types: groups of similar objects which obey the same rules. We learned that bigints, longs, ints, shorts, and bytes are all integers, that doubles and floats are approximations to decimal numbers, and that ratios represent fractions exactly. We learned the differences between strings for text, symbols as references, and keywords as short labels. Finally, we learned how to compose, alter, and inspect collections of elements. Armed with the basic nouns of Clojure, were ready to write a broad array of programs.

I'd like to conclude this tour with one last type of value. We've inspected dozens of types so far but what happens when you turn the camera on itself?

```
user=> (type type)
clojure.core$type
```

What is this type thing, exactly? What are these verbs we've been learning, and where do they come from? This is the central question of chapter three: functions.

Chapter 2

From ideas to implementation

2.1 Starting from Java

2.2 Extends and Implements

2.3 Understanding Persistent Vectors (Jean Lórange)

This is quoted from Lórange [L13].

You may or may not have heard about Clojure's persistent vectors. It is a data structure invented by Rich Hickey (influenced by Phil Bagwell's paper on Ideal Hash Trees) for Clojure, which gives practically $O(1)$ runtime for insert, update, lookups and subvec. As they are persistent, every modification creates a new vector instead of changing the old one.

So, how do they work? I'll try to explain them through a series of blogposts, in which we look at manageable parts each time. It will be a detailed explanation, with all the different oddities around the implementation as well. Consequently, this blog series may not be the perfect fit for people who want a "summary" on how persistent vectors work.

For today, we'll have a look at a naive first attempt, and will cover updates, insertion and popping (removal at the end).

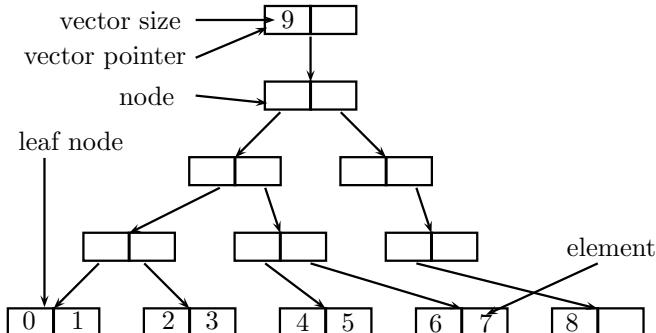
Note that this blogpost does not represent how `PersistentVector` is implemented: There are some speed optimizations, solutions for transients and other details which we will cover later. However, this serves as a basis for understanding how

they work, and the general idea behind the vector implementation.

The Basic Idea

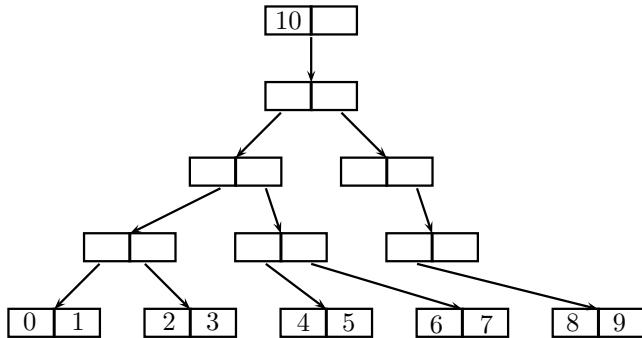
Mutable vectors and ArrayLists are generally just arrays which grows and shrinks when needed. This works great when you want mutability, but is a big problem when you want persistence. You get slow modification operations because you'll have to copy the whole array all the time, and it will use a lot of memory. It would be ideal to somehow avoid redundancy as much as possible without losing performance when looking up values, along with fast operations. That is exactly what Clojure's persistent vector does, and it is done through balanced, ordered trees.

The idea is to implement a structure which is similar to a binary tree. The only difference is that the interior nodes in the tree have a reference to at most two subnodes, and does not contain any elements themselves. The leaf nodes contain at most two elements. The elements are in order, which means that the first element is the first element in the leftmost leaf, and the last element is the rightmost element in the rightmost leaf. For now, we require that all leaf nodes are at the same depth. As an example, take a look at the tree below: It has the integers 0 to 8 in it, where 0 is the first element and 8 the last. The number 9 is the vector size:



Visualization of a vector with 9 elements in it.

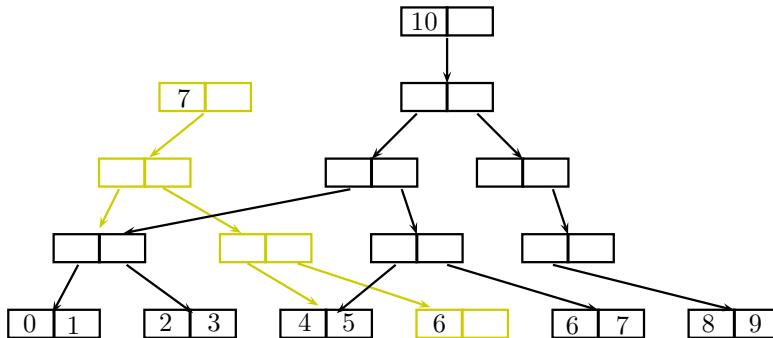
If we wanted to add a new element to the end of this vector and we were in the mutable world, we would insert 9 in the rightmost leaf node:



Visualization of a vector with 10 elements in it.

But here's the issue: We cannot do that if we want to be persistent. And this would obviously not work if we wanted to update an element! We would need to copy the whole structure, or at least parts of it.

To minimize copying while retaining full persistence, we perform path copying: We copy all nodes on the path down to the value we're about to update or insert, and replace the value with the new one when we're at the bottom. A result of multiple insertions is shown below. Here, the vector with 7 elements share structure with a vector with 10 elements:



A visualization of two vectors, which use structural sharing.

The pink coloured nodes and edges are shared between the vectors, whereas the brown and blue are separate. Other vectors not visualized may also share nodes with these vectors. Update

The easiest "modification" operator to understand would be updates/replacing values in a vector, so we will explain how updating works first. In Clojure, that's an assoc modification, or an update-in.

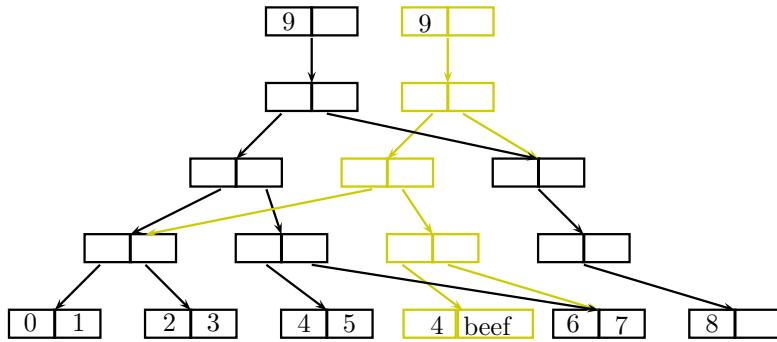
To update an element, we would have to walk the tree down to the leaf node where the element is placed. While we walk down, we copy the nodes on our path to ensure persistence. When we've gotten down to the leaf node, we copy it and

replace the value we wanted to replace with the new value. We then return the new vector with the modified path.

As an example, let's assume we perform an assoc on the vector with the elements from 0 to 8, like this:

```
(def brown [0 1 2 3 4 5 6 7 8])
(def blue (assoc brown 5 'beef))
```

The internal structures, where the blue one has the copied path, is shown below:



Two vectors, where we've updated a value.

Given that we have a way to know which node to go down, this seems easy enough. We'll go through how we find the path to a specific index in a later part of this series.

2.3.1 Insertion

Insertion is not too much different from an update, except that we have some edge cases where we have to generate nodes in order to fit in a value. We essentially have three cases:

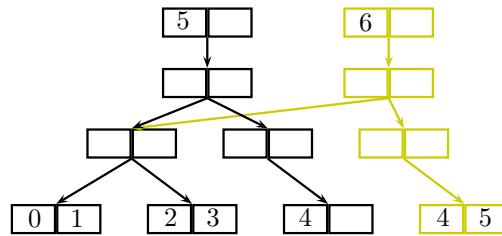
There is room for a new value in the rightmost leaf node. There is space in the root, but not in the rightmost leaf node. There is not enough space in the current root.

We'll go through them all, as their solutions are not that difficult to grasp.

Just Like Assoc

Whenever there is enough space in the rightmost leaf node, we can just do as we do when we perform an assoc: We just copy the path, and at the newly created leaf node, we put in the value to the right of the rightmost element.

As an example, here's how we would do (conj [0 1 2 3 4] 5), and the internal structures from doing so. black is the old:



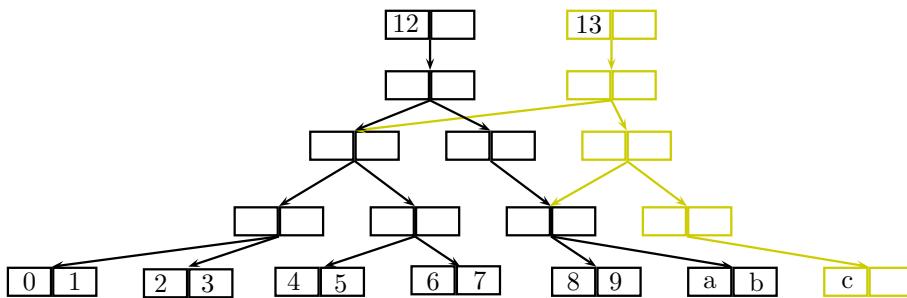
Insertion with enough space in the leaf node.

That's it. There's no magic, just path copying and insertion in the leaf node.

Generate Nodes When You Need Them

So, what do we do when there's not enough space in the leftmost leaf node? Luckily, we'll never end up in a position where we find out that we're in the wrong leaf node: We will always take the right path down to the leaf node.

Instead, we will realize that the node we're trying to go down to doesn't yet exist (the pointer is null). When a node doesn't exist, we generate one and set that one as the "copied" node.



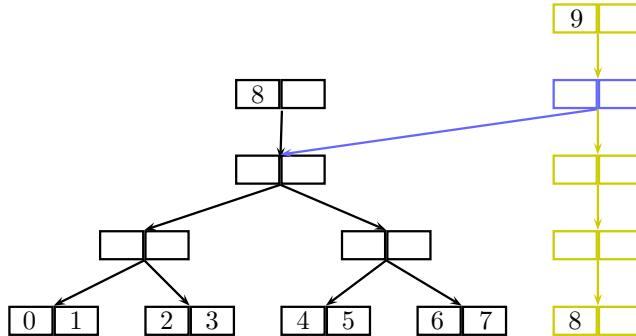
Insertion where we generate new nodes instead of copying.

In the figure above, the black nodes represent original nodes.

Root Overflow

The last case is the root overflow case. This happens when there isn't more space in the tree with the current root node.

It's not that difficult to understand how we would solve this: We make a new root node, and set the old root as the first child of the new root. From there on, we perform the node generating, just as we did in the previous solution.



Insertion where we generate a new root.

One thing is to solve the problem, but detecting when it happens is also important. Luckily, this is also rather easy. When we have a two-way branching vector, this happens when the old vector's size is a power of two. Generally speaking, an n-way branching vector will have an overflow when the size is a power of n.

2.3.2 Popping

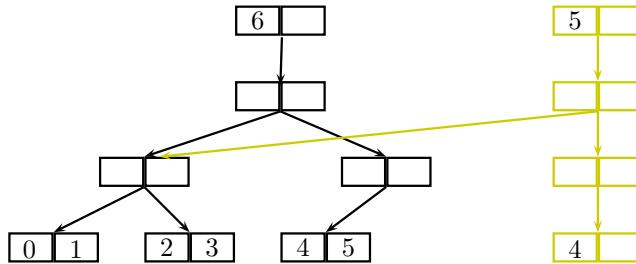
The solutions for popping (removing the last element) isn't that difficult to grasp either. Popping similar to inserting in that there are three cases:

The rightmost leaf node contains more than one element. The rightmost leaf node contains exactly one element (zero after popping). The root node contains exactly one element after popping.

Essentially, these are all ways of reverting 1, 2 and 3 in the previous section, neither of which are extremely complex.

Dissoc to the Rescue

Again, we have a case where we can just do as we do when we update a structure: We copy the path down to the rightmost leaf node, and remove the rightmost element in the copied leaf node. As long as there is at least one element left in the new leaf node, we don't have to do any magic.

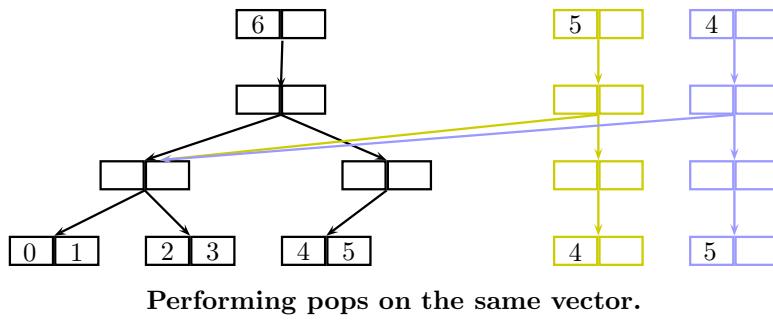


Popping a value from a vector with more than one element in the rightmost leaf node.

Keep in mind that popping multiple times on a vector will not yield identical vectors: They are equal, but they don't share the root. For instance,

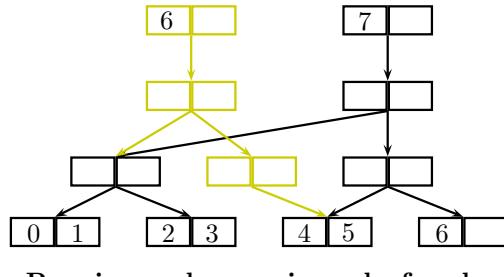
```
(def brown [0 1 2 3 4 5])
(def blue (pop brown))
(def green (pop brown))
```

will result in the following internal structure.



Removing Empty Nodes

Whenever we have a leaf node with only a single node, we have a different case. We would like to avoid empty nodes in our tree at all cost. Therefore, whenever we have an empty node, instead of returning it, we return null instead. The parent node will then contain a null pointer, instead of a pointer to an empty node:



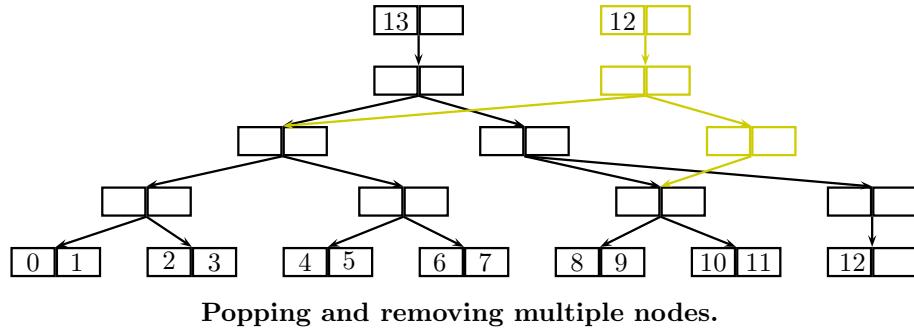
Here, the brown vector is the original, whereas the blue one is the popped one. Unfortunately, it is not as easy as to just remove leaf nodes. You see, if we return a null pointer to a node, which originally only had one child, we must convert that one into a null pointer which we send back: The results of emptying

a node propagates upwards. This is a bit tricky to get right, but essentially it works by looking at the new child, check if it is null and is supposed to be placed at index 0, and return null if that's the case.

If this was implemented in Clojure, it may look something like this recursive function:

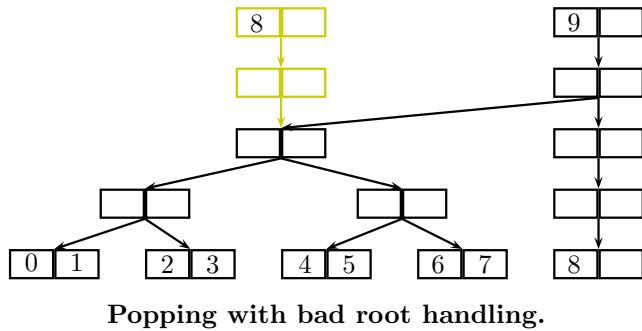
```
(defn node-pop [idx depth cur-node]
  (let [sub-idx (calculate-subindex idx depth)]
    (if (leaf-node? depth)
        (if (= sub-idx 0)
            nil
            (copy-and-remove cur-node sub-idx))
        ; not leaf node
        (let [child (node-pop idx (- depth 1)
                               (child-at cur-node sub-idx))]
          (if (nil? child)
              (if (= sub-idx 0)
                  nil
                  (copy-and-remove cur-node sub-idx))
              (copy-and-replace cur-node sub-idx child)))))))
```

When such a function has been implemented, node removal has been taken care of completely. As an example, see the graph below. Here, the popped (blue) vector has removed two nodes: The leaf node containing c and its parent.



Root Killing

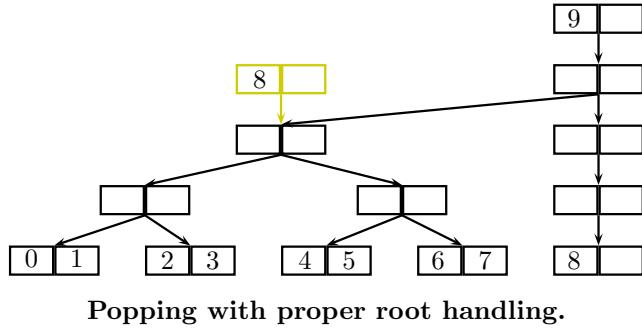
We have now covered all cases, except for one. With the current implementation, we would get the following result if we popped a vector with nine elements:



That's right, we'd have a root with a single pointer to a child node. Pretty useless, as we would always move down into the child when we lookup or assoc values, and inserting values would create a new root. What we would like to do is to get rid of it.

This is possibly the easiest thing in this blogpost to actually do: After we have finished popping, check if the root node contains only a single child (check that the second child is null, for instance). If that's the case, and the root node is not a leaf node, we can just replace the root node with its child.

The result is, as expected, a new vector (blue) with the first child of the original vector's root node as root:

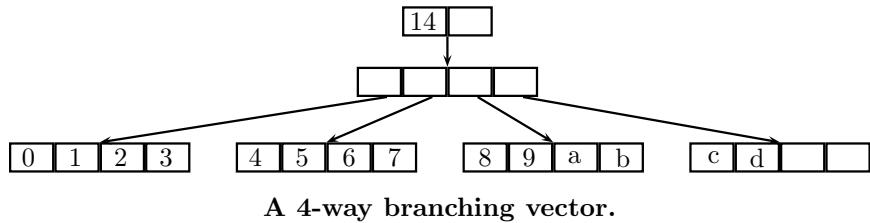


2.3.3 $O(1) \neq O(\log n)$

Some people out there are probably wondering how this can be said to be $O(1)$ at all. In fact, with only two children per node, this is $O(\log_2 n)$, which is (relatively) far from $O(1)$.

However, no one has said that we have to only have two children per node (often referred to as the branching factor). Clojure has 32 per node, which in result turns into very shallow trees. In fact, the trees will be at most 6 nodes deep if you have less than 1 billion elements in the vector. You need about 35 billion elements to get up to a depth of 8 nodes. At that point, I would believe memory consumption is a more serious issue.

To actually see the difference: Here is a 4-way branching tree with 14 elements, which only is 2 levels deep. If you scroll up a bit, you'll see a figure with two vectors containing 13 and 12 elements, respectively. With two-way branching, this is already 4 levels deep, double the height as this one.



As a result of the incredibly shallow trees, we tend to call modifications and lookup of Clojure vectors to be "effectively" constant time, although they in theory are $O(\log_{32} n)$. People with basic knowledge in big O notation know that this is exactly the same as $O(\log n)$, but for marketing reasons people like to add in the constant factor.

To understand how we pick the right branch, I think it's good to give the proper name of the structure and explain why it is named that. It sounds a bit weird to explain branching through the name of a data structure, but it makes sense when you consider that such a name may describe how it works. Naming

A more formal name for Clojure's persistent vector structure is persistent bit-partitioned vector trie. What I explained in the previous post was how persistent digit-partitioned vector tries work. Don't worry, a bit-partitioned one is just an optimized digit-partitioned trie, and in the previous post there is nothing different about them. In this one, there is a small difference related to performance. It has otherwise no practical differences.

I guess many of you don't know all of those words I mentioned in the above paragraph, so let's describe them, one by one.

2.3.4 Persistence

In the last post, I used the word persistent. I said we want to be "persistent", but didn't really explain what persistence itself really means.

A persistent data structure doesn't modify itself: Strictly speaking they don't have to be immutable internally, just have to be perceived as such. Whenever you do "updates", "inserts" and "removals" on a persistent data structure, you get a new data structure back. The old version will always be consistent, and whenever given some input in, it will always spit out the same output.

When we talk about a fully persistent data structure, all versions of a structure should be updateable, meaning that all possible operations you can do on a version can be performed on another. In early "functional data structure" time,

it was common to "cheat" with the structures and make the older versions "decay" over time by mutating the internals, making them slower and slower compared to the newer versions. However, Rich Hickey decided that all the versions of Clojure's persistent structures should have the same performance guarantees, regardless of which version of the structure you are using.

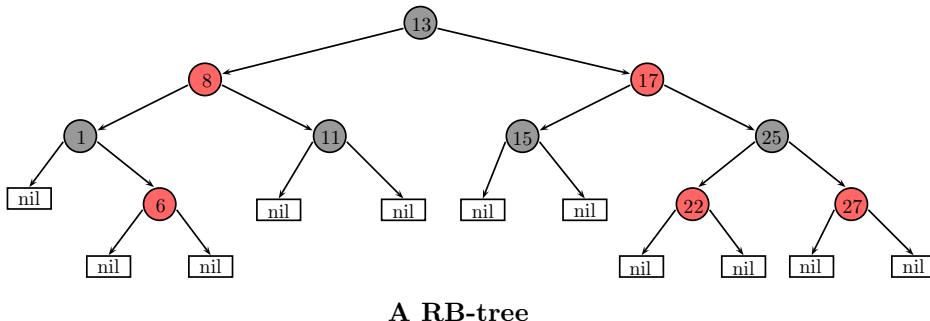
Vector

A vector is a one-dimensional growable array. C++'s `std::vector` and Java's `java.util.ArrayList` are examples of mutable implementations. There's not much more to it than that, really. A vector trie is a trie which represents a vector. It doesn't have to be persistent, but in our case, it is.

Trie

Tries are a specific type of trees, and I think it's best to show the actual difference by explaining the more known trees first.

In RB-trees and most other binary trees, mappings or elements are contained in the interior nodes. Picking the right branch is done by comparing the element/key at the current node: If the element is lower than the node element, we branch left, and if it is higher, we branch right. Leaves are usually null pointers/nil, and doesn't contain anything.

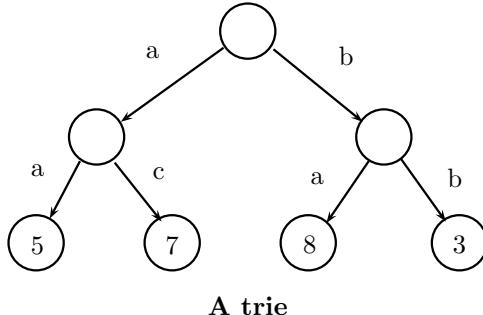


The RB-tree above is taken from Wikipedia's article on RB-trees. I'm not going to explain how those work in detail, but let us take a tiny example on how we check if 22 is contained in the RB-tree:

```
We start at the root, 13, and compare it with 22.
As 13 < 22, we go right.
The new node has 17 in it, and compare it with 22.
As 17 < 22, we still go right.
The next node we've walked into is 25.
As 25 > 22, we go left.
The next node is 22, so we know that 22 is contained in the tree.
```

If you want a good explanation on how RB-trees work, I would recommend Julianne Walker's Red Black Tree Tutorial.

A trie, on the other hand, has all the values stored in its leaves. Picking the right branch is done by using parts of the key as a lookup. Consequently, a trie may have more than two branches. In our case, we may have as many as 32!



An example of a general trie is illustrated in the figure above. That specific trie is a map: It takes a string of length two, and returns an integer represented by that string if it exists in the trie. ac has the value 7, whereas ba has the value 8. Here's how the trie works:

For strings, we split the string into characters. We then take the first character, find the edge represented by this value, and walk down that edge. If there is no edge for that value, we stop, as it is not contained in the trie. If not, we continue with the second character, and so on. Finally, when we are done, we return the value if it exists.

As an example, consider ac. We do as follows:

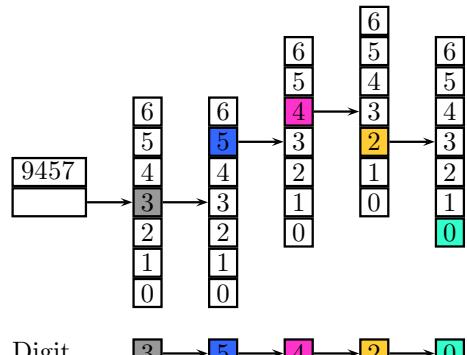
```
We split ac up into [a, c], and start at the root node.
We check if there is an edge in the node for a,
and there is: We follow it.
We check if there is an edge in the node for c,
and there is. We follow that as well.
We have no more characters left, which means the current node contains
our value, 7. We therefore return 7.
```

Clojure's Persistent Vector is a trie where the indices of elements are used as keys. But, as you may guess, we must split up the index integers in some way. To split up integers, we either use digit partitioning or its faster sibling, bit partitioning.

Digit Partitioning

Digit partitioning means that we split up the key into digits, which we then use as a basis for populating a trie. For instance, we can split up the key 9128 to [9, 1, 2, 8], and put an element into a trie based on that. We may have to pad with zeroes at the front of the list, if the depth of the trie is larger than the size of the list.

We can also use whatever base we would like, not just base 10. We would then have to convert the key to the base we wanted to use, and use the digits from the conversion. As an example, consider 9128 yet again. 9128 is 35420 in base 7, so we would have to use the list [3, 5, 4, 2, 0] for lookup/insertion in the trie.



Visualization of the 35420 lookup.

The trie (laying sideways, without the edges and nodes we're not walking) above shows how we traverse a digit-partitioned trie: We pick the most significant digit, in this case 3, and walk that specific branch. We continue with the second most significant digit in the same manner, until we have no digits left. When we've walked the last branch, the object we're standing with the first object in the rightmost array in this case is the object we wanted to look up.

Implementing such a lookup scheme is not too hard, if you know how to find the digits. Here's a Java version where everything not related to lookup is stripped away:

```
public class DigitTrie {
    public static final int RADIX = 7;

    // Array of objects. Can itself contain an array of objects.
    Object[] root;
    // The maximal size/length of a child node (1 if leaf node)
    int rDepth; // equivalent to RADIX ** (depth - 1)

    public Object lookup(int key) {
        Object[] node = this.root;

        // perform branching on internal nodes here
        for (int size = this.rDepth; size > 1; size /= RADIX) {
            node = (Object[]) node[(key / size) % RADIX];
            // If node may not exist, check if it is null here
        }

        // Last element is the value we want to lookup, return it.
    }
}
```

```

        return node[key % RADIX];
    }
}

```

The rDepth value represents the maximal size of a child of the root node: A number with n digits will have n to the power of RADIX possible values, and we must be able to put them all in the trie without having collisions.

In the for loop within the lookup method, the value size represents the maximal size a child of the current node can have. For each child we go over, that size is decremented by the branching factor, i.e. the radix or base of the digit trie.

The reason we're performing a modulo operation on the result is to ignore the more significant digits digits we've branched on earlier. We could potentially remove the higher digit from the key every time we branch into a child, but the code would be a tiny bit more complicated in that case. Bit Partitioning

Digit-partitioned tries would generally have to do a couple of integer divisions and modulo operations. Doing this is on every branch we must take is a bit time consuming. We would therefore like to speed this part up if it is possible.

So, as you may guess, bit-partitioned tries are a subset of the digit-partitioned tries. All digit-partitioned tries in a base which is a power of two (2, 4, 8, 16, 32, etc) can be turned into bit-partitioned ones. With some knowledge of bit manipulation, we can remove those costly arithmetic operations.

Conceptually, it works in the same way as digit partitioning does. However, instead of splitting the key into digits, we split it into chunks of bits with some predefined size. For 32-way branching tries, we need 5 bits in each part, and for 4-way branching tries, we need 2. In general, we need as many bits as the size of our exponent.

So, why is this faster? By using bit tricks, we can get rid of both integer division and modulo. If power is two to the power of n, we can use that

```

x / power == x >>> n and
x % power == x & (power - 1).

```

These formulas are just identities related to how integers are represented internally, namely as sequences of bits.

If we use this result and combine it with the previous implementation, we end up with the following code:

```

public class BitTrie {
    public static final int BITS = 5,
                           WIDTH = 1 << BITS, // 2^5 = 32
                           MASK = WIDTH - 1; // 31, or 0x1f

    // Array of objects. Can itself contain an array of objects.
    Object[] root;
}

```

```

// BITS times (the depth of this trie minus one).
int shift;

public Object lookup(int key) {
    Object[] node = this.root;

    // perform branching on internal nodes here
    for (int level = this.shift; level > 0; level -= BITS) {
        node = (Object[]) node[(key >> level) & MASK];
        // If node may not exist, check if it is null here
    }

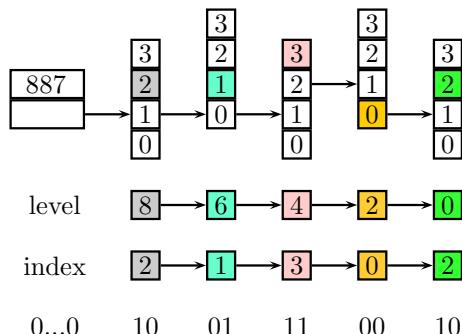
    // Last element is the value we want to lookup, return it.
    return node[key & MASK];
}
}

```

This is more or less exactly what Clojure's implementation is doing! See these lines of the Clojure code to verify it; The only difference is that it performs boundary checks and a tail check as well.

The important thing to note here is that we've not only changed the operators, but we've also replaced the rDepth value with a shift value. Instead of storing the whole value, we're only storing the exponent. This makes us able to use bitshifting on the key, which we use in the $(key \gg level)$ part. The other parts should be fairly straightforward to understand, given that one knows bit operations well. However, let's take an example for the ones unfamiliar with such tricks. The explanation is quite thorough, so feel to skip parts you understand.

Say we have only have 2 bit partitioning (4-way branching) instead of 5 bits (32-way) for visualization purposes. If we want to look up a value in a trie with 887 elements, we would have a shift equal to 8: All the children of the root node can contain at most $1 \gg 8 == 256$ elements each. The width and mask is also changed by the bit count: The mask will here be 3 instead of 31.



Visualization of the 626 lookup.

Say we want to look up the contents of the element with key 626. 626 in its

binary representation is 0000 0010 0111 0010. Following the algorithm, step by step, both written above and within Clojure's source code, we would have to do the following:

```

node is set up to be the root note, and level is set up to be 8.
As level is over 0, we start the for loop.
    We perform the operation key >>> level first. In this case,
        this is 636 >>> 8, which cuts away the first 8 bits: We're left
        with 0000 0010 or 2 in decimal.
    We perform the masking: (key >>> level) & MASK == 2 & 3. The
        masking sets all the bits except the first two to zero. This
        yields no difference here: We still have 2.
    We replace the current node with its child at index 2.
We decrement level by 2, and set it to 6.
As level is over 0, we continue the for loop.
    We again perform the operation key >>> level == 636 >>> 6. This
        cuts away the last 6 bits, and we're left with 0000 0010 01,
        or 9 in decimal.
    We perform the masking: (key >>> level) & MASK == 9 & 3,
        or 1001 & 0011 in binary. This shaves off the top 10,
        and we're left with 01, or 1 in decimal.
    We replace the current node with its child at index 1.
We decrement level by 2, and set it to 4.
As level is over 0, we continue the for loop
    Same trick here again. 636 >>> 4 leaves us with the bits 0010 0111.
    The mask sets all but the 2 first bits to zero, and we're left with 11.
    We replace the current node with its child at index 0b11 == 3.
We decrement level by 2, and set it to 2.
As level is over 0, we continue the for loop
    636 >>> 2 leaves us with 0010 0111 00.
    The 2 first bits are 0b00 == 0.
    We replace node with its first child, the one at index 0.
We decrement level by 2, and set it to 0.
As level is (finally) not over 0, we jump over the for loop.
We mask key with the mask, and get back the bits 10.
We return the contents at index 2 from node.

```

That is almost every single machine instruction you would have to perform to lookup a value in a Clojure vector, with a depth of 5. Such a vector would contain between 1 and 33 million elements. The fact that the shifts and masks are some of the most efficient operations on a modern CPU makes the whole deal even better. From a performance perspective, the only "pain point" left on lookups are the cache misses. For Clojure, that is handled pretty well by the JVM itself.

And that's how you do lookups in tries and in Clojure's vector implementation. I would guess the bit operations are the hardest one to grok, everything else is actually very straightforward. You just need a rough understanding on how tries work, and that's it!

2.4 Red Black Trees

There is a special kind of binary search tree called a red-black tree. It has the property that the longest path from the root to any leaf is no more than twice as long as the shortest path from the root to any other leaf.

In order to guarantee this property we construct a tree such that no two connected nodes are red. The longest possible path will alternate red and black nodes. The shortest path will be all black. Thus, the two paths will differ by at most twice the length.

There are various rules defined (see Okasaki [Oka98] p25 and Wikipedia [Wiki]). The Clojure implementation follows the Okasaki model, which we will use as our background. Okasaki gives two rules that are always true in a properly formed tree:

- No red node has a red child
- Every path from the root to an empty node contains the same number of black nodes.

2.4.1 Persistence

Driscoll defines a *persistent* data structure as one that supports multiple versions. A data structure that allows only a single version at a time is called *ephemeral* [DSST89].

The basic technique of creating a persistent data structure is to make copies. However this is expensive in time and space. Driscoll shows that data structures can be defined using **links** between **nodes** where the nodes carry information and the links connect the nodes. In such structures it is possible to only copy nodes that are changed and update the links. Using this technique does not modify the old data structure so the links can share portions of the old structure. Since the old structure is not changed it is also available at the same time as the new structure, albeit with different root nodes.

2.4.2 Persistent Red Black Trees

Okasaki has combined these two ideas to create Persistent Red Black Trees. Each modification of the tree is done by making copies of changed nodes but maintaining the red-black tree properties in the copy.

Since Clojure uses these persistent trees the actual tree is new copy, not a modified tree. We will now look into the details of how this is done.

2.4.3 Node Structure

First we have to mirror Driscoll's concept of a **Node**. This is created in the code with a new Java abstract class called `Node`. Okasaki requires that a node have 4 pieces of information, described as `T (Color, left, this, right)`. Thus we find that Nodes have the ability to manipulate their left and right members. Thus, for the left member we find:

- `left` – to return the left Node
- `addLeft` – to add a left Node
- `removeLeft` – to remove a left Node

and for the right member we find:

- `left` – to return the right Node
- `addLeft` – to add a right Node
- `removeLeft` – to remove a right Node

We have functions to set the color, `blacken` and `redden`. We can ask this node for its `key` and `val`

Since every node has to maintain the Red-Black balance we have two methods, `balanceLeft` and `balanceRight`. Note that these methods always return a black node as the root of the red-black tree is always black.

We also have a method to replace this node, the `replace` method.

Some of the methods are abstract which means that any class that implements a `Node` has to implement the following methods:

- `Node addLeft(Node ins);`
- `Node addRight(Node ins);`
- `Node removeLeft(Node del);`
- `Node removeRight(Node del);`
- `Node blacken();`
- `Node redden();`
- `Node replace(Object key, Object val, Node left, Node right);`

Since the data structure is persistent we need to copy rather than modify the structure. Each implementation of this abstract class has to handle the details.

Node assumes that it will return a Black node from its balance operations.

Nodes in a Clojure PersistentTreeMap have one of 8 possible subtypes of `Node`:

- `Black` - a black leaf node with a null value
- `BlackVal` - a black leaf node with a value
- `BlackBranch` - a black interior node with children and a null value

- BlackBranchVal - a black interior node with children and a value
- Red - a red leaf node with a null value
- RedVal - a red leaf node with a value
- RedBranch - a red interior node with children and a null value
- RedBranchVal - a red interior node with children and a value

(AMapEntry [527])
 — PersistentTreeMap Node Class —

```
static abstract class Node extends AMapEntry{
    final Object key;

    Node(Object key){
        this.key = key;
    }

    public Object key(){
        return key;
    }

    public Object val(){
        return null;
    }

    public Object getKey(){
        return key();
    }

    public Object getValue(){
        return val();
    }

    Node left(){
        return null;
    }

    Node right(){
        return null;
    }

    abstract Node addLeft(Node ins);

    abstract Node addRight(Node ins);

    abstract Node removeLeft(Node del);

    abstract Node removeRight(Node del);

    abstract Node blacken();
```

```

abstract Node redder();

Node balanceLeft(Node parent){
    return black(parent.key, parent.val(), this, parent.right());
}

Node balanceRight(Node parent){
    return black(parent.key, parent.val(), parent.left(), this);
}

abstract Node replace(Object key, Object val,
                      Node left, Node right);

}

```

2.4.4 The Black Node implementation

Since the Node class is abstract it has to have certain methods implemented in the extending class.

We have two types of implementations, one for Red nodes and one for Black nodes. Lets look at the Black node class. This is broken out into a set of subclasses and we will drill down the chain for implementation Black nodes.

The class *Black* extends *node* and implements the abstract methods of *Node*. This basically involves making sure that the subtrees maintain their balance under the operations of adding and deleting the subtrees.

Black nodes assume they are black but they implement a method to change their color. Since it is a copy operation it returns a new Red node as the result of *redder*.

Black nodes are the default color for a node so the Black class does not have to do anything special for color.

There are 4 possible Black node types

- Black - a black leaf node with a null value
- BlackVal - a black leaf node with a value
- BlackBranch - a black interior node with children and a null value
- BlackBranchVal - a black interior node with children and a value

A Black node is a leaf node with a null value. This is constructed by PersistentTreeMap's [45] *black* method.

(Node [41])

— PersistentTreeMap Black Class —

```

static class Black extends Node{
    public Black(Object key){
        super(key);
    }

    Node addLeft(Node ins){
        return ins.balanceLeft(this);
    }

    Node addRight(Node ins){
        return ins.balanceRight(this);
    }

    Node removeLeft(Node del){
        return balanceLeftDel(key, val(), del, right());
    }

    Node removeRight(Node del){
        return balanceRightDel(key, val(), left(), del);
    }

    Node blacken(){
        return this;
    }

    Node redder(){
        return new Red(key);
    }

    Node replace(Object key, Object val, Node left, Node right){
        return black(key, val, left, right);
    }
}

```

A BlackVal node is a leaf node with a value. This is constructed by PersistentTreeMap's [45] black method.

(Black [42])

— PersistentTreeMap BlackVal Class —

```

static class BlackVal extends Black{
    final Object val;

    public BlackVal(Object key, Object val){
        super(key);
        this.val = val;
    }
}

```

```

    }

    public Object val(){
        return val;
    }

    Node redder(){
        return new RedVal(key, val);
    }

}

```

Interior nodes of Clojure's Red Black trees can have a value. A BlackBranch node is a leaf node with a null value and children, one of which could possibly be null. This is constructed by PersistentTreeMap's [45] black method.

(Black [42])

— PersistentTreeMap BlackBranch Class —

```

static class BlackBranch extends Black{
    final Node left;

    final Node right;

    public BlackBranch(Object key, Node left, Node right){
        super(key);
        this.left = left;
        this.right = right;
    }

    public Node left(){
        return left;
    }

    public Node right(){
        return right;
    }

    Node redder(){
        return new RedBranch(key, left, right);
    }

}

```

A BlackBranchVal node is a leaf node with a value and children, one of which

could possibly be null. This is constructed by PersistentTreeMap's [45] black method.

(BlackBranch [44])

— PersistentTreeMap BlackBranchVal Class —

```
static class BlackBranchVal extends BlackBranch{
    final Object val;

    public BlackBranchVal(Object key, Object val,
                          Node left, Node right){
        super(key, left, right);
        this.val = val;
    }

    public Object val(){
        return val;
    }

    Node redder(){
        return new RedBranchVal(key, val, left, right);
    }
}
```

—————

2.4.5 Constructing a Black Node

We can construct a new black node given 4 parameters:

- key – the TreeMap key
- val – the value stored in the TreeMap for this key
- left – a Node, possibly null
- right – a Node, possibly null

If both of the children are null we must be constructing a leaf node. If the value is null we need only construct a naked *Black* node, otherwise we construct a *BlackVal* node and store both the key and the value.

If either of the children exist but have no value for this node then construct a *BlackBranch* internal tree node pointing at the children.

If we have all four parameters then we have an internal node that also has a value associated with it. Save the value and the children in a *BlackBranchVal* node.

— PersistentTreeMap black method —

```

static Black black(Object key, Object val, Node left, Node right){
    if(left == null && right == null)
    {
        if(val == null)
            return new Black(key);
        return new BlackVal(key, val);
    }
    if(val == null)
        return new BlackBranch(key, left, right);
    return new BlackBranchVal(key, val, left, right);
}

```

2.4.6 The Red Node implementation

The Red Node class differs from the Black node in many ways. In particular, most of the operations return a new Red object whereas the Black node implementation balances the subtrees.

Black nodes are the default color for a node so the Red class has to return Red nodes for most operations.

There are 4 possible Red node types

- Red - a black leaf node with a null value
- RedVal - a black leaf node with a value
- RedBranch - a black interior node with children and a null value
- RedBranchVal - a black interior node with children and a value

A Red node is a leaf node with a null value. This is constructed by PersistentTreeMap's [50] red method.

(Node [41])

— PersistentTreeMap Red Class —

```

static class Red extends Node{
    public Red(Object key){
        super(key);
    }

    Node addLeft(Node ins){
        return red(key, val(), ins, right());
    }

    Node addRight(Node ins){
        return red(key, val(), left(), ins);
    }
}

```

```

Node removeLeft(Node del){
    return red(key, val(), del, right());
}

Node removeRight(Node del){
    return red(key, val(), left(), del);
}

Node blacken(){
    return new Black(key);
}

Node redden(){
    throw new UnsupportedOperationException("Invariant violation");
}

Node replace(Object key, Object val, Node left, Node right){
    return red(key, val, left, right);
}

}

```

A RedVal node is a leaf node with a value. This is constructed by PersistentTreeMap's [50] red method.

(Red [46])

— PersistentTreeMap RedVal Class —

```

static class RedVal extends Red{
    final Object val;

    public RedVal(Object key, Object val){
        super(key);
        this.val = val;
    }

    public Object val(){
        return val;
    }

    Node blacken(){
        return new BlackVal(key, val);
    }
}

```

Interior nodes of Clojure's Red Black trees can have a value. A RedBranch node is a leaf node with a null value and children, one of which could possibly be null. This is constructed by PersistentTreeMap's [50] red method.

(Red [46])

— PersistentTreeMap RedBranch Class —

```
static class RedBranch extends Red{
    final Node left;
    final Node right;

    public RedBranch(Object key, Node left, Node right){
        super(key);
        this.left = left;
        this.right = right;
    }

    public Node left(){
        return left;
    }

    public Node right(){
        return right;
    }

    Node balanceLeft(Node parent){
        if(left instanceof Red)
            return red(key, val(), left.blacken(),
                      black(parent.key, parent.val(),
                            right, parent.right())));
        else if(right instanceof Red)
            return red(right.key, right.val(),
                      black(key, val(), left, right.left()),
                      black(parent.key, parent.val(),
                            right.right(), parent.right())));
        else
            return super.balanceLeft(parent);
    }

    Node balanceRight(Node parent){
        if(right instanceof Red)
            return red(key, val(),
                      black(parent.key, parent.val(),
                            parent.left(), left),
                      right.blacken());
        else if(left instanceof Red)
            return red(left.key, left.val(),
                      black(parent.key, parent.val(),
                            right,
                            blacken())));
        else
            return super.balanceRight(parent);
    }
}
```

```

        parent.left(), left.left()),
        black(key, val(), left.right(), right));
    else
        return super.balanceRight(parent);
}

Node blacken(){
    return new BlackBranch(key, left, right);
}

}

```

A RedBranchVal node is a leaf node with a value and children, one of which could possibly be null. This is constructed by PersistentTreeMap's [50] red method.

(RedBranch [48])

— PersistentTreeMap RedBranchVal Class —

```

static class RedBranchVal extends RedBranch{
    final Object val;

    public RedBranchVal(Object key, Object val, Node left, Node right){
        super(key, left, right);
        this.val = val;
    }

    public Object val(){
        return val;
    }

    Node blacken(){
        return new BlackBranchVal(key, val, left, right);
    }
}

```

2.4.7 Constructing a Red Node

We can construct a new red node given 4 parameters:

- key – the TreeMap key
- val – the value stored in the TreeMap for this key
- left – a Node, possibly null
- right – a Node, possibly null

If both of the children are null we must be constructing a leaf node. If the value is null we need only construct a naked *Red* node, otherwise we construct a *RedVal* node and store both the key and the value.

If either of the children exist but have no value for this node then construct a *RedBranch* internal tree node pointing at the children.

If we have all four parameters then we have an internal node that also has a value associated with it. Save the value and the children in a *RedBranchVal* node.

— PersistentTreeMap red method —

```
static Red red(Object key, Object val, Node left, Node right){
    if(left == null && right == null)
    {
        if(val == null)
            return new Red(key);
        return new RedVal(key, val);
    }
    if(val == null)
        return new RedBranch(key, left, right);
    return new RedBranchVal(key, val, left, right);
}
```

2.4.8 Okasaki's balance cases

Okasaki distinguishes 4 cases of balancing. It turns out that all four cases balance to the same tree, which is hardly a surprise given the constraints on the red-black alternation.

ML [Wiki1, Har05] is a functional programming language and is used by Okasaki to express his tree transformations.

Each case shows the ML code and the associated diagram. The ML code represents a node as a 4-tuple of type **T**. Thus we see a node as

```
Type ( Color, leftnode, thisnode, rightnode )
```

so we can read the first case as:

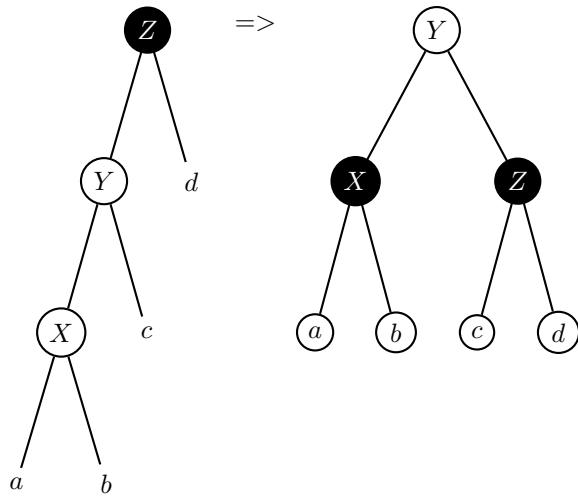
```
Node ( Black,
       Node ( Red ,
              Node ( Red, Node a, Node X, Node b ),
              Node Y,
              Node c
            )
      )
```

```

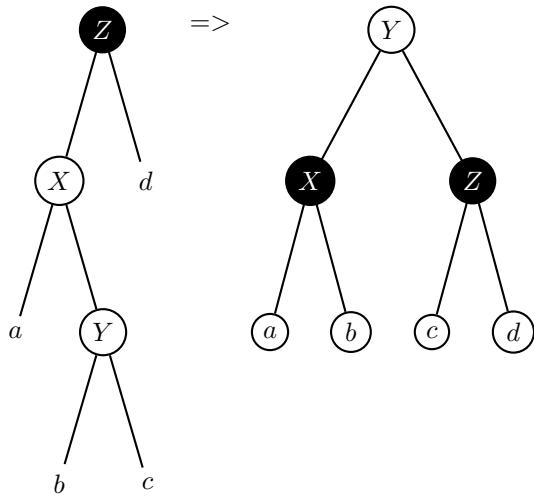
Node Z
Node d
)

```

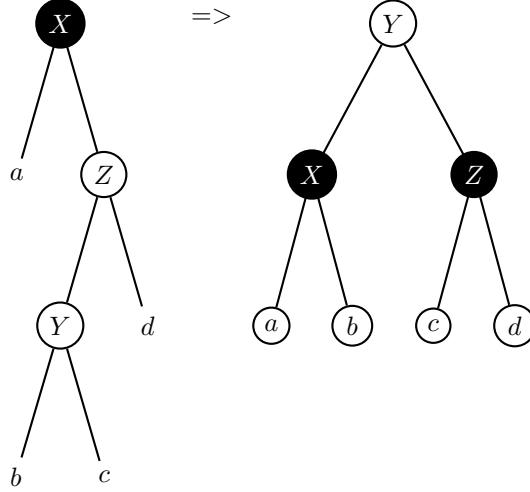
Case 1: $(B, T(R, T(R, a, x, b), y, c), z, d) = T(R, T(B, a, x, b), y, T(B, c, z, d))$



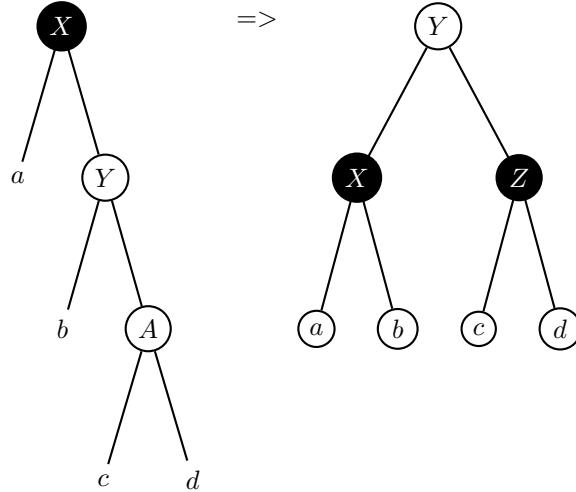
Case 2: $(B, T(R, a, x, T(R, b, y, c)), z, d) = T(R, T(B, a, x, b), y, T(B, c, z, d))$



Case 3: $(B, a, x, T(R, T(R, b, y, c), z, d)) = T(R, T(B, a, x, b), y, T(B, c, z, d))$



Case 4: $(B, a, x, T(R, b, y, T(R, c, z, d))) = T(R, T(B, a, x, b), y, T(B, c, z, d))$



2.4.9 Deleting a Node

Here we are being asked to remove the Node *del*. This request is either a call to a black node's [42] *removeRight* method or PersistentTreeMap's [1000] *remove* method.

— PersistentTreeMap **balanceRightDel** method —

```
static Node balanceRightDel(Object key, Object val,
```

```

        Node left, Node del){
    if(del instanceof Red)
        return red(key, val, left, del.blacken());
    else if(left instanceof Black)
        return leftBalance(key, val, left.redden(), del);
    else if(left instanceof Red && left.right() instanceof Black)
        return red(left.right().key, left.right().val(),
                  leftBalance(left.key, left.val(),
                               left.left().redden(),
                               left.right().left()),
                  black(key, val, left.right().right(), del));
    else
        throw new UnsupportedOperationException("Invariant violation");
}

```

— PersistentTreeMap balanceLeftDel method —

```

static Node balanceLeftDel(Object key, Object val,
                           Node del, Node right){
    if(del instanceof Red)
        return red(key, val, del.blacken(), right);
    else if(right instanceof Black)
        return rightBalance(key, val, del, right.redden());
    else if(right instanceof Red && right.left() instanceof Black)
        return red(right.left().key, right.left().val(),
                  black(key, val, del, right.left().left()),
                  rightBalance(right.key, right.val(),
                               right.left().right(),
                               right.right().redden()));
    else
        throw new UnsupportedOperationException("Invariant violation");
}

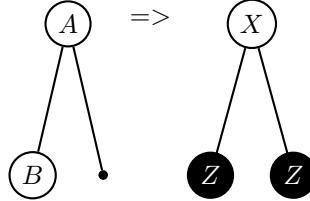
```

2.4.10 Clojure's balance cases

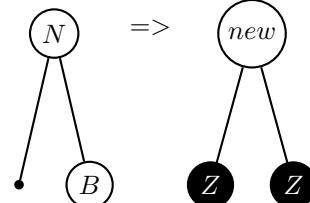
We find the *leftBalance* handling the case where a red node has a red node as its left child. There are three cases.

Case 1: The left child of a red node is red. We return new nodes X, Y, and Z where

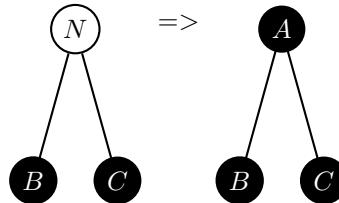
- X – Node (Red,
- Y
- Z – Node (Black,



Case 2: The right child of a red node is red. We rewrite that as:



Case 3: Both children of a red node are black. Rewrite the node black.



— PersistentTreeMap leftBalance method —

```
static Node leftBalance(Object key, Object val,
                      Node ins, Node right){
    if(ins instanceof Red && ins.left() instanceof Red)
        return red(ins.key, ins.val(),
                  ins.left().blacken(),
                  black(key, val, ins.right(), right));
    else if(ins instanceof Red && ins.right() instanceof Red)
        return red(ins.right().key, ins.right().val(),
                  black(ins.key, ins.val(),
                        ins.left(), ins.right().left()),
                  black(key, val, ins.right().right(), right));
    else
        return black(key, val, ins, right);
}
```

— PersistentTreeMap rightBalance method —

```

static Node rightBalance(Object key, Object val, Node left, Node ins){
    if(ins instanceof Red && ins.right() instanceof Red)
        return red(ins.key, ins.val(),
                   black(key, val, left, ins.left()),
                   ins.right().blacken());
    else if(ins instanceof Red && ins.left() instanceof Red)
        return red(ins.left().key, ins.left().val(),
                   black(key, val, left, ins.left().left()),
                   black(ins.key, ins.val(),
                         ins.left().right(), ins.right()));
    else
        return black(key, val, left, ins);
}

```

2.4.11 Replacing a Node

— PersistentTreeMap replace method —

```

Node replace(Node t, Object key, Object val){
    int c = doCompare(key, t.key);
    return t.replace(t.key,
                     c == 0 ? val : t.val(),
                     c < 0 ? replace(t.left(), key, val) : t.left(),
                     c > 0 ? replace(t.right(), key, val) : t.right());
}

```

[1000] PersistentTreeMap implements several subclasses to support the interoperability with the rest of Clojure. These will not be explained here. Refer to the other sections of this document for more details. In particular, see [103] Iterators for methods of iterating across keys and values. See [60] Seqs for information about manipulating sequences over tree maps.

2.5 Immutable Data Structures

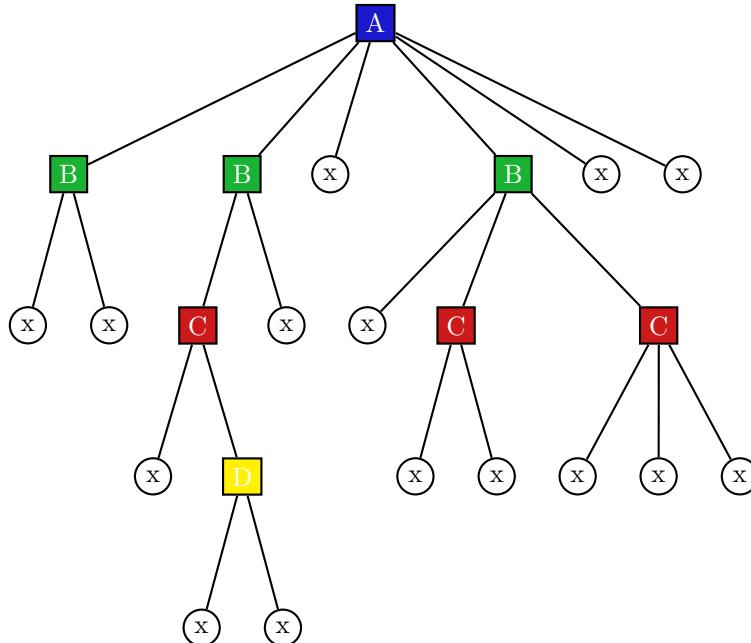
2.6 Bit-Partitioned Hash Tries

A trie is a data structure that exploits the fact that a prefix or suffix has some regular pattern[Bag00]. For instance, credit card numbers have a fixed format. We can look at the set of 16 digit credit card numbers which have the pattern AAAA-BBBB-CCCC-DDDD. Given this sequence of 4 digit fields we can create a trie data structure for fast lookup.

If we created a tree structure where the first level of lookup was the first 4 digits AAAA we would have a maximum of 9999 possible first level branches. If the second layer of the tree were based on BBBB then it would also have a branching factor of 9999. We eventually get a tree at most 4 layers deep that handle all 10^{16} possible values.

Alternatively we can reduce the fan-out by choosing 2 digits, so AA gives a fan-out of 99 and this happens at each level. Now the tree is 8 layers deep in order to contain all possible nodes.

Clojure has chosen a strategy based a bit pattern. So if we look at the bits in groups of five we get this kind of a trie:



Notice that the trie has a fan-out of 2⁵ or 32 at each level except the last. To contain all possible 32 bit values requires a trie of at most depth 7. This means that a lookup will take at most 7 possible fetches. For any reasonable set of numbers this is a near constant time lookup.

One advantage of this scheme is the ability to mask out a subset of the word, fully right-shift the bits, and use the resulting number as an index into the array. Any masked off value has to be between 0 and 31 because the mask is 5 bits wide.

The 5 bit mask function is part of [969] PersistentHashMap. Given a number of bits to shift right (a multiple of 5 appears to be used everywhere) this routine returns a number between 0 and 31.

— PersistentHashMap mask method —

```
static int mask(int hash, int shift){  
    //return ((hash << shift) >>> 27); // & 0x01f;  
    return (hash >>> shift) & 0x01f;  
}
```

This is used in the `ArrayNode` class. For instance, the `find` method

- uses mask to get an index from 5 bits
- uses that value to index into an array
- if that entry is not set we didn't find it
- shift 5 more bits right and search again.

— **ArrayNode find Object method** —

```
public Object find(int shift, int hash, Object key,
                  Object notFound){
    int idx = mask(hash, shift);
    INode node = array[idx];
    if(node == null)
        return notFound;
    return node.find(shift + 5, hash, key, notFound);
}
```

—————

Note that ArrayNode is a container class so it is actually recursively calling the *find* method on whatever was found in that container slot. It might be anything that implements the INode interface: (Serializable [1723])

— **PersistentHashMap INode interface** —

```
static interface INode extends Serializable {
    INode assoc(int shift, int hash, Object key,
               Object val, Box addedLeaf);

    INode without(int shift, int hash, Object key);

    IMapEntry find(int shift, int hash, Object key);

    Object find(int shift, int hash, Object key, Object notFound);

    ISeq nodeSeq();

    INode assoc(AtomicReference<Thread> edit, int shift, int hash,
               Object key, Object val, Box addedLeaf);

    INode without(AtomicReference<Thread> edit, int shift, int hash,
                 Object key, Box removedLeaf);
}
```

—————

There are 3 classes that implement INode.

- [955] ArrayNode

- [959] BitmapIndexedNode
- [965] HashCollisionNode

2.6.1 Lists

2.6.2 Vectors

2.6.3 Hashmaps

Maps are associations of keywords and data. Hashmaps are untyped data structures, as opposed to Records which carry their type.

```
> (def tim {:fname "Tim"
             :lname "Daly"
             :address {:street "Long Ridge Road"
                       :city "Pittsburgh"
                       :state "PA"
                       :zip 15022}})
#'user/tim
```

We can look up by keyword since keywords are

```
> (:lname tim)
"Daly"
```

We can nest access using the “thread first” operator which threads arbitrary pieces of code (as opposed to “..” which threads Java classes).

```
> (-> tim :address :city)
"Pittsburgh"
```

We can create a new object with a change to a field.

```
> (assoc tim :fname "Timothy")
{:lname "Daly",
 :address {:state "PA",
           :city "Pittsburgh",
           :street "Long Ridge Road",
           :zip 15022},
 :fname "Timothy"}
```

The `update-in` function threads into a nested data structure and then applies a function, in this case the `inc` function. This reduces the need to create intermediate classes.

```
> (update-in tim [:address :zip] inc)
{:lname "Daly",
:address {:state "PA",
:city "Pittsburgh",
:street "Long Ridge Road",
:zip 15023},
:fname "Tim"}
```

2.6.4 Seqs

(ASeq [571])
— PersistentTreeMap Seq Class —

```
static public class Seq extends ASeq{
    final ISeq stack;
    final boolean asc;
    final int cnt;

    public Seq(ISeq stack, boolean asc){
        this.stack = stack;
        this.asc = asc;
        this.cnt = -1;
    }

    public Seq(ISeq stack, boolean asc, int cnt){
        this.stack = stack;
        this.asc = asc;
        this.cnt = cnt;
    }

    Seq(IPersistentMap meta, ISeq stack, boolean asc, int cnt){
        super(meta);
        this.stack = stack;
        this.asc = asc;
        this.cnt = cnt;
    }

    static Seq create(Node t, boolean asc, int cnt){
        return new Seq(push(t, null, asc), asc, cnt);
    }

    static ISeq push(Node t, ISeq stack, boolean asc){
        while(t != null)
        {
            stack = RT.cons(t, stack);
            t = asc ? t.left() : t.right();
        }
        return stack;
    }
}
```

```

    }

    public Object first(){
        return stack.first();
    }

    public ISeq next(){
        Node t = (Node) stack.first();
        ISeq nextstack =
            push(asc ? t.right() : t.left(), stack.next(), asc);
        if(nextstack != null)
        {
            return new Seq(nextstack, asc, cnt - 1);
        }
        return null;
    }

    public int count(){
        if(cnt < 0)
            return super.count();
        return cnt;
    }

    public Obj withMeta(IPersistentMap meta){
        return new Seq(meta, stack, asc, cnt);
    }
}

```

2.7 Records

Records carry their type, in this case “Person”, as opposed to hashmaps which are untyped data structures.

```
> (defrecord Person [fname lname address])
user.Person
```

and in this case “Address”

```
> (defrecord Address [street city state zip])
user.Address
```

We can create a Var reference to a new record and set the address field to be a new Address record.

```
> (def tim
  (Person. "Tim" "Daly"
    (Address. "Long Ridge Road" "Pittsburgh" "PA" 15022)))
#'user/tim
```

Record field names are functions so we can use them as accessors.

```
> (:lname tim)
"Daly"
```

And we can access them with the “thread first” operator which threads through the set of accessors and then applies the function, in this case, the `inc` function. This is similar to the “..” operator except that it applies to any kind of operator, not just Java operators.

```
> (-> tim :address :city)
"Pittsburgh"
```

```
> (assoc tim :fname "Timothy")
#:user.Person{:fname "Timothy",
:lname "Daly",
:address #:user.Address{
:street "Long Ridge Road",
:city "Pittsburgh",
:state "PA",
:zip 15022}}
```

```
> (update-in tim [:address :zip] inc)
#:user.Person{:fname "Tim",
:lname "Daly",
:address #:user.Address{
:street "Long Ridge Road",
:city "Pittsburgh",
:state "PA",
:zip 15023}}
```

2.8 Java Interoperability

2.8.1 Language primitives

Atomic data types[Hal11]

type	Clojure	Java
string	"foo"	String
character	\f	Character
regular expression	#"fo*"	Pattern
integer	42	Long
big integer	42N	BigInteger
double	3.14159	Double
big decimal	3.141519M	BigDecimal
boolean	true	Boolean
nil	nil	null
ratio	22/7	N/A
symbol	foo	N/A
keyword	:foo ::foo	N/A

Data literals

These data structures can contain anything of any type.

type	property	example
list	singly-linked, insert at front	(1 2 3)
vector	indexed, insert at rear	[1 2 3]
map	key-value pairs	{:a 10 :b 20}
set	key	#{1 2 3}

People make claims that Lisp has no syntax . . . A more accurate thing to say is that Lisp has the syntax of its data structures and that syntax has been built upon to write code.

—Stuart Halloway “Clojure-Java Interp (Jan 19, 2011)”

Function Calling

```
(println "Hello Wikileaks")
```

This is a list data structure with a first element of a Symbol and a second element of a String. The first element is the function to be called and the second element is the argument to that function.

Function Definition

```
(defn funname
  "documentation string of the function"
  [vector of arguments]
  (function arg1 ... argN))
```

This is a list data structure with the elements:

- defn, a Symbol, used as a function to define a function
- funname, a Symbol, used as the name of the new function
- docstring, a String, used to document the function
- argvec, a Vector of arguments to the new function
- function, a Symbol, used as the name of some other function to call
- arg1...argN, a list of arguments to the function

For example,

```
> (defn say "says the name and age" [name age] (str name " " age))
 #'user/say
```

The defn function creates the new function called "say" which takes 2 arguments and returns a string with the two arguments separated by a space as in:

```
> (say "Tim" 24)
"Tim 24"
```

You can add meta information to your programs to improve the performance or simplify the runtime lookup of classes. The syntax for metadata is a Symbol or hashmap prefixed by a caret ^ . For example, the say function returns a String so we could write

```
(defn ^String say
  "says the name and age"
  [name age]
  (str name " " age))
```

If the item after the caret is a single Symbol it is expanded into a hashmap with the keyword :tag and the Symbol. In our case, this is {:tag String}

2.8.2 Clojure calling Java

Java “new” in Clojure

```
new MyClass("arg1")
```

becomes

```
(MyClass. "arg1")
```

Accessing a Java static member

```
Math.PI
```

becomes

Math/PI

Accessing instance members

Since Clojure shifts the focus from the object to the function we call, the Clojure syntax favors putting the function at the head of the list, the so-called “function” position.

`rnd.nextInt()`

becomes

`(.nextInt rnd)`

Chaining function calls

`person.getAddress().getZip()`

becomes

`(.. person getAddress getZip)`

The `doto` macro uses the first form to create an object upon which subsequent method calls can be applied in series.

```
(doto (JFrame. "argument")
  (.add (proxy [ JPanel ] []))
  (.setSize 640 480)
  (.setVisible true))
```

2.9 Recursion

2.10 Argument Deconstruction

2.11 The Read-Eval-Print Loop

2.12 Special Forms

2.13 Reader Macros

2.14 Namespaces

Namespaces map to java packages.

2.15 Dynamic Comopilation

2.16 Ahead-Of-Time Comopilation

2.17 Lazy Evaluation

2.18 Metadata

2.19 Concurrency

2.20 Identity and State

Identity is a logical entity associated with a series of causally related values, called states, over time. Identity is not a name but can be named (e.g. my “Mom” is not your mom). Identity can be a composite object (e.g. the New York Yankees). Programs that are processes need identity. [Hic10]

State is the value of an identity at a given time. Variables are not sufficient to refer to a state because state could be composite. This leads to a situation where, at a given time, the variable does not refer to a properly formed state but refers to inconsistent composite information. There is a lack of time coordination.

In order to properly capture state we need to incorporate some notion of time. Once we have time as a marker we can talk about state at a particular time. Since Clojure deals with immutable objects that implies that a changed state must be a different object since things don’t change in place.

Since state is a combination of information and time it is possible for multiple observers to query the state. Each will see a consistent view of the information (the value) based on the time of the observation.

Clojure manages state by creating an intermediate object called a reference. The reference object has atomic semantics. Changing a reference happens as an all-or-nothing event. References are essentially a pointer to a stateful object. The reference points at the current state. When a new state is being constructed as a function of the old state there is no way to reference it until the function completes. At that time the reference is atomically updated and the new state can be observed as a wholly-formed object. This becomes the new state which is a function of the current time and the new information. Observers of the prior state will still see the old state unchanged.

There are four kinds of reference objects in Clojure which we will discuss in detail below. These are *vars*, *atoms*, *refs*, and *agents*.

2.21 Software Transactional Memory

2.22 Symbols

Internally, a Clojure Symbol contains four pieces of data

- **ns** an interned string of the namespace for this symbol
- **name** a interned string for this symbol name
- **hash** a hash of the name and the namespace
- **_meta** meta data associated with this Symbol

— Symbol private data —

```
//these must be interned strings!
final String ns;
final String name;
final int hash;
final IPersistentMap _meta;
```

Java's String class has an *intern* method [Ora11] which returns a canonical representation for the string object. The String class maintains a pool of strings. When String's *intern* method is invoked the pool is searched for an *equals* match. If the search succeeds then a reference to the stored string is returned, otherwise the new string is added to the pool and a reference to this String object is returned. Thus, the String *name* of a Symbol is a unique reference.

As pointed out in [Web11], Clojure creates a new Symbol for each occurrence it reads. The Symbols are not unique despite having equal names:

```
node2: clj
Clojure 1.3.0-alpha4
user=> (identical? 'a 'a)
false
user=> (= 'a 'a)
true
user=> (identical? (clojure.lang.Symbol/intern "a")
                    (clojure.lang.Symbol/intern "a"))
false
```

— Symbol intern method —

```
static public Symbol intern(String nsname){
    int i = nsname.lastIndexOf('/');
```

```

if(i == -1 || nsname.equals("/"))
    return new Symbol(null, nsname.intern());
else
    return
        new Symbol(nsname.substring(0, i).intern(),
                   nsname.substring(i + 1).intern());
}

```

A new Symbol is automatically prepended with its namespace.

```

user=> (def a 1)
#'user/a

```

— Symbol intern method 2 —

```

static public Symbol intern(String ns, String name){
    return new Symbol(ns == null ? null : ns.intern(), name.intern());
}

```

Since the namespace was null the current namespace `user` was used. We can see this from the `toString` result.

— Symbol method `toString` —

```

public String toString(){
    if(ns != null)
        return ns + "/" + name;
    return name;
}

```

Since Symbols optionally include a namespace `user/a`, which includes a namespace, and `a` which does not include a namespace, are different symbols but they name the same variable.

```

user=> (= 'user/a 'a)
false
user=> (def a 1)
#'user/a
user=> user/a
1

```

```
user=> (def user/a 2)
#'user/a
user=> a
2
```

— Symbol method equals —

```
public boolean equals(Object o){
    if(this == o)
        return true;
    if(!(o instanceof Symbol))
        return false;

    Symbol symbol = (Symbol) o;

    //identity compares intended, names are interned
    return name == symbol.name && ns == symbol.ns;
}
```

—————

2.23 The Lisp Reader

When the LispReader read function is invoked it reads each character as an integer. The read function is parsing S-expressions so it skips whitespace and fails if it does not get a complete S-expression. Numbers are handled specially by the reader.

— LispReader read method —

```
static public Object read(PushbackReader r,
                         boolean eofIsError,
                         Object eofValue,
                         boolean isRecursive)
    throws Exception{

    try
    {
        for(; ;)
        {
            int ch = r.read();

            while(isWhitespace(ch))
                ch = r.read();

            if(ch == -1)
```

```

{
    if(eofIsError)
        throw new Exception("EOF while reading");
    return eofValue;
}

if(Character.isDigit(ch))
{
    Object n = readNumber(r, (char) ch);
    if(RT.suppressRead())
        return null;
    return n;
}

IFn macroFn = getMacro(ch);
if(macroFn != null)
{
    Object ret = macroFn.invoke(r, (char) ch);
    if(RT.suppressRead())
        return null;
    //no op macros return the reader
    if(ret == r)
        continue;
    return ret;
}

if(ch == '+' || ch == '-')
{
    int ch2 = r.read();
    if(Character.isDigit(ch2))
    {
        unread(r, ch2);
        Object n = readNumber(r, (char) ch);
        if(RT.suppressRead())
            return null;
        return n;
    }
    unread(r, ch2);
}

String token = readToken(r, (char) ch);
if(RT.suppressRead())
    return null;
return interpretToken(token);
}
}

catch(Exception e)
{
    if(isRecursive || !(r instanceof LineNumberingPushbackReader))
        throw e;
}

```

```

    LineNumberingPushbackReader rdr =
        (LineNumberingPushbackReader) r;
    //throw new Exception(String.format("ReaderError:(%d,1) %s",
    //        rdr.getLineNumber(), e.getMessage()), e);
    throw new ReaderException(rdr.getLineNumber(), e);
}

```

2.23.1 Reader Syntax Macros

Almost every special character is handled by looking up the special function associated with that character in the `macros` array:

— LispReader macros statement —

```
static IFn[] macros = new IFn[256];
```

The default values are specified in the syntax macro table. These are all class objects that implement the [774] IFn interface which requires that they have an `invoke` method.

For the special case of the # character there is a separate table, the [84] Dispatch Macro Table which reads the next character and calls another dispatch function. All of the special characters get their meaning from these tables.

By default the special syntax characters are

- " a [72] StringReader
- ; a [74] CommentReader
- ' a [74] WrappingReader(QUOTE)
- @ a [74] WrappingReader(DEREF)
- ^ a [75] MetaReader
- ` a [76] SyntaxQuoteReader
- ~ a [79] UnquoteReader
- (a [80] ListReader
-) a [80] UnmatchedDelimiterReader
- [a [80] VectorReader
-] a [80] UnmatchedDelimiterReader
- { a [81] MapReader
- } a [80] UnmatchedDelimiterReader
- \ a [81] CharacterReader
- % a [82] ArgReader

- # a [83] DispatchReader

— LispReader Syntax Macro Table —

```

macros[','] = new StringReader();
macros[';'] = new CommentReader();
macros['``'] = new WrappingReader(QUOTE);
macros['@'] = new WrappingReader(DEREF); //new Dereference();
macros['^'] = new MetaReader();
macros['`'] = new SyntaxQuoteReader();
macros['~'] = new UnquoteReader();
macros['('] = new ListReader();
macros[')'] = new UnmatchedDelimiterReader();
macros['['] = new VectorReader();
macros[''] = new UnmatchedDelimiterReader();
macros['{'] = new MapReader();
macros['}'] = new UnmatchedDelimiterReader();
// macros['|'] = new ArgVectorReader();
macros['\\'] = new CharacterReader();
macros['%'] = new ArgReader();
\getchunk{LispReader sharpsign macro statement}

```

2.23.2 The String reader macro

(AFn [509])

— LispReader StringReader class —

```

public static class StringReader extends AFn{
    public Object invoke(Object reader, Object doublequote)
        throws Exception{
        StringBuilder sb = new StringBuilder();
        Reader r = (Reader) reader;

        for(int ch = r.read(); ch != '''; ch = r.read())
        {
            if(ch == -1)
                throw new Exception("EOF while reading string");
            if(ch == '\\')    //escape
            {
                ch = r.read();
                if(ch == -1)
                    throw new Exception("EOF while reading string");
                switch(ch)
                {
                    case 't':

```

```

        ch = '\t';
        break;
    case 'r':
        ch = '\r';
        break;
    case 'n':
        ch = '\n';
        break;
    case '\\':
        break;
    case '"':
        break;
    case 'b':
        ch = '\b';
        break;
    case 'f':
        ch = '\f';
        break;
    case 'u':
    {
        ch = r.read();
        if (Character.digit(ch, 16) == -1)
            throw new Exception(
                "Invalid unicode escape: \\" + (char) ch);
        ch =
            readUnicodeChar((PushbackReader) r, ch, 16, 4, true);
        break;
    }
    default:
    {
        if(Character.isDigit(ch))
        {
            ch =
                readUnicodeChar(
                    (PushbackReader) r, ch, 8, 3, false);
            if(ch > 0377)
                throw new Exception(
                    "Octal escape sequence must be in range [0, 377].");
            }
        else
            throw new Exception(
                "Unsupported escape character: \\" + (char) ch);
        }
    }
    sb.append((char) ch);
}
return sb.toString();
}
}
```

2.23.3 The Comment reader macro

(AFn [509])
— LispReader CommentReader class —

```
public static class CommentReader extends AFn{
    public Object invoke(Object reader, Object semicolon)
        throws Exception{
        Reader r = (Reader) reader;
        int ch;
        do
            {
                ch = r.read();
            } while(ch != -1 && ch != '\n' && ch != '\r');
        return r;
    }
}
```

2.23.4 The Wrapping reader macro

(AFn [509])
— LispReader WrappingReader class —

```
public static class WrappingReader extends AFn{
    final Symbol sym;

    public WrappingReader(Symbol sym){
        this.sym = sym;
    }

    public Object invoke(Object reader, Object quote)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        Object o = read(r, true, null, true);
        return RT.list(sym, o);
    }
}
```

2.23.5 The Meta reader macro

(AFn [509])

— LispReader MetaReader class —

```
public static class MetaReader extends AFn{
    public Object invoke(Object reader, Object caret)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        int line = -1;
        if(r instanceof LineNumberingPushbackReader)
            line = ((LineNumberingPushbackReader) r).getLineNumber();
        Object meta = read(r, true, null, true);
        if(meta instanceof Symbol || meta instanceof String)
            meta = RT.map(RT.TAG_KEY, meta);
        else if (meta instanceof Keyword)
            meta = RT.map(meta, RT.T);
        else if(!(meta instanceof IPersistentMap))
            throw new IllegalArgumentException(
                "Metadata must be Symbol,Keyword,String or Map");

        Object o = read(r, true, null, true);
        if(o instanceof IMeta)
        {
            if(line != -1 && o instanceof ISeq)
                meta = ((IPersistentMap) meta).assoc(RT.LINE_KEY, line);
            if(o instanceof IReference)
            {
                ((IReference)o).resetMeta((IPersistentMap) meta);
                return o;
            }
            Object omeca = RT.meta(o);
            for(ISeq s = RT.seq(meta); s != null; s = s.next()) {
                IMapEntry kv = (IMapEntry) s.first();
                omeca = RT.assoc(omeca, kv.getKey(), kv.getValue());
            }
            return ((IObj) o).withMeta((IPersistentMap) omeca);
        }
        else
            throw new IllegalArgumentException(
                "Metadata can only be applied to IMetas");
    }
}
```

2.23.6 The SyntaxQuote reader macro

(AFn [509])

— LispReader SyntaxQuoteReader class —

```
public static class SyntaxQuoteReader extends AFn{
    public Object invoke(Object reader, Object backquote)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        try
        {
            Var.pushThreadBindings(
                RT.map(GENSYM_ENV, PersistentHashMap.EMPTY));

            Object form = read(r, true, null, true);
            return syntaxQuote(form);
        }
        finally
        {
            Var.popThreadBindings();
        }
    }

    static Object syntaxQuote(Object form) throws Exception{
        Object ret;
        if(Compiler.isSpecial(form))
            ret = RT.list(Compiler.QUOTE, form);
        else if(form instanceof Symbol)
        {
            Symbol sym = (Symbol) form;
            if(sym.ns == null && sym.name.endsWith("#"))
            {
                IPersistentMap gmap =
                    (IPersistentMap) GENSYM_ENV.deref();
                if(gmap == null)
                    throw new IllegalStateException(
                        "Gensym literal not in syntax-quote");
                Symbol gs = (Symbol) gmap.valAt(sym);
                if(gs == null)
                    GENSYM_ENV.set(
                        gmap.assoc(sym,
                            gs = Symbol.intern(null,
                                sym.name.substring(0, sym.name.length() - 1)
                                + "--" + RT.nextID() + "__auto__")));
                sym = gs;
            }
        else if(sym.ns == null && sym.name.endsWith("."))
```

```

        sym.name.substring(0, sym.name.length() - 1));
        csym = Compiler.resolveSymbol(csym);
        sym = Symbol.intern(null, csym.name.concat("."));
    }
    else if(sym.ns == null && sym.name.startsWith(".")) {
        // Simply quote method names.
    }
    else {
        Object maybeClass = null;
        if(sym.ns != null)
            maybeClass = Compiler.currentNS().getMapping(
                Symbol.intern(null, sym.ns));
        if(maybeClass instanceof Class)
            {
                // Classname/foo ->
                // package.qualified.Classname/foo
                sym = Symbol.intern(
                    ((Class)maybeClass).getName(), sym.name);
            }
        else
            sym = Compiler.resolveSymbol(sym);
    }
    ret = RT.list(Compiler.QUOTE, sym);
}
else if(isUnquote(form))
    return RT.second(form);
else if(isUnquoteSplicing(form))
    throw new IllegalStateException("splice not in list");
else if(form instanceof IPersistentCollection)
{
    if(form instanceof IPersistentMap)
    {
        IPersistentVector keyvals = flattenMap(form);
        ret = RT.list(APPLY, HASHMAP,
            RT.list(SEQ,
                RT.cons(CONCAT, sqExpandList(keyvals.seq()))));
    }
    else if(form instanceof IPersistentVector)
    {
        ret = RT.list(APPLY, VECTOR,
            RT.list(SEQ,
                RT.cons(CONCAT,
                    sqExpandList(
                        (IPersistentVector) form).seq()))));
    }
    else if(form instanceof IPersistentSet)
    {
        ret = RT.list(APPLY, HASHSET,

```

```

        RT.list(SEQ,
        RT.cons(CONCAT,
        sqExpandList(
        ((IPersistentSet) form).seq()))));
    }
else if(form instanceof ISeq ||
        form instanceof IPersistentList)
{
    ISeq seq = RT.seq(form);
    if(seq == null)
        ret = RT.cons(LIST,null);
    else
        ret =
            RT.list(SEQ, RT.cons(CONCAT, sqExpandList(seq)));
}
else
    throw new UnsupportedOperationException(
        "Unknown Collection type");
}
else if(form instanceof Keyword
        || form instanceof Number
        || form instanceof Character
        || form instanceof String)
    ret = form;
else
    ret = RT.list(Compiler.QUOTE, form);

if(form instanceof IObj && RT.meta(form) != null)
{
    //filter line numbers
    IPersistentMap newMeta =
        ((IObj) form).meta().without(RT.LINE_KEY);
    if(newMeta.count() > 0)
        return
            RT.list(WITH_META, ret,
                syntaxQuote(((IObj) form).meta()));
    }
    return ret;
}

private static ISeq sqExpandList(ISeq seq) throws Exception{
    PersistentVector ret = PersistentVector.EMPTY;
    for(; seq != null; seq = seq.next())
    {
        Object item = seq.first();
        if(isUnquote(item))
            ret = ret.cons(RT.list(LIST, RT.second(item)));
        else if(isUnquoteSplicing(item))
            ret = ret.cons(RT.second(item));
        else

```

```

        ret = ret.cons(RT.list(LIST, syntaxQuote(item)));
    }
    return ret.seq();
}

private static IPersistentVector flattenMap(Object form){
    IPersistentVector keyvals = PersistentVector.EMPTY;
    for(ISeq s = RT.seq(form); s != null; s = s.next())
    {
        IMapEntry e = (IMapEntry) s.first();
        keyvals = (IPersistentVector) keyvals.cons(e.key());
        keyvals = (IPersistentVector) keyvals.cons(e.val());
    }
    return keyvals;
}

```

2.23.7 The Unquote reader macro

(AFn [509])

— LispReader UnquoteReader class —

```

static class UnquoteReader extends AFn{
    public Object invoke(Object reader, Object comma) throws Exception{
        PushbackReader r = (PushbackReader) reader;
        int ch = r.read();
        if(ch == -1)
            throw new Exception("EOF while reading character");
        if(ch == '@')
        {
            Object o = read(r, true, null, true);
            return RT.list(UNQUOTE_SPLICING, o);
        }
        else
        {
            unread(r, ch);
            Object o = read(r, true, null, true);
            return RT.list(UNQUOTE, o);
        }
    }
}

```

2.23.8 The List reader macro

(AFn [509])

— LispReader ListReader class —

```

public static class ListReader extends AFn{
    public Object invoke(Object reader, Object leftparen)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        int line = -1;
        if(r instanceof LineNumberingPushbackReader)
            line = ((LineNumberingPushbackReader) r).getLineNumber();
        List list = readDelimitedList(')', r, true);
        if(list.isEmpty())
            return PersistentList.EMPTY;
        IObj s = (IObj) PersistentList.create(list);
        //      IObj s = (IObj) RT.seq(list);
        if(line != -1)
            return s.withMeta(RT.map(RT.LINE_KEY, line));
        else
            return s;
    }
}

```

2.23.9 The Unmatched Delimiter reader macro

(AFn [509])

— LispReader UnmatchedDelimiterReader class —

```

public static class UnmatchedDelimiterReader extends AFn{
    public Object invoke(Object reader, Object rightdelim)
        throws Exception{
        throw new Exception("Unmatched delimiter: " + rightdelim);
    }
}

```

2.23.10 The Vector reader macro

(AFn [509])

— LispReader VectorReader class —

```

public static class VectorReader extends AFn{
    public Object invoke(Object reader, Object leftparen)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        return LazilyPersistentVector
            .create(readDelimitedList(']', r, true));
    }
}

```

2.23.11 The Map reader macro

(AFn [509])
— LispReader MapReader class —

```

public static class MapReader extends AFn{
    public Object invoke(Object reader, Object leftparen)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        return RT.map(readDelimitedList('}', r, true).toArray());
    }
}

```

2.23.12 The Character reader macro

(AFn [509])
— LispReader CharacterReader class —

```

public static class CharacterReader extends AFn{
    public Object invoke(Object reader, Object backslash)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        int ch = r.read();
        if(ch == -1)
            throw new Exception("EOF while reading character");
        String token = readToken(r, (char) ch);
        if(token.length() == 1)
            return Character.valueOf(token.charAt(0));
        else if(token.equals("newline"))
            return '\n';
        else if(token.equals("space"))
            return ' ';
    }
}

```

```

        else if(token.equals("tab"))
            return '\t';
        else if(token.equals("backspace"))
            return '\b';
        else if(token.equals("formfeed"))
            return '\f';
        else if(token.equals("return"))
            return '\r';
        else if(token.startsWith("u"))
        {
            char c = (char) readUnicodeChar(token, 1, 4, 16);
            if(c >= '\uD800' && c <= '\uDFFF') // surrogate code unit?
                throw new Exception(
                    "Invalid character constant: \\u" +
                    Integer.toHexString(c, 16));
            return c;
        }
        else if(token.startsWith("o"))
        {
            int len = token.length() - 1;
            if(len > 3)
                throw new Exception(
                    "Invalid octal escape sequence length: " + len);
            int uc = readUnicodeChar(token, 1, len, 8);
            if(uc > 0377)
                throw new Exception(
                    "Octal escape sequence must be in range [0, 377].");
            return (char) uc;
        }
        throw new Exception("Unsupported character: \\\" + token);
    }
}

```

2.23.13 The Arg reader macro

(AFn [509])
— LispReader ArgReader class —

```

static class ArgReader extends AFn{
    public Object invoke(Object reader, Object pct) throws Exception{
        PushbackReader r = (PushbackReader) reader;
        if(ARG_ENV.deref() == null)
        {
            return interpretToken(readToken(r, '%'));
        }
        int ch = r.read();

```

```

unread(r, ch);
//% alone is first arg
if(ch == -1 || isWhitespace(ch) || isTerminatingMacro(ch))
{
    return registerArg(1);
}
Object n = read(r, true, null, true);
if(n.equals(Compiler._AMP_))
    return registerArg(-1);
if(!(n instanceof Number))
    throw new IllegalStateException(
        "arg literal must be %, %& or %integer");
return registerArg(((Number) n).intValue());
}
}

```

—————

2.24 Reader Dispatch Macros

The [825] LispReader read function, when it encounters a # character indexes into the [72] Syntax Macro Table to find an entry initialized to a [83] DispatchReader object.

— LispReader sharpsign macro statement —

```
macros['#'] = new DispatchReader();
```

—————

A DispatchReader reads the next character and uses it as an index into the [84] dispatchMacros array. The reader function associated with the macro character, in this case, is a [85] VarReader. This gets assigned to fn and then calls the invoke method of the [85] VarReader class.

(AFn [509])

— LispReader DispatchReader class —

```

public static class DispatchReader extends AFn{
    public Object invoke(Object reader, Object hash)
        throws Exception{
        int ch = ((Reader) reader).read();
        if(ch == -1)
            throw new Exception("EOF while reading character");
        IFn fn = dispatchMacros[ch];
        if(fn == null)
            throw new Exception(

```

```

        String.format("No dispatch macro for: %c", (char) ch));
    return fn.invoke(reader, ch);
}
}

```

The dispatchMacros array is defined to be an array of IFn objects. The [774] IFn interface requires that all implementations have to have an `invoke` method, which VarReader has.

— **LispReader dispatchMacros statement** —

```
static IFn[] dispatchMacros = new IFn[256];
```

By default the special dispatch characters are:

- ` a [75] MetaReader, for attaching meta information to objects
- ' a [85] VarReader, returns a Var object
- " a [85] RegexReader, returns a regular expression
- (a [86] FnReader, returns a function object
- { a [87] SetReader, returns a set object
- = a [87] EvalReader, returns an evalued object
- ! a [74] CommentReader, returns a commented object
- < a [89] UnreadableReader, returns an unreadable object
- _ a [89] DiscardReader, returns a discarded object

These are all defined in this table and they are explained in the following sections.

— **LispReader Dispatch Macro Table** —

```

dispatchMacros['^'] = new MetaReader();
\getchunk{LispReader dispatchMacros VarReader statement}
dispatchMacros['"'] = new RegexReader();
dispatchMacros['('] = new FnReader();
dispatchMacros['{'] = new SetReader();
dispatchMacros['='] = new EvalReader();
dispatchMacros['!'] = new CommentReader();
dispatchMacros['<'] = new UnreadableReader();
dispatchMacros['_'] = new DiscardReader();

```

2.24.1 The Var reader macro

Vars have a special reader macro #' so that the input form #'foo expands into (var foo). The Symbol foo must be a Var and the Var object itself, not its value,

is returned.

The single-quote (') array entry contains a [85] VarReader function.

— **LispReader dispatchMacros VarReader statement** —

```
dispatchMacros['\'' = new VarReader();
```

—————

The VarReader *invoke* method reads the next object and returns the Var reference. (AFn [509])

— **LispReader VarReader class** —

```
public static class VarReader extends AFn{
    public Object invoke(Object reader, Object quote)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        Object o = read(r, true, null, true);
        //      if(o instanceof Symbol)
        //      {
        //          Object v =
        //              Compiler.maybeResolveIn(Compiler.currentNS(),
        //              (Symbol) o);
        //          if(v instanceof Var)
        //              return v;
        //      }
        return RT.list(THE_VAR, o);
    }
}
```

—————

The #' macro is useful for naming functions or variables in another namespace. You can call function across namespaces by writing

```
(#'mynamespace/functionname ...)
```

2.24.2 The Regular Expression reader macro

(AFn [509])

— **LispReader RegexReader class** —

```
public static class RegexReader extends AFn{
    static StringReader stringrdr = new StringReader();

    public Object invoke(Object reader, Object doublequote)
        throws Exception{
        StringBuilder sb = new StringBuilder();
        Reader r = (Reader) reader;
```

```
        for(int ch = r.read(); ch != '''; ch = r.read())
        {
            if(ch == -1)
                throw new Exception("EOF while reading regex");
            sb.append( (char) ch );
            if(ch == '\\\\')      //escape
            {
                ch = r.read();
                if(ch == -1)
                    throw new Exception("EOF while reading regex");
                sb.append( (char) ch ) ;
            }
        }
        return Pattern.compile(sb.toString());
    }
}
```

2.24.3 The Function reader macro

(AFn [509])

— LispReader FnReader class —

```

        {
            Object sym = argsyms.valAt(i);
            if(sym == null)
                sym = garg(i);
            args = args.cons(sym);
        }
    }
    Object restsym = argsyms.valAt(-1);
    if(restsym != null)
    {
        args = args.cons(Compiler._AMP_);
        args = args.cons(restsym);
    }
}
return RT.list(Compiler.FN, args, form);
}
finally
{
    Var.popThreadBindings();
}
}
}
}

```

2.24.4 The Set reader macro

(AFn [509])
— LispReader SetReader class —

```

public static class SetReader extends AFn{
    public Object invoke(Object reader, Object leftbracket)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        return PersistentHashSet
            .createWithCheck(readDelimitedList('}', r, true));
    }
}

```

2.24.5 The Eval reader macro

(AFn [509])
— LispReader EvalReader class —

```
public static class EvalReader extends AFn{
```

```

public Object invoke(Object reader, Object eq) throws Exception{
    if (!RT.booleanCast(RT.READEVAL.deref()))
    {
        throw new Exception(
            "EvalReader not allowed when *read-eval* is false.");
    }

    PushbackReader r = (PushbackReader) reader;
    Object o = read(r, true, null, true);
    if(o instanceof Symbol)
    {
        return RT.className(o.toString());
    }
    else if(o instanceof IPersistentList)
    {
        Symbol fs = (Symbol) RT.first(o);
        if(fs.equals(THE_VAR))
        {
            Symbol vs = (Symbol) RT.second(o);
            //Compiler.resolve((Symbol) RT.second(o),true);
            return RT.var(vs.ns, vs.name);
        }
        if(fs.name.endsWith("."))

        {
            Object[] args = RT.toArray(RT.next(o));
            return
                Reflector.invokeConstructor(
                    RT.className(
                        fs.name.substring(0, fs.name.length() - 1)),
                    args);
        }
        if(Compiler.namesStaticMember(fs))
        {
            Object[] args = RT.toArray(RT.next(o));
            return
                Reflector.invokeStaticMethod(
                    fs.ns, fs.name, args);
        }
        Object v =
            Compiler.maybeResolveIn(Compiler.currentNS(), fs);
        if(v instanceof Var)
        {
            return ((IFn) v).applyTo(RT.next(o));
        }
        throw new Exception("Can't resolve " + fs);
    }
    else
        throw new IllegalArgumentException("Unsupported #= form");
}
}

```

2.24.6 The Unreadable reader macro

(AFn [509])
— LispReader UnreadableReader class —

```
public static class UnreadableReader extends AFn{
    public Object invoke(Object reader, Object leftangle)
        throws Exception{
        throw new Exception("Unreadable form");
    }
}
```

2.24.7 The Discard reader macro

(AFn [509])
— LispReader DiscardReader class —

```
public static class DiscardReader extends AFn{
    public Object invoke(Object reader, Object underscore)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        read(r, true, null, true);
        return r;
    }
}
```

2.25 Vars

2.26 Transients

Transients are intended to make your code faster. Transients ignore structural sharing so time and space is saved. The underlying data structure is different.

For example, we thread a persistent map through some associations.

— transients example —

```
(-> {} (assoc :a 1) (assoc :b 2) (assoc :c 3))
```

which results in the persistent map

— **transients example** —

```
{:c 3, :b 2, :a 1}
```

A **transient** version is similar but transients are not mutable.

— **transients example** —

```
(-> (transient {}) (assoc! :a 1) (assoc! :b 2) (assoc! :c 3) persistent!)
```

— **defn transient** —

```
(defn transient
  "Alpha - subject to change.
  Returns a new, transient version of the collection, in constant time."
  {:added "1.1"
   :static true}
  [^clojure.lang.IEditableCollection coll]
  (.asTransient coll))
```

The difference is that the transient call makes the persistent map is transformed to a transient map. The **assoc!** call operates similar to assoc. The **persistent!** call transforms the result to a persistent map.

There are several function that operate on transients, **persistent!**, **conj!**, **assoc!**, **dissoc!**, **pop!**, and **disj!**.

— **defn persistent!** —

```
(defn persistent!
  "Alpha - subject to change.
  Returns a new, persistent version of the transient collection, in
  constant time. The transient collection cannot be used after this
  call, any such use will throw an exception."
  {:added "1.1"
   :static true}
  [^clojure.lang.ITransientCollection coll]
  (.persistent coll))
```

— defn conj! —

```
(defn conj!
  "Alpha - subject to change.
  Adds x to the transient collection, and return coll. The 'addition'
  may happen at different 'places' depending on the concrete type."
  {:added "1.1"
   :static true}
  [^clojure.lang.ITransientCollection coll x]
  (.conj coll x))
```

— defn assoc! —

```
(defn assoc!
  "Alpha - subject to change.
  When applied to a transient map, adds mapping of key(s) to
  val(s). When applied to a transient vector, sets the val at index.
  Note - index must be <= (count vector). Returns coll."
  {:added "1.1"
   :static true}
  ([^clojure.lang.ITransientAssociative coll key val]
   (.assoc coll key val))
  ([^clojure.lang.ITransientAssociative coll key val & kvs]
   (let [ret (.assoc coll key val)]
     (if kvs
       (recur ret (first kvs) (second kvs) (nnnext kvs))
       ret))))
```

— defn dissoc! —

```
(defn dissoc!
  "Alpha - subject to change.
  Returns a transient map that doesn't contain a mapping for key(s)."
  {:added "1.1"
   :static true}
  ([^clojure.lang.ITransientMap map key] (.without map key))
  ([^clojure.lang.ITransientMap map key & ks]
   (let [ret (.without map key)]
     (if ks
```

```
(recur ret (first ks) (next ks))
ret))))
```

— defn pop! —

```
(defn pop!
  "Alpha - subject to change.
  Removes the last item from a transient vector. If
  the collection is empty, throws an exception. Returns coll"
  {:added "1.1"
   :static true}
  [^clojure.lang.ITransientVector coll]
  (.pop coll))
```

— defn disj! —

```
(defn disj!
  "Alpha - subject to change.
  disj[oin]. Returns a transient set of the same (hashed/sorted) type,
  that does not contain key(s)."
  {:added "1.1"
   :static true}
  ([set] set)
  ([^clojure.lang.ITransientSet set key]
   (. set (disjoin key)))
  ([set key & ks]
   (let [ret (disj set key)]
     (if ks
       (recur ret (first ks) (next ks))
       ret))))
```

The imperative style will not work everywhere. For instance,

— transients example —

```
(let [a (transient {})]
  (dotimes [i 20] (assoc! a i i))
  (persistent! a))
```

returns with the broken result

— transients example —

```
(0 0, 1 1, 2 2, 3 3, 4 4, 5 5, 6 6, 7 7)
```

—————

which is only the first 8 values of the map.

However, if it is done in a recursive style, as in

— transients example —

```
(persistent!
  (reduce (fn[m v] (assoc! m v v))
    (transient {})
    (range 1 21)))
```

—————

yields all twenty pairs.

— transients example —

```
(0 0, 1 1, 2 2, 3 3, 4 4, 5 5, 6 6, 7 7, 8 8, 9 9, 10 10, 11 11, 12 12,
13 13, 14 14, 15 15, 16 16, 17 17, 18 18, 19 19, 20 20)
```

—————

Transients can only be modified by the thread that creates them. Attempting to access them will throw a `java.util.concurrent.ExecutionException`

— transients example —

```
@(let [c (transient [])] (future (conj! c :a)))
```

—————

It is also not possible to modify a Transient after the call to `persistent!` will throw a `java.lang.IllegalAccessError` exception. So this will not work:

— transients example —

```
(let [c (transient #{})] (persistent! c) (conj! c :a))
```

—————

Transients do not work on every data structure. Lists, sets, and maps all throw a `java.lang.ClassCastException`

— transients example —

```
(transient '())
(transient (sorted-set))
(transient (sorted-map))
```

The Transient data structure must support `clojure.lang.IEditableCollection`.

2.27 Atoms

2.28 Refs

2.29 Agents

2.30 Promise and Deliver

A Promise is a mechanism for storing data and reading this data across threads.

The `promise` function creates a promise that can be stored as in:

— promise example —

```
(def prom (promise))
```

— defn promise —

```
(defn promise
  "Alpha - subject to change.
  Returns a promise object that can be read with deref/@, and set,
  once only, with deliver. Calls to deref/@ prior to delivery will
  block. All subsequent derefs will return the same delivered value
  without blocking."
  {:added "1.1"
   :static true}
  []
  (let [d (java.util.concurrent.CountDownLatch. 1)
        v (atom nil)]
    (reify
      clojure.lang.IDeref
      (deref [_] (.await d) @v)
      clojure.lang.IPromiseImpl
      (hasValue [this]
        (= 0 (.getCount d)))
      clojure.langIFn
      (invoke [this x]
        (locking d
          (if (pos? (.getCount d))
            (do (reset! v x)
```

```
(.countDown d)
this)
(throw (IllegalStateException.
"Multiple deliver calls to a promise"))))))))
```

To fulfill a promise we use the **deliver** function. So we would use
 — **promise example** —

```
(deliver prom 1)
```

— **defn deliver** —

```
(defn deliver
  "Alpha - subject to change.
  Delivers the supplied value to the promise, releasing any pending
  derefs. A subsequent call to deliver on a promise will throw an
  exception."
  {:added "1.1"
   :static true}
  [promise val] (promise val))
```

derefrence @ promise To see the delivered value we deference the variable.
 — **promise example** —

```
@prom
```

Note that dereferencing a promise before the value is delivered will block the current thread. So you can have a future deliver on the promise from another thread and wait on the result.

— **promise example** —

```
(do (future
      (Thread/sleep 5000)
      (deliver prom 1))
  @prom)
```

IllegalStateException IllegalStateException promise Attempting to deliver a value more than once will throw a `java.lang.IllegalStateException`

2.31 Futures

Futures are a mechanism for executing a function in a different thread and retrieving the results later.

Futures are useful for long running tasks that should not block, such as a UI thread, communicating across networks, or a task that has a side-effect that does not affect the main computation, such as displaying an image or printing. For example,

— future example —

```
(do
  (future
    (Thread/sleep 3000)
    (sendToPrinter "resume.pdf"))
  (messages/plain-message "file being printed"))
```

A future can return a value. We define a var that will hold the value of a future as in:

— defn future-call —

```
;;;;;;;;
(defn future-call
  "Takes a function of no args and yields a future object that will
  invoke the function in another thread, and will cache the result and
  return it on all subsequent calls to deref/. If the computation has
  not yet finished, calls to deref/0 will block."
  {:added "1.1"
   :static true}
  [f]
  (let [f (binding-conveyor-fn f)
        fut (.submit clojure.lang.Agent/soloExecutor ^Callable f)]
    (reify
      clojure.lang.IDeref
      (deref [_] (.get fut))
      java.util.concurrent.Future
      (get [_] (.get fut))
      (get [_ timeout unit] (.get fut timeout unit))
      (isCancelled [_] (.isCancelled fut))
      (isDone [_] (.isDone fut))
      (cancel [_ interrupt?] (.cancel fut interrupt?)))))
```

— defmacro future —

```
(defmacro future
  "Takes a body of expressions and yields a future object that will
  invoke the body in another thread, and will cache the result and
  return it on all subsequent calls to deref/@. If the computation has
  not yet finished, calls to deref/@ will block."
  {:added "1.1"}
  [& body] '(future-call (#{:once true} fn* [] ~@body)))
```

—

— future var —

```
(def result
  (future
    (Thread/sleep 3000)
    "The future has arrived"))
```

—

derefence @ future We can get the value of this var using the dereference operator "@".

— future dereference —

```
@result
```

—

CancellationException CancellationException future If we dereference the var before the future completes then the dereference blocks until the result is available. The result retains its value so a second dereference will return immediately. If the future is cancelled (see below), then the dereference will throw a —java.util.concurrent.CancellationException—.

future? We can test if a var is a future with the **future?** predicate.

— future dereference —

```
(future? result) ==> true
```

—

— defn future? —

```
(defn future?
  "Returns true if x is a future"
  {:added "1.1"
   :static true}
  [x] (instance? java.util.concurrent.Future x))
```

future-done? We can test if the future has completed with the **future-done?** predicate.

— **future done** —

```
(future-done? result) ==> true
```

— **defn future-done?** —

```
(defn future-done?
  "Returns true if future f is done"
  {:added "1.1"
   :static true}
  [^java.util.concurrent.Future f] (.isDone f))
```

A pending future can be cancelled with the **future-cancel** function

— **future cancel** —

```
(future-cancel result) ==> true
```

which, if it successfully cancels the future returns true.

— **defn future-cancel** —

```
(defn future-cancel
  "Cancels the future, if possible."
  {:added "1.1"
   :static true}
  [^java.util.concurrent.Future f] (.cancel f true))
```

We can test if the future was cancelled with the **future-cancelled?** function

— **future cancelled** —

```
(future-cancelled? result) ==> true
```

which, if the future was cancelled, will return true.

— defn future-cancelled? —

```
(defn future-cancelled?
  "Returns true if future f is cancelled"
  {:added "1.1"
   :static true}
  [^java.util.concurrent.Future f] (.isCancelled f))
```

2.32 MultiMethods

2.33 Deftype

2.34 Defrecord

Define a record object with slots named :a, :b, and :c

```
> (defrecord Foo [a b c])
user.Foo
```

This makes a Java class `Foo` in the Java package `user`.

Make a Var reference to our new record object.

```
> (def f (Foo. 1 2 3))
#user.Foo{:a 1, :b 2, :c 3}
```

Access the value of the :b slot in the record instance f

```
> (:b f)
2
```

or

```
> (.b f)
2
```

What is the Var f actually pointing to? It points to a class object.

```
> (class f)
user.Foo
```

We can apply normal functions to the query information about the class of f.

```
> (supers (class f))
#{clojure.lang.ILookup
  java.lang.Object
  java.util.Map
  clojure.lang.Seqable
  clojure.lang.IKeywordLookup
  java.lang.Iterable
  clojure.lang.IMeta
  clojure.lang.IPersistentCollection
  clojure.lang.IPersistentMap
  clojure.lang.Counted
  clojure.lang.IObj
  java.io.Serializable
  clojure.lang.Associative}
```

We see that the Var f references a record class which has a number of interesting properties. In particular, given this record we know many things about its properties.

- [797] ILookup
- Object Since records are Java Objects they can be compared with the default `equals` method and can be stored in containers or passed as arguments.
- Map Since records implement Map they provide three collection views, as a set of keys, as a set of values, and as a set of key-value mappings.
- [1140] Seqable
- [796] IKeywordLookup
- Iterable Since records are Iterable they implement an `iterator` method that allows them to be the target of a `foreach` statement.
- [799] IMeta
- [800] IPersistentCollection
- [801] IPersistentMap
- [768] Counted
- [800] IObj
- Serializable Since records are Serializable they implement `writeObject` and `readObject` methods that make it possible to stream the state of the object out and recreate it by reading it back.
- [576] Associative

2.35 Protocols

```
> (defprotocol coerce
  "coerce between things"
  (as-file [x] "Coerce argument to a file")
  (as-url [x] "Coerce argument to a URL"))
coerce
```

Defprotocol defines a named set of generic functions which dispatch on the type of the first argument. The functions are defined in the same namespace (and thus the same Java package) as the protocol.

Similar to Java interfaces but are functions, not associated with classes. In this example, we mimic the Java idioms. A record is like class and coerce is like an interface so we are saying that the class `Name` implements the `as-file` method of interface `coerce`.

Extending inline

```
> (defrecord Name [n]
  coerce
  (as-file [] (File. n)))
user.Name
```

```
> (def n (Name. "johndoe"))
 #'user/n
```

```
> (as-file n)
 #!File johndoe;
```

Extending protocol to classes that already exist

Protocols can implement methods without classes by associating them with types of their first argument. So we extend the `coerce` protocol to work with `nil` arguments and `String` arguments.

```
> (extend-protocol coerce
  nil
  (as-file [] nil)
  (as-url [] nil)
  String
  (as-file [s] (File. s))
  (as-url [s] (URL. s)))
nil
```

```
> (as-file "data.txt")
#File data.txt
```

Extending existing classes to add protocols

```
> (defprotocol MyNewProtocol (myNewFunction [x] "answer"))
MyNewProtocol
```

```
> (extend-type String
  coerce
  (as-file [s] (File. s))
  (as-url [s] (URL. s)))
MyNewProtocol
(myNewMethod [x] (str "answer: " x)))
nil
```

```
> (myNewFunction "test")
"answer"
```

```
> (extend-type Long MyNewProtocol (myNewFunction [x] (+ x 1)))
nil
```

```
> (myNewFunction 1)
2
```

2.36 Prototypes

2.37 Genclass

2.37.1 Overriding protected methods

In order to override a protected method and call the superclass you need to make the protected method available under an alternate name. For example, to extend Java class Foo with fooMethod(bar) you would use

```
(ns.com.example.subclass-of-Foo
  (:gen-class :extends com.example.Foo
  :exposes-methods {fooMethod fooSuperMethod}))

(defn -fooMethod [this parameter]
  (fooSuperMethod this parameter))
```

2.38 Proxies

2.39 Macros

2.40 Iterators

(Iterator [1723])

— PersistentTreeMap NodeIterator Class —

```
static public class NodeIterator implements Iterator{
    Stack stack = new Stack();
    boolean asc;

    NodeIterator(Node t, boolean asc){
        this.asc = asc;
        push(t);
    }

    void push(Node t){
        while(t != null)
        {
            stack.push(t);
            t = asc ? t.left() : t.right();
        }
    }

    public boolean hasNext(){
        return !stack.isEmpty();
    }

    public Object next(){
        Node t = (Node) stack.pop();
        push(asc ? t.right() : t.left());
        return t;
    }

    public void remove(){
        throw new UnsupportedOperationException();
    }
}
```

—

(Iterator [1723])

— PersistentTreeMap KeyIterator Class —

```
static class KeyIterator implements Iterator{
```

```

NodeIterator it;

KeyIterator(NodeIterator it){
    this.it = it;
}

public boolean hasNext(){
    return it.hasNext();
}

public Object next(){
    return ((Node) it.next()).key;
}

public void remove(){
    throw new UnsupportedOperationException();
}
}

```

—————
 (Iterator [1723])
 — PersistentTreeMap ValIterator Class —

```

static class ValIterator implements Iterator{
    NodeIterator it;

    ValIterator(NodeIterator it){
        this.it = it;
    }

    public boolean hasNext(){
        return it.hasNext();
    }

    public Object next(){
        return ((Node) it.next()).val();
    }

    public void remove(){
        throw new UnsupportedOperationException();
    }
}

```

—————

2.41 Generating Java Classes

2.42 Type Hints

2.43 Native Arithmetic

Chapter 3

Writing Idiomatic Clojure

Chapter 4

The Ants Demo (Kai Wu)

This is adapted from Wu [1]

4.1 The Simulation World

The first part of **ants.clj** sets up the simulation world, where we'll be introduced to some of Clojure's powers.

4.1.1 Initial setup of constants/magic-numbers

After the copyright notice, the initial setup code of **ants.clj** is easy to understand (for coders at least), even if you've never dealt with Lisp before. We see parameters (aka constants and magic numbers) being defined for later use using Clojure's special form: **def** creates a var (a mutable storage location) which connects a symbol to a value in the current **namespace**.

— sim-world-setup —

```
;;;;;;;;
;; ant sim ;;;;;;;;;;
;; Copyright (c) Rich Hickey. All rights reserved.
;;
;; The use and distribution terms for this software are covered by the
;; Common Public License 1.0 (http://opensource.org/licenses/cpl.php)
;; which can be found in the file CPL.TXT at the root of this distribution.
;; By using this software in any fashion, you are agreeing to be bound by
;; the terms of this license.
;;
;; You must not remove this notice, or any other, from this software.
```

```

;; Set dimensions of the world, as a square 2-D board:
(def dim 80)
;; Number of ants = nants-sqrt^2
(def nants-sqrt 7)
;; Number of places with food:
(def food-places 35)
;; Range of amount of food at a place:
(def food-range 100)
;; Scale factor for pheromone drawing:
(def pher-scale 20.0)
;; Scale factor for food drawing:
(def food-scale 30.0)
;; Evaporation rate:
(def evap-rate 0.99)

(def animation-sleep-ms 100)
(def ant-sleep-ms 40)
(def evap-sleep-ms 1000)

(def running true)

```

4.1.2 The board: ready to mutate via transactions

Things get more interesting once the actual simulation environment needs defining:

— cell —

```
(defstruct cell :food :pher) ; May also have :ant and :home values
```

First, a call to **defstruct** (like a hashmap or dictionary in other languages) defines a baseline *cell*.

- **defstruct** is like a very lightweight class or constructor/template function, and conveniently wraps Clojure's **create-struct**.
- Here, a cell has two keys to start, **:food** and **:pher**, to indicate the presence of food and pheromones. A cell may also have keys of **:ant** and **:home**, depending on whether an ant and/or the home-colony is present.

Next, the **world** function creates the 2-dimensional "board" of cells (here, a square of 80x80 cells), represented as vectors (rows or the vertical y-dimension) of a vector (the horizontal x-dimension columns in one row):

— sim-world-board-creation —

```
;; World is a 2d vector of refs to cells
(def world
  (apply vector
    (map (fn [_]
      (apply vector
        (map (fn [_]
          (ref (struct cell 0 0)))
        (range dim)))))
    (range dim))))
```

Reading the above:

- Start with the innermost **map** call, which uses an anonymous function to create one column of 80 cells, per (**range dim**). The **struct** returns a new structmap instance using the earlier cell as the basis, initializing the **:food** and **:pher** values to zero.
- But notice that **struct** is wrapped with a **transactional ref**, and here's the first glimpse of Clojure's concurrency powers. With each cell being stateful (possibly time-varying values of **:food**, **:pher**, **:ant**, and **:home** values) and with multiple threads updating the board and board elements, we'd typically think of using locks on each cell when updating its state.
But in Clojure with its **software transactional memory** (STM), we just use **ref** for safe references to mutable collections (here, a **struct**) - all changes to a cell will then be atomic, consistent, and isolated!¹ Like using an RDBMS, you don't need to manually manage concurrency.
- Once you understand the innermost (**ref (struct cell 0 0)**) **map** call, the rest of (**def world...**) is straightforward: **apply** uses **vector** as a constructor function with the **map** function producing the vector's arguments, creating a "column" in the 2-D board.
- Then the pattern is repeated in the outermost (**apply vector (map...)**) call, creating all the columns of the 2-D board.
- Note that as defined, each vector in **world** (again, a 2-D vector of vectors) corresponds to an x-position, and of course, within that vector are the y-positions (here, a total of 80 cells).

The **place** function is a selector function (think of "place" as the noun, not the verb) returning particular cells in the 2-D world. Once we have a cell, we can then mutate it to represent ants, food, and pheromones (or their absence):

— place —

¹STM is like a memory-only SQL database, thus the last property of being durable/persistent won't be satisfied.

```
(defn place [x y]
  (-> world (nth x) (nth y)))
```

- **place** takes a single vector argument (having two elements **x** and **y**), then applies the *thrust operator* (the arrow-like \rightarrow) on the **world** object, first selecting the “column” (**nth x**) on **world**, then the “row” (**nth y**) on that column.

Aside: the thrust operator

The thrust operator helps make code more concise, and arguably clearer: instead of reading code “inside-out” to mentally evaluate it, we can read it left-to-right.² Consider how the equivalent **place** function would look without thrushing:

```
(defn place-verbose [[x y]]
  (nth (nth world x) y))
```

4.1.3 Ants as agents - doing asynchronous uncoordinated changes

Next we’ll consider the “active things” in **ants.clj**, the ants themselves. As before, we start with **defstruct**, defining an ant as having only one required key, its direction. (An ant may temporarily have another key, **:food**.)

— ants-defined —

```
(defstruct ant :dir) ; Always has dir heading; may also have :food

(defn create-ant
  "Create an ant at given location, returning an ant agent on the location."
  [location direction]
  (sync nil
    (let [the-place (place location)
          the-ant (struct ant direction)]
      (alter the-place assoc :ant the-ant)
      (agent location))))
```

²Apparently Clojure’s thrust is not quite a true thrust, see Michael Fogus’ article at blog.fogus.me/2010/09/28/thrush-in-clojure-redux

To explain the above constructor function for ants, **create-ant**:

- Takes two arguments, **location** and **direction**. **location** will be a vector `[x y]`, and as we saw, passed on to the `place` function as an argument; **direction** is a number from 0-7 inclusive corresponding to one of the eight cardinal directions.
- More concurrency support: the **sync** function takes a flags argument (as of Clojure 1.3, it's still ignored so just pass nil), and then a list of expressions that will be executed together atomically (all or nothing) as a transaction.
- The **let special form** binds pairs of symbols and expressions in its arguments vector, providing local, lexical bindings within the scope of the body following.
- **sync** will ensure that any mutations of refs using the **alter** function will be atomic. Previously we had used **ref** around each cell, so in the above code where **the-place** is such a ref-wrapped cell, **alter** takes **the-place** ref as its first argument, then **assoc** as the function to be **apply**'ed on the-place, tying a new ant instance to it (remember that as a cell, **the-place** is sure to have `:food` and `:pher` key-values already, now we add `:ant`). Like the `thrust` operator earlier, the syntax of **alter** enables convenient left-to-right reading.
- Finally, the **agent** function. What are Clojure agents? To quote the docs,

Agents provide shared access to mutable state. They allow non-blocking (asynchronous as opposed to synchronous atoms) and independent change of individual locations (unlike coordinated change of multiple locations through refs).

Clojure's **agent** function takes one required argument of state, returning an agent object with initial value of that given state. Here, as the last line of **create-ant**, **agent** effectively returns the ant object at its starting location. Ants as agents make sense: we expect them to move around independently (i.e. asynchronously) in the simulation world.

4.1.4 Setting up the home, and ants

The home of the ants is not a single cell on the world-board, but a square of cells, with its top-left corner offset from the origin (0, 0). Its sides are proportional to the number of ants because the home square will initially contain all the ants - one ant per cell - before the simulation runs. We can see these two aspects of the home-square in the two **def** calls for **home-offset** and **home-range** below.

— home-setup —

```
(def home-offset (/ dim 4))
(def home-range (range home-offset (+ nants-sqrt home-offset)))

(defn setup
  "Places initial food and ants, returns seq of ant agents."
  []
  (sync nil
    (dotimes [i food-places]
      (let [p (place [(rand-int dim) (rand-int dim)])]
        (alter p assoc :food (rand-int food-range))))
    (doall
      (for [x home-range y home-range]
        (do
          (alter (place [x y]) assoc :home true)
          (create-ant [x y] (rand-int 8)))))))
```



The **setup** function's docstring tells us what it's doing, so on to the details:

- **setup** takes no arguments.
- As we saw before in **create-ant**, the **sync** function wraps a sequence of expressions that together should be executed atomically, all-or-nothing.
- Setup initial food: The **dotimes** function takes two arguments, the first a vector **[name n]** with **n** being the number of times that the **body** (the second argument) will be repeatedly executed, usually for its side-effects/mutations.
 - Here, the unused name **i** is bound to the integers from 0 to 34, since we had specified food-places as 35 initially.
 - The **body** is clear enough: bind **p** to the randomly chosen place on the world-board (using the **rand-int** function for (x, y). The already-seen **alter** function modifies that **p** to have a random amount of food value.
- Placing the ants in their starting positions: The **doall** function forces immediate evaluation of a lazy sequence - in this case the lazy sequence produced by the **for** function.
 - Here, the **for** function's first argument is: two binding-form/collection-expr pairs for every x and y position within the square of the ants' home.

- The **for** function's second argument is the body-expression, here wrapped in the **do** special form which ensures order of evaluation (usually, of expressions having side-effects): designate the place as a home position, then create an ant on that place with a random initial direction.

In sum, the **setup** function shows how to deal with state and its mutation in Clojure: we started with a 2-D world-board of places (cells) as Clojure refs; then we modify/mutate each place using **alter**. We can use various looping functions such as **dotoimes** and **doall** to process a batch of state-mutations (of the world-board) atomically and consistently.

4.1.5 Orientation and moving around the world

Next, consider facing/orientation and moving to another place in the 2-D world. Three functions below, followed by explanations:

— world-wrapping —

```
(defn bound
  "Returns given n, wrapped into range 0-b"
  [b n]
  (let [n (rem n b)]
    (if (neg? n)
        (+ n b)
        n)))

;; Directions are 0-7, starting at north and going clockwise. These are
;; the 2-D deltas in order to move one step in a given direction.
(def direction-delta {0 [0 -1]
                      1 [1 -1]
                      2 [1 0]
                      3 [1 1]
                      4 [0 1]
                      5 [-1 1]
                      6 [-1 0]
                      7 [-1 -1]})

(defn delta-location
  "Returns the location one step in the given direction. Note the
  world is a torus."
  [[x y] direction]
  (let [[dx dy] (direction-delta (bound 8 direction))]
    [(bound dim (+ x dx)) (bound dim (+ y dy))]))
```

With the 2-D world board, we have the 8 cardinal directions (North, North-East, East, etc.), and board edges that wrap-around to the opposite side - like the old arcade games of the 1980's, e.g. *Pac-Man* and *Asteroids*. The functions **bound** and **delta-location** help enforce these world-behaviors, while the definition of **direction-delta** maps a movement in a cardinal direction to the corresponding change in x and y. A few comments on each:

- The **bound** function using the built-in **rem** (i.e. remainder) function is straightforward. Observe how **bound** is used in delta-location to ensure wrap-around behavior in: 1) cardinal directions; 2) the world-board, at its edges given by **dim**.
- **direction-delta** maps the eight cardinal directions (0 is North) to the corresponding changes in [x y]. Note the syntax: it's an array-map literal, where the order of insertion of key-value pairs (here, keys 0-7) will be preserved.
- **delta-location** takes the current [x y] location and a direction, returning the new corresponding location on the world-board.

4.1.6 Ant-agent behavior functions

In Hickey's simulation, ants need to move (rotation and translation), pick up and drop-off food, and make rudimentary decisions.

Ant movements

Our ants need two behaviors to get around their world: turning (or changing the direction they "face"), and stepping forward. Let's deal with turning first:

— ant-agent-turn —

```
;ant agent functions
; An ant agent tracks the location of an ant, and controls the
; behavior of the ant at that location.

(defn turn
  "Turns the ant at the location by the given amount."
  [loc amt]
  (dosync
    (let [p (place loc)
          ant (:ant @p)]
      (alter p assoc :ant (assoc ant :dir (bound 8 (+ (:dir ant) amt)))))))
  loc)
```

The **turn** function takes two arguments, location and the amount of turn. What's interesting is the usage of the **dosync** function, which ensures the ant's turn - the changes of state within the **assoc** function calls - is all-or-nothing. The ant gets a new direction per the innermost **assoc**, then the outermost **assoc** updates the **place** with the updated ant.

Now for actual movement to a new place:

— ant-agent-move —

```
(defn move
  "Moves the ant in the direction it is heading. Must be called in a
  transaction that has verified the way is clear."
  [startloc]
  (let [oldp (place startloc)
        ant (:ant @oldp)
        newloc (delta-location startloc (:dir ant))
        newp (place newloc)]
    ;; move the ant
    (alter newp assoc :ant ant)
    (alter oldp dissoc :ant)
    ;; leave pheromone trail
    (when-not (:home @oldp)
      (alter oldp assoc :pher (inc (:pher @oldp))))
    newloc))
```

The **move** function changes state of both the ant and board, thus the doc-string note that it must be called in a transaction. The code is self-explanatory, though if “pheromone” is a new term to you, you’ll want to learn about a dominant form of chemical communication on Earth. Whenever our artificial ant is not within its home, it will “secrete” pheromone (**inc** the **:pher** value by 1) at the place it just left, making it easier (more likely) for it and other ants to travel between home and food locations in the future (instead of doing a completely random walk).

Ants and food

When an ant finds food, it “picks up” one unit of it; when it returns home with a food unit, it will “drop” its food there. These two interactions (each having two steps) change the board, and as with the **move** function, they need to occur atomically (all-or-nothing) to ensure the world is in a consistent state.

— ant-agent-food —

```
(defn take-food [loc]
  "Takes one food from current location. Must be called in a
  transaction that has verified there is food available."
  (let [p (place loc)
        ant (:ant @p)]
    (alter p assoc
           :food (dec (:food @p))
           :ant (assoc ant :food true)))
  loc))

(defn drop-food [loc]
  "Drops food at current location. Must be called in a
  transaction that has verified the ant has food."
  (let [p (place loc)
        ant (:ant @p)]
    (alter p assoc
           :food (inc (:food @p))
           :ant (dissoc ant :food)))
  loc))
```

Notice how similar the structure is for the two functions above; possibly they're candidates for macro refactoring.

Ant judgment

Our ants need some decision-making for their overall task of finding food and bringing it home. As we'll see shortly, an ant's behavior is based on two states, either:

1. The ant does not have food, and is looking for it. In this mode, it weighs the three map locations ahead of it (ahead, ahead-left, ahead-right) by the presence of either food or pheromone.
2. The ant has food, and needs to bring it to the home box/location. Now it weighs which of the three ahead-positions to take by the presence of pheromone, or home.

So we need functions to express preference of the next location for an ant. The functions **rank-by** and **wrand** help with that.

— ant-agent-judgment-1 —

```
(defn rank-by
  "Returns a map of xs to their 1-based rank when sorted by keyfn."
  [keyfn xs])
```

```
(let [sorted (sort-by (comp float keyfn) xs)]
  (reduce (fn [ret i] (assoc ret (nth sorted i) (inc i)))
          {} (range (count sorted))))
```

The **rank-by** function gives weights to where an ant will move next in the simulation world. It takes two arguments, **keyfn** and **xs** - but what do those args look like, and where is **rank-by** used? In the **behave** function below; you'll see that the **keyfn** checks for the presence of **:food**, **:pher**, or **:home** - in the three cells (board locations) of the **xs** vector of [**ahead ahead-left ahead-right**].³

- The (**sort-by keyfn coll**) function returns a sorted sequence of items in coll, ordered by comparing (**keyfn item**). Here, for the local value sorted, it will be ascending order of cells/places, by their **:food**/**:home**/**:pher** values - each of those is valuable to an ant depending on whether it's looking for food, or bringing it home.
- The (**reduce f initial-val coll**) functionn in its 3-arguments form here has its 1st argument **f** as a function taking two arguments, the current/initial-val value and the next/first item from coll. In this case, it will “build-up” a map from the local sorted value, with the keys being the ranked cells/places, and the values being integers 1, 2 and 3. To get a sense of what's going on, try this on your Clojure REPL:

```
(let [sorted [0 0.7 1.0]]
  (reduce (fn [ret i] (assoc ret (nth sorted i) (inc i)))
          {}
          (range (count sorted))))
  ; You should see {1.0 3, 0.7 2, 0 1}
  ;;
  ; Within the behave function below, the return value might be
  ; like {<cell-ahead-left> 3, <cell-ahead-right> 2, <cell-ahead> 1}
  ; or similar.
```

Next: The **wrand** function helps with the larger task of randomizing which location/cell the ant moves to next in a weighted manner; i.e. the “dice” are loaded with **rank-by**, then “rolled” here:

— ant-agent-judgment-2 —

³Remember that **:food**, **:pher**, and **:home** are mutually exclusive in a cell. When an ant wants to go home with food, and the home cell(s) is ahead of it, it will always go home, there won't be competing **:pher** presence.

```
(defn wrand
  "Given a vector of slice sizes, returns the index of a slice given a
  random spin of a roulette wheel with compartments proportional to
  slices."
  [slices]
  (let [total (reduce + slices)
        r (rand total)]
    (loop [i 0 sum 0]
      (if (< r (+ (slices i) sum))
          i
          (recur (inc i) (+ (slices i) sum))))))
```

How is **wrand** used? Like **rank-by**, look in the **behave** function: its single argument of slices is a vector of 3 integers (from **rank-by** above), corresponding to the relative desirability of the 3 cells ahead of the ant. So if the slices argument looked like **[0 3 1]**, that would correspond to zero probability of moving ahead, and 3/4 chance moving to the ahead-left cell over the ahead-right cell.

- The **let** value **total** uses **reduce** to set the upper bound on the random number; loosely like setting the maximum number of faces on the die to be rolled (albeit that some die numbers are geometrically impossible).
- The **rand** function returns a random floating point number from 0 (inclusive) to n (exclusive).
- Here's the only looping construct in the entire ants program: it's analogous to checking which compartment of the roulette wheel the ball fell in. The **if** checks if **r** "fell into" the current pocket - the size of which is given by **(slices i)**. If yes, return the index corresponding to that pocket; if not, check the next pocket/slice.

Tying it all together: the **behave** function for ants

The **behave** function below is the largest one, so it helps to keep in mind its main parts while diving into details:

1. **let** values - help with readability.
2. **Thread/sleep** - helps slow down ants in the UI display.
3. **dosync** - ensures ants behavior is transactional, all-or-nothing.
4. **if** branch: main logic for an ant, if ant has **:food** take it home, otherwise look for food.

Also, consider the context of how **behave** is first used: within the main invocation at the end, there's the expression:

```
(dorun (map #(send-off % behave) ants))
```

So the **behave** function is called on every ant agent via the **send-off** function, which is how Clojure dispatches potentially blocking actions to agents. And there certainly are potentially blocking actions when using **behave**, since ants may try to move into the same cell, try to acquire the same food, etc.

— ant-agent-behave —

```
(defn behave
  "The main function for the ant agent."
  [loc]
  (let [p (place loc)
        ant (:ant @p)
        ahead (place (delta-location loc (:dir ant)))
        ahead-left (place (delta-location loc (dec (:dir ant))))
        ahead-right (place (delta-location loc (inc (:dir ant))))
        places [ahead ahead-left ahead-right]]
    ;; Old way of Java interop: (. Thread (sleep ant-sleep-ms))
    ;; New idiomatic way is,
    (Thread/sleep ant-sleep-ms)
    (dosync
      (when running
        (send-off *agent* #'behave)))
    (if (:food ant)
        ;; Then take food home:
        (cond
          (:home @p)
          (-> loc drop-food (turn 4))
          (and (:home @ahead) (not (:ant @ahead)))
          (move loc)
          :else
          (let [ranks (merge-with +
                           (rank-by (comp #(if (:home %) 1 0) deref) places)
                           (rank-by (comp :pher deref) places))]
            (([move #(turn % -1) #(turn % 1)]
              (wrand [(if (:ant @ahead) 0 (ranks ahead))
                      (ranks ahead-left) (ranks ahead-right)]))
             loc)))
        ;; No food, go foraging:
        (cond
          (and (pos? (:food @p)) (not (:home @p)))
          (-> loc take-food (turn 4)))
          (and (pos? (:food @ahead)) (not (:home @ahead)) (not (:ant @ahead)))
          (move loc)))
```

```
:else
  (let [ranks (merge-with +
                    (rank-by (comp :food deref) places)
                    (rank-by (comp :pher deref) places))]
    (([move #(turn % -1) #(turn % 1)]
      (wrand [(if (:ant @ahead) 0 (ranks ahead))
              (ranks ahead-left) (ranks ahead-right)])
      loc))))))
```

The let values

The **let** values: quite straightforward, just note the twist in how **behave** receives a cell/location as its argument, not an ant (which an OO-centric design might expect).

The only JVM/concurrency leakage: Thread/sleep

The

```
(. Thread (sleep ant-sleep-ms))
```

or

```
(Thread/sleep ant-sleep-ms)
```

call is our first encounter with Clojure's Java Interop.

- The first version uses the **dot** special form and in particular, the

```
(. Classname-symbol (method-symbol args*))
```

format, with **Thread** as the Classname-symbol, and **sleep** as the method-symbol.

- However, outside of macros, the idiomatic form for accessing method members is the second form,

```
(Classname/staticMethod args*)
```

- Beyond syntax, the point of this expression is to slow down an ant (one ant-agent per thread) between their movements, so you can see in the UI what they're doing, and they'll appear more realistic.

But more interesting still: in this highly concurrent program, the **sleep** expression is about the *only explicit reference to threads* in the entire code, i.e. one of the very few “leaky abstractions” hinting at Clojure's use of underlying JVM

concurrency constructs. Besides this call, there are no locks, and no explicit thread allocations.

The main dosync call

Next, let's look at what's going on within the **dosync** transaction.

Repeating asynchronously, without looping

The first expression is:

```
(when running (send-off *agent* #'behave))
```

Initially this may seem strange; aren't we in the **behave** function because **send-off** already called it before entering it? Won't this just loop uselessly, not hitting the core **if** code below? Not quite:

- Instead, **send-off** adds another execution of **behave** to the current agent's *queue* of work/functions, and immediately returns.
 - The current agent is referenced by the asterisk-surrounded ***agent*** which Clojure dynamically binds to the current active agent on a thread-local basis.
- Thus after finishing this call of **behave** the ant will do another action (execute **behave** again), and another, and so on. No explicit looping, just *queue and repeat*.

Also, note the #' sharp-quote, before **behave**; this is a Clojure Var, one of Clojure's mutable reference types. It's just syntactic sugar for (**var behave**). Invoking a Var referring to a function is the same as invoking the function itself...so why bother with it? I don't know; here's what I could find:

- Besides Clojure docs, this SO thread also suggests there's no difference, "Apply a **var** is the same as applying the value store in the **var**."
- Maybe the #' prefix on **behave** causes the current thread's value of the function (with the current ant/location) to be sent to the queue? NO/unlikely. If it was meant to be a dynamic var, it would have asterisks around it like ***agent***.
- Another possibility is that the #' maintains a runtime look-up so that **behave** can be redefined on a running simulation while developing.

Why use **send-off** instead of **send**?

- send vs. send-off - **send** uses threadpool of fixed size which has low switching overhead but blocking can dry up the threadpool. By contrast, **send-off** uses a dynamic threadpool and blocking is tolerated - and that's the right approach here as ant contention for the same location/food can certainly cause (temporary) blocking.

Determining what the ant does next

Finally, the ant's logic for what to do next is in the large **if** expression. The code looks dense but at the top level it's just a binary choice:

- If the ant has food, take it home; the **cond** specifies 3 sub-cases:
 1. At a home cell, drop the food and turn around 180 degrees, to exit home for more food.
 2. If a home cell is ahead, move to it.
 3. Otherwise, do a ranking of cells ahead (**places** has the cells **ahead**, **ahead-left**, **ahead-right**) per presence of pheromones, or home, and then randomly select from those 3 cells per their ranking/weighting.

4.1.7 World behavior: pheromone evaporation

— evaporate —

```
(defn evaporate
  "Causes all the pheromones to evaporate a bit."
  []
  (dorun
    (for [x (range dim) y (range dim)]
      (dosync
        (let [p (place [x y])]
          (alter p assoc :pher (* evap-rate (:pher @p)))))))
```

For a bit of realism and a cleaner UI/visual, it's useful to have the ants' pheromones diminish and evaporate from the world over time. The **evaporate** function fulfills that requirement:

- It takes no arguments, it will work over the entire world/board of cells, accessed via the tuples of **x** and **y**.
- The **dorun** function takes a lazy collection/sequence (here, that of the **for** expression) and forces the realization of that collection for its side effects, discarding any returned values.
 - It's unlike the similarly-named **doall** where we do care about the values.
 - And it's unlike **doseq**, which is like Clojure's **for** but runs immediately and does not collect the results.

- **dosync** is used as before, for lock-free updating of a **place** cell. Here, the desired side-effect/“mutation” is to update the **:pher** value at the **place** cell with a lower number.

We'll see shortly that **evaporate** will run every second, a process that (like the ants) will be handled asynchronously using a Clojure agent.

4.2 The UI

The user interface for the ants relies heavily on Clojure's Java inter-operation capabilities. But as we'll see, it's more than just wrapping calls to Java.

4.2.1 Using the Java AWT

— clojureUI —

```
;;;;;; UI ;;;;;;;;
(import '(java.awt Color Graphics Dimension)
        '(java.awt.image BufferedImage)
        '(javax.swing JPanel JFrame))
```

The **import** pulls in classes from Java's *Abstract Window Toolkit* (AWT) package, and from the Java Swing package. Assuming unfamiliarity with Java Swing, let's describe the classes used:

- The **Color** class encapsulates a color in the standard RGB color space. In the code below, its usage as a constructor for a color instance follows several arities:
 - 4 integer arguments: r, g, b, and a for the alpha/transparency (0 transparent, 255 opaque)
 - 3 integer arguments: r g b
 - 1 argument: not a constructor call, but an access of a predefined static **Color** field by name, returning the color in the RGB color space.
- The **Graphics** class is an abstract base class for all graphics contexts, i.e. a **Graphics** instance holds the current state data needed for rendering it: the **Component** object on which to draw, the current clip, color, and

font, etc. Below, we'll see that the Clojure functions that take a **Graphics** instance as an argument:

- **fill-cell**
- **render-ant**
- **render-place**
- **render**

...all do some kind of rendering/drawing.

- The **Dimension** class encapsulates the integer width and height of a component. This class is used just once below, in setting the size of the panel of the UI.
- **BufferedImage** class is needed for raster image data; below, the **render** function uses it to paint the background panel.
- The **JPanel** class is the generic “lightweight” UI container in Java Swing (seems like the **div** element in HTML). Below, it's used just once for the main display.
- The **JFrame** class creates a top-level window (w/ title and border) in Swing; it's used just once below for the main ants UI window.

4.2.2 Functions to render the board and the ants

Each discrete cell on the world board is a square matrix of pixels; with an odd number of pixels chosen, we can have a central position:

— **UI-scale** —

```
;pixels per world cell
(def scale 5)
```

By default, cells are empty; drawing cells having food or ant-deposited pheromones is done by filling with symbolic colors - here by running the Java methods **setColor** and **fillRect**:

— **UI-fill-cell** —

```
(defn fill-cell [#^Graphics g x y c]
  (doto g
    (.setColor c)
    (.fillRect (* x scale) (* y scale) scale scale)))
```

Note the use of the **doto** function here and in many places below: in Java, procedural mutation of a newly constructed instance is common for initialization. Clojure's **doto** function is meant to be more concise in specifying the target object just once, and then methods/setters acting on it and then returning it, implicitly.

Drawing an ant: the graphical appearance of an ant is just a (5-pixel long) line pointing in one of the 8 cardinal directions, of two different colors (having food or not):

— UI-render-ant —

```
(defn render-ant [ant #^Graphics g x y]
  (let [black (. (new Color 0 0 0 255) (getRGB))
        gray (. (new Color 100 100 100 255) (getRGB))
        red (. (new Color 255 0 0 255) (getRGB))
        [hx hy tx ty] ({{0 [2 0 2 4] ; Up/North pointing
                      1 [4 0 0 4]
                      2 [4 2 0 2]
                      3 [4 4 0 0]
                      4 [2 4 2 0] ; Down/South
                      5 [0 4 4 0]
                      6 [0 2 4 2]
                      7 [0 0 4 4]}}
                     (:dir ant))]
    (doto g
      (.setColor (if (:food ant)
                    (new Color 255 0 0 255)
                    (new Color 0 0 0 255)))
      (.drawLine (+ hx (* x scale)) (+ hy (* y scale))
                 (+ tx (* x scale)) (+ ty (* y scale))))))
```

Note the cleverly concise destructuring for the start and end drawing coordinates, needed in AWT's **drawLine** method.

If a cell in the ants' world is not empty, it has one or more of three things present: pheromone, food, or an ant. The **render-place** function updates the cell's appearance accordingly:

— UI-render-place —

```
(defn render-place [g p x y]
  (when (pos? (:pher p))
    (fill-cell g x y (new Color 0 255 0
                                (int (min 255 (* 255 (/ (:pher p) pher-scale)))))))
  (when (pos? (:food p))
    (fill-cell g x y (new Color 255 0 0
                                (int (min 255 (* 255 (/ (:food p) food-scale)))))))
```

```
(when (:ant p)
  (render-ant (:ant p) g x y)))
```

Finally, the **render** function ties everything together: initializing the UI/window appearance by applying **render place** to every cell, and also drawing the home space of the ants. Note the heavy usage of the dot special form: the UI code relies heavily on Java, though Clojure's **for** and **doto** help us avoid Java boilerplate and stay concise:

— UI-render —

```
(defn render [g]
  (let [v (dosync (apply vector (for [x (range dim) y (range dim)]
                                     @(place [x y]))))
        img (new BufferedImage (* scale dim) (* scale dim)
                               (. BufferedImage TYPE_INT_ARGB))
        bg (. img (getGraphics))]
    ;; First paint everything white, on the bg instance:
    (doto bg
      (.setColor (. Color white))
      (.fillRect 0 0 (. img (getWidth)) (. img (getHeight))))
    (dotrun
      (for [x (range dim) y (range dim)]
        (render-place bg (v (+ (* x dim) y)) x y)))
    ;; Draw the home space of the ants:
    (doto bg
      (.setColor (. Color blue))
      (.drawRect (* scale home-offset) (* scale home-offset)
                 (* scale nants-sqrt) (* scale nants-sqrt)))
    (. g (drawImage img 0 0 nil))
    (. bg (dispose)))) ; Finished using Graphics object, release it.
```

4.2.3 Setting the scene, then updating it continually

Almost ready to begin our simulation; we need to setup some additional elements per AWT conventions: the main UI **panel** where visual changes take place, the top-level window **frame**, and an **animator** agent that continually updates the visual elements:

— UI-panel —

```
(def panel (doto
            (proxy [ JPanel ] [] (paint [g] (render g)))
            (.setPreferredSize (new Dimension
```

```
(def frame (doto (new JFrame) (.add panel) .pack .show))

-----
```

Animation, panel-by-panel

Now for bringing the static starting “picture” to life - like the cartoons of old, the **animation** function will “draw” the next state of the main panel displaying the ants. Below, Hickey uses the queue-itself-then-run, again-and-again code pattern we’ve seen before (above, in updating an ant’s state):

— UI-animation —

```
(def animator (agent nil))

(defn animation [x]
  (when running
    (send-off *agent* #'animation))
  (. panel (repaint))
  (. Thread (sleep animation-sleep-ms))
  nil)

-----
```

Finally, we need another agent to handle one more time-track of changes: evaporation, using the **evaporate** function defined above.

— UI-evaporation —

```
(def evaporator (agent nil))

(defn evaporation [x]
  (when running
    (send-off *agent* #'evaporation))
  (evaporate)
  (. Thread (sleep evap-sleep-ms))
  nil)

-----
```

4.3 Running the Program

4.3.1 Running the simulator

At the REPL, you can enter the entire `do` expression below, or try each line within it separately:

— runtheprogram —

```
(do
  (load-file "./literate-ants.clj")
  (def ants (setup))
  (send-off animator animation)
  (dorun (map #(send-off % behave) ants))
  (send-off evaporator evaporation))
```

— — —

Either way you'll see a new window appear with a white background, blue square representing the ants' home, red squares of food, black or red (w/ food) moving lines representing each ant, and green squares for pheromones in various concentrations. A lot happening concurrently, with no locks, and beautifully concise code - welcome to Clojure!

— The Ants Demo —

```
\getchunk{sim-world-setup}
\getchunk{cell}
\getchunk{sim-world-board-creation}
\getchunk{place}
\getchunk{ants-defined}
\getchunk{home-setup}
\getchunk{world-wrapping}
\getchunk{ant-agent-judgment-2}
\getchunk{ant-agent-turn}
\getchunk{ant-agent-move}
\getchunk{ant-agent-food}
\getchunk{ant-agent-judgment-1}
\getchunk{ant-agent-behave}
\getchunk{evaporate}
\getchunk{clojureUI}
\getchunk{UI-scale}
\getchunk{UI-fill-cell}
\getchunk{UI-render-ant}
\getchunk{UI-render-place}
\getchunk{UI-render}
\getchunk{UI-panel}
\getchunk{UI-animation}
\getchunk{UI-evaporation}
```

\getchunk{runtheprogram}

—

Chapter 5

Parallel Processing

5.1 Natural Graphs

Could we do massively parallel work in “native” lisp (i.e. not just a swig cover for MPI). Could we develop a lisp graph data structure that was designed to be run in parallel with multiple processes (aka closures, continuations) “owning” subgraphs and using the subgraph edges leaving their “clades” as natural communication paths? That way we could develop a “natural map” from the graph to the available resources.

It raises interesting questions, such as how to dynamically allocate the graph map to nodes transparently, similar to the way garbage collection is now transparent. This would allow Lisp programs to be written against the graph and run in parallel without knowing about parallel issues. The immutable data structures and STM would help with the coordination and state issues.

The combination of a processor and a subgraph is called a “clade”, similar to the idea of calling a processor and a program a “process”.

5.2 Steele’s parallel ideas

From Steele [Ste09] we find the following ideas:

- effective parallelism uses trees
- associative combining operators are good
- mapreduce is good
- catamorphisms are good
- there are systematic strategies for parallelizing sequential code
- we must lose the accumulator paradigm
- we must emphasize divide and conquer

Contrast good sequential code and good parallel code

- good sequential code minimizes total number of operations
 - clever tricks to reuse previously computed results
 - good parallel code often performs redundant operations to reduce communication
- good sequential algorithms minimize space usage
 - clever tricks to reuse storage
 - good parallel code often requires extra space to permit temporal decoupling
- sequential idioms stress linear problem decomposition
 - process one thing at a time and accumulate results
 - good parallel code usually requires multiway problem
- linearly linked lists are inherently sequential
 - compare Peano arithmetic: $5 = (((0+1)+1)+1)+1$
 - binary arithmetic is much more efficient than unary
- we need a multiway decomposition paradigm

```
length [] = 0
length [a] = 1
length (a++b) = (length a) + (length b)
```

This is just a summation problem: adding up a bunch of 1's!

Total work: $\Theta(n)$

Delay: $\Omega(\log n)$, $O(n)$ depending on how $a++b$ is split even worse if splitting has worse than constant cost

Thoughts

What you want to do versus how you want to do it.

Give the compiler wiggle room. The wiggle room concept is based on the idea that certain mathematical properties allow the compiler to rewrite a computation into a parallel form. So the idea of associativity, which allows terms of an expression to be grouped in arbitrary ways without affecting the computation allows us to rewrite

$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8$

as

$((1 + 2) + (3 + 4)) + ((5 + 6) + (7 + 8))$

which then allows the compiler to add the subterms and then add their results. Steele identifies several mathematical properties that the compiler can exploit:

- associativity
- commutativity

What are the parallel operators. Which of these parallel operators are thinking vs implementation.

Could the add tree be log 32 since add can be multi-arg? Is there a mapping from persistent trees to parallel code?

Clojure could partition tasks based on log32. A better, more fine grained alternative is to use [55] bit-partitioning to split tasks based on the size of the task and the number of processors. That is,

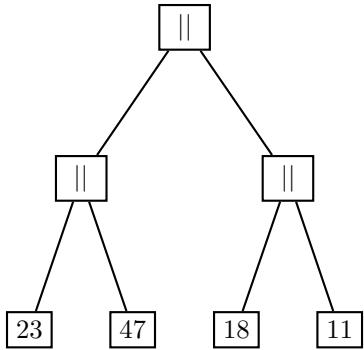
```
(partition chunksize processors)
```

Concatenation lists give wiggle room. That is,

- `<>` is the empty list
- `<23>` is a singleton list
- `< a || b >` is a concatenation of a and b

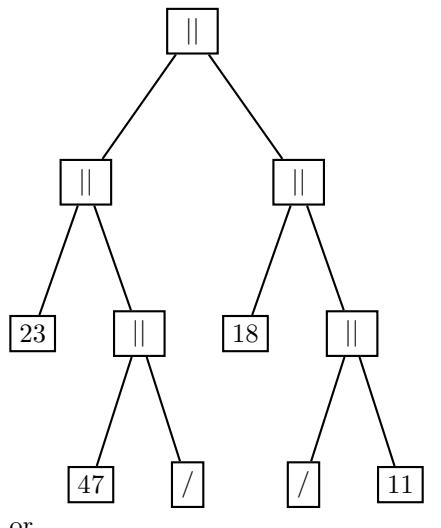
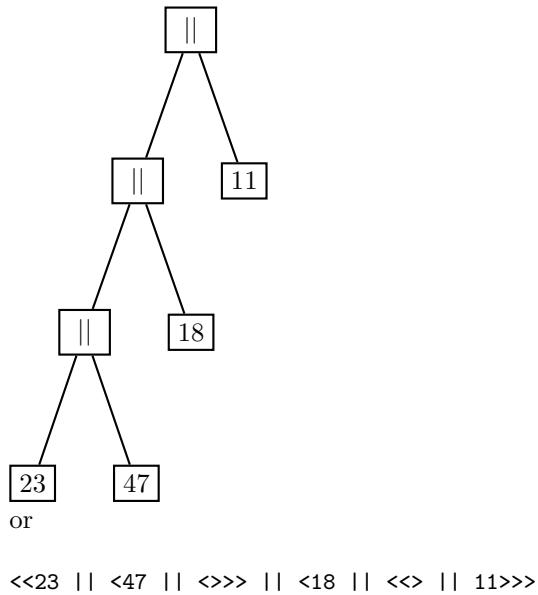
so

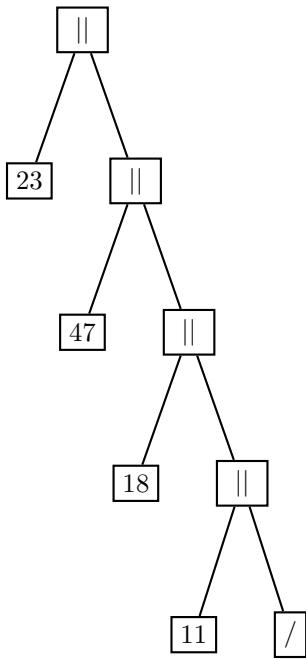
```
<<23 || 47> || <18 || 11>>
```



but the same list could be:

```
<<<23 || 47> || 18> || 11>>
```





5.3 Sequential mapreduce

For the list form Steele defines three operators, map, reduce, and mapreduce. They are:

— Steele map —

```
(define (map f xs) ; linear in (length xs)
  (cond ((null? xs) '())
        (else (cons (f (car xs)) (map f (cdr xs))))))
```

—————

so, for example,

$(\text{map} (\lambda (x) (* x x)) '(1 2 3)) \Rightarrow (1 4 9)$

— Steele reduce —

```
(define (reduce g id xs) ; linear in (length xs)
  (cond ((null? xs) id)
        (else (g (car xs) (reduce g id (cdr xs))))))
```

—————

so, for example,

$(\text{reduce} + 0 '(1 4 9)) \Rightarrow 14$

— Steele mapreduce —

```
(define (mapreduce f g id xs) ; linear in (length xs)
  (cond ((null? xs) id)
        (else (g (f (car xs)) (mapreduce f g id (cdr xs))))))
```

—————

so, for example,

$(\text{mapreduce} (\lambda (x) (* x x)) + 0 '(1 2 3)) \Rightarrow 14$

— Steele length —

```
(define (length xs) ; linear in (length xs)
  (mapreduce (lambda (q) 1) + 0 xs))
```

—————

Using structural recursion, we can define filter as:

— Steele filter —

```
(define (filter p xs) ; linear in (length xs)
  (cond ((null? xs) '())
        ((p (car xs)) (cons p (filter p (cdr xs))))
        (else (filter p (cdr xs)))))
```

—————

Alternatively we could map the predicate down the list:

— Steele filter 2 —

```
(define (filter p xs) ; linear in (length xs) ??
  (apply append
    (map (lambda (x) (if (p x) (list x) '())) xs)))
```

—————

Or, by using mapreduce

— Steele filter 3 —

```
(define (filter p xs) ; linear in (length xs) !!
  (mapreduce (lambda (x) (if (p x) (list x) '()))
    append '() xs)) ; each call to append is constant time
```

—————

The reverse function could be defined structurally as:

— Steele reverse 1 —

```
(define (reverse xs) ; quadratic in (length xs)
  (cond ((null? xs) '())
        (else (addright (reverse (cdr xs)) (car xs)))))
```

—

Or we could define a new function, revappend, by using the accumulator trick to reduce the time complexity from quadratic to linear.

— Steele revappend —

```
(define (revappend xs ys) ; linear in (length xs)
  (cond ((null? xs) ys)
        (else (revappend (cdr xs) (cons (car xs) ys)))))
```

—

and use it to define reverse:

— Steele reverse 2 —

```
(define (reverse xs) ; linear in (length xs)
  (revappend xs '()))
```

—

5.3.1 Parallel mapreduce

— Steele parallel mapreduce —

```
(define (mapreduce f g id xs) ; logarithmic in (length xs)?
  (cond ((null? xs) id)
        ((singleton? xs) (f (item xs)))
        (else (split xs (lambda (ys zs)
          (g (mapreduce f g id ys) ; opportunity for
            (mapreduce f g id zs))))))) ; parallelism
```

—

— Steele parallel map —

```
(define (map f xs)
  (mapreduce (lambda (x) (list (f x))) append '() xs)) ; or conc
```

—

— Steele parallel reduce —

```
(define (reduce g id xs)
  (mapreduce (lambda (x) x) g id xs))
```

— Steele parallel length —

```
(define (length xs) ; logarithmic in (length xs) ??
  (mapreduce (lambda (q) 1) + 0 xs))
```

— Steele parallel filter —

```
(define (filter p xs) ; logarithmic in (length xs) ??
  (mapreduce (lambda (x) (if (p x) (list x) '()))
    append '() xs))
```

— Steele parallel reverse —

```
(define (reverse xs) ; logarithmic in (length xs) ??
  (mapreduce list (lambda (ys zx) (append zs ys)) '() xs))
```

- for filter, unlike summation, we rely on maintaining the original order of the elements in the input list. Thus \parallel and $+$ are associative, but only $+$ is commutative.
- do not confuse the ordering of elements in the result list (which is a spatial order) with the order in which they are computed (which is a temporal order)
- sequential programming often ties spatial order to temporal order
- it is incorrect to say that since I don't pay attention to the order in which things are done then we require commutativity. But commutativity has to do with the spatial ordering in the result, not the temporal order in which they are computed.

Steele defines “Conjugate Transforms” which, instead of mapping input items directly to the output data type T:

- map inputs (maybe by way of T) to a richer data type U

- perform the computation in this richer space U (chosen to make computation simpler or faster)
- finally, project the result from U back into T

Thought: he is just wrapping T with a structure for side information

Thought: much effort needs to be placed on the combining operator

5.4 MPI Message Passing

5.4.1 The N-colony Ant Demo

5.5 MapReduce

Chapter 6

The ant build sequence

6.1 ant.build

— build.xml —

```
<project name="clojure" default="all" xmlns:mvn="urn:maven-artifact-ant">

<description>
  Build with "ant jar" and then start the
  REPL with: "java -cp clojure.jar clojure.main".
  You will need to install the Maven Ant
  Tasks to ${ant.home}/lib in order to execute
  the nightly-build or stable-build targets.
</description>

<property name="src" location="src"/>
<property name="test" location="test"/>
<property name="jsrc" location="${src}/jvm"/>
<property name="cljsrc" location="${src}/cljs"/>
<property name="build" location="classes"/>
<property name="test-classes" location="test-classes"/>
<property name="dist" location="dist"/>
<!-- override this with
     -Ddeployment.url=scp://build.clojure.org/srv/test-deploy
     to test a deployment -->
<property name="deployment.url"
          value="scp://build.clojure.org/srv/www/releases"/>

<target name="debug">
  <echo message="${deployment.url}" />
</target>
```

```

<!-- version related properties -->
<property file="${cljsrc}/clojure/version.properties"/>
<!-- ensures all version properties are present -->
<fail unless="clojure.version.major"/>
<fail unless="clojure.version.minor"/>
<fail unless="clojure.version.interim"/>

<condition property="clojure.version.incremental.label"
            value=".${clojure.version.incremental}"
            else="">
    <length string="${clojure.version.incremental}"
            when="greater" length="0" />
</condition>
<condition property="clojure.version.qualifier.label"
            value="-${clojure.version.qualifier}"
            else="">
    <length string="${clojure.version.qualifier}" when="greater"
            length="0" />
</condition>
<condition property="clojure.version.interim.label"
            value="-SNAPSHOT"
            else="">
    <!-- We place -SNAPSHOT whenever interim is not set to false, not
        only if interim is set to true (this is less typo prone in the
        worst case -->
    <not><equals arg1="${clojure.version.interim}" arg2="false"
                trim="true"/></not>
</condition>

<property name="cvm"      value="${clojure.version.major}"/>
<property name="cvm1"     value="${clojure.version.minor}"/>
<property name="cvil"     value="${clojure.version.incremental.label}"/>
<property name="cvql"     value="${clojure.version.qualifier.label}"/>
<property name="cvil1"    value="${clojure.version.interim.label}"/>
<property name="clojure.version.label"
          value="${cvm}.${cvm1}${cvil}${cvql}${cvil1}"/>

<!-- general filterset for use when clojure version must be copied -->
<filterset id="clojure-version-filterset">
    <filter token="clojure-version" value="${clojure.version.label}"/>
</filterset>

<property name="clojure_noversion_jar" location="clojure.jar"/>
<property name="slim_noversion_jar" location="clojure-slim.jar"/>
<property name="src_noversion_jar" location="clojure-sources.jar"/>
<property name="clojure_jar"
          location="clojure-${clojure.version.label}.jar"/>
<property name="slim_jar"
          location="clojure-slim-${clojure.version.label}.jar"/>
<property name="src_jar"

```

```
location="clojure-sources-${clojure.version.label}.jar"/>

<!-- These make sense for building on tapestry.formos.com -->

<property name="snapshot.repo.dir"
          location="/var/www/maven-snapshot-repository"/>
<property name="stable.repo.dir" location="/var/www/maven-repository"/>

<target name="init" depends="clean">
    <tstamp/>
    <mkdir dir="${build}" />
    <antcall target="init-version"/>
</target>

<target name="init-version">
    <copy file="pom-template.xml"
          tofile="pom.xml">
        <filterset refid="clojure-version-filterset"/>
    </copy>
    <!--prevents users from modifying accidentally the generated
        pom.xml works only on linux.-->
    <chmod file="pom.xml" perm="ugo-w"/>
</target>

<target name="compile-java" depends="init"
        description="Compile Java sources.">
    <javac srcdir="${jsrc}" destdir="${build}" includeJavaRuntime="yes"
          debug="true" target="1.5"/>
</target>

<target name="compile-clojure" depends="compile-java"
        description="Compile Clojure sources.">
    <java classname="clojure.lang.Compile"
          classpath="${build}:${cljsrc}"
          failonerror="true"
          fork="true">
        <sysproperty key="clojure.compile.path" value="${build}"/>
        <!-- <sysproperty key="clojure.compile.warn-on-reflection"
                      value="true"/> -->
        <arg value="clojure.core"/>
        <arg value="clojure.core.protocols"/>
        <arg value="clojure.main"/>
        <arg value="clojure.set"/>
        <arg value="clojure.xml"/>
        <arg value="clojure.zip"/>
        <arg value="clojure.inspector"/>
        <arg value="clojure.walk"/>
        <arg value="clojure.stacktrace"/>
        <arg value="clojure.template"/>
        <arg value="clojure.test"/>
```

```

<arg value="clojure.test.tap"/>
<arg value="clojure.test.junit"/>
<arg value="clojure.pprint"/>
<arg value="clojure.java.io"/>
<arg value="clojure.repl"/>
<arg value="clojure.java/browse"/>
<arg value="clojure.java.javadoc"/>
<arg value="clojure.java.shell"/>
<arg value="clojure.java/browse-ui"/>
<arg value="clojure.string"/>
<arg value="clojure.data"/>
<arg value="clojure.reflect"/>
</java>
</target>

<target name="build"
       description="Build Clojure (compilation only, no tests)."
       depends="compile-java, compile-clojure"/>

<target name="compile-tests"
       description="Compile the subset of tests that require compilation.">
<delete dir="${test-classes}"/>
<mkdir dir="${test-classes}"/>
<java classname="clojure.lang.Compile"
      classpath="${test-classes}:${test}:${build}:${cljsrc}"
      failonerror="true">
<sysproperty key="clojure.compile.path" value="${test-classes}"/>
<arg value="clojure.test-clojure.protocols.examples"/>
<arg value="clojure.test-clojure.genclass.examples"/>
</java>
</target>

<target name="test"
       description="Run clojure tests without recompiling clojure."
       depends="compile-tests">
<java classname="clojure.main" failonerror="true">
<classpath>
<path location="${test-classes}"/>
<path location="${test}"/>
<path location="${build}"/>
<path location="${cljsrc}"/>
</classpath>
<arg value="-e"/>
<arg value=
      "(require '(clojure [test-clojure :as main])) (main/run-ant)"/>
</java>
</target>

<target name="clojure-jar" depends="build"
       description="Create clojure jar file.">

```

```
<jar jarfile="${clojure_jar}" basedir="${build}">
  <fileset dir="${cljsrc}">
    <include name="**/*.cljs"/>
    <include name="clojure/version.properties"/>
  </fileset>
  <manifest>
    <attribute name="Main-Class" value="clojure.main"/>
    <attribute name="Class-Path" value=". "/>
  </manifest>
</jar>
<copy file="${clojure_jar}" tofile="${clojure_noVERSION_jar}" />
</target>

<target name="clojure-jar-slim" depends="build"
       description=
         "Create clojure-slim jar file (omits compiled Clojure code).">
  <jar jarfile="${slim_jar}">
    <fileset dir="${build}" includes="clojure/asm/**"/>
    <fileset dir="${build}" includes="clojure/lang/**"/>
    <fileset dir="${build}" includes="clojure/main.class"/>
    <fileset dir="${cljsrc}">
      <include name="**/*.cljs"/>
      <include name="clojure/version.properties"/>
    </fileset>
    <manifest>
      <attribute name="Main-Class" value="clojure.main"/>
      <attribute name="Class-Path" value=". "/>
    </manifest>
  </jar>
  <copy file="${slim_jar}" tofile="${slim_noVERSION_jar}" />
</target>

<target name="clojure-jar-sources" depends="build"
       description="Create a JAR of Java sources.">
  <jar jarfile="${src_jar}" basedir="${jsrc}" includes="**/*">
    <fileset dir="${cljsrc}">
      includes="clojure/version.properties"/>
  </jar>
  <copy file="${src_jar}" tofile="${src_noVERSION_jar}" />
</target>

<target name="all"
       depends=
         "build,test,clojure-jar,clojure-jar-slim,clojure-jar-sources"/>

<target name="clean"
       description="Remove autogenerated files and directories.">
  <delete dir="${build}" />
  <delete dir="${test-classes}" />
  <delete dir="${dist}" />
```

```

<delete file="pom.xml"/>
<delete verbose="true">
  <fileset dir="${basedir}" includes="*.jar"/>
  <fileset dir="${basedir}" includes="*.zip"/>
</delete>
</target>

<target name="setup-maven">
  <typedef resource="org/apache/maven/artifact/ant/antlib.xml"
    uri="urn:maven-artifact-ant"/>
</target>

<target name="nightly-build" depends="ci-build"
  description="Build and deploy to nightly (snapshot) repository.">
  <sequential>
    <typedef resource="org/apache/maven/artifact/ant/antlib.xml"
      uri="urn:maven-artifact-ant"/>
    <mvn:deploy file="${closure_jar}">
      <pom file="pom.xml"/>
      <attach file="${src_jar}" classifier="sources"/>
      <attach file="${slim_jar}" classifier="slim"/>
      <remoteRepository url="file:${snapshot.repo.dir}" />
    </mvn:deploy>
  </sequential>
</target>

<target name="release" depends="ci-build,dist"
  description="Build and deploy to remote stable repository.">
  <sequential>
    <typedef resource="org/apache/maven/artifact/ant/antlib.xml"
      uri="urn:maven-artifact-ant"/>
    <mvn:install-provider artifactId="wagon-ssh" version="1.0-beta-2"/>
    <echo message="Deploying to ${deployment.url}"/>
    <mvn:deploy file="${closure_jar}">
      <pom file="pom.xml"/>
      <attach file="${src_jar}" classifier="sources"/>
      <attach file="${slim_jar}" classifier="slim"/>
      <remoteRepository url="${deployment.url}">
        <authentication username="root"
          privateKey="${user.home}/.ssh/id_rsa"/>
      </remoteRepository>
    </mvn:deploy>
  </sequential>
</target>

<target name="ci-build" depends="clean,all,setup-maven"
  description=
    "Continous integration build, installed to local repository.">
  <mvn:install file="${closure_jar}">
    <pom file="pom.xml"/>

```

```

<attach file="${src_jar}" classifier="sources"/>
<attach file="${slim_jar}" classifier="slim"/>
</mvn:install>
</target>

<target name="dist" depends="clean,clojure-jar"
       description="Build distribution ZIP file.">
  <property name="distdir"
            value="${dist}/clojure-${clojure.version.label}"/>
  <mkdir dir="${distdir}" />
  <copy todir="${distdir}" includeEmptyDirs="false">
    <fileset dir="${basedir}">
      <exclude name="pom.xml"/>
      <exclude name="**/.git/**"/>
      <exclude name="**/*.class"/>
      <exclude name="**/*.iml"/>
      <exclude name="**/*.ipr"/>
      <exclude name="**/*.iws"/>
      <exclude name="**/*.jar"/>
    </fileset>
  </copy>
  <copy file="${clojure_noversion_jar}" todir="${distdir}" />
  <zip basedir="${dist}" destfile="clojure-${clojure.version.label}.zip"/>
</target>
</project>

```

6.2 The Execution

```

ant -v build
Apache Ant version 1.7.1 compiled on November 10 2008
Buildfile: build.xml
Detected Java version: 1.6 in: /usr/lib/jvm/java-6-sun-1.6.0.13/jre
Detected OS: Linux
parsing buildfile BASE/build.xml
    with URI = file:BASE/build.xml
Project base dir set to: BASE
[antlib:org.apache.tools.ant] Could not load definitions from resource
        org/apache/tools/ant/antlib.xml. It could not be found.
[property] Loading BASE/src/clj/clojure/version.properties
Build sequence for target(s) 'build' is
    [clean, init, compile-java, compile-clojure, build]
Complete build sequence is
    [clean, init, compile-java, compile-clojure, build, setup-maven,
     init-version, compile-tests, test, clojure-jar, clojure-jar-slim,
     clojure-jar-sources, all, ci-build, nightly-build, debug, dist,

```

```
release, ]  
  
clean:  
    [delete] Could not find file BASE/pom.xml to delete.  
  
init:  
    [mkdir] Created dir: BASE/classes  
Project base dir set to: BASE  
    [antcall] calling target(s) [init-version] in build file  
        BASE/build.xml  
parsing buildfile BASE/build.xml with  
    URI = file:BASE/build.xml  
Project base dir set to: BASE  
Override ignored for property "src"  
Override ignored for property "test"  
Override ignored for property "jsrc"  
Override ignored for property "cljsrc"  
Override ignored for property "build"  
Override ignored for property "test-classes"  
Override ignored for property "dist"  
Override ignored for property "deployment.url"  
    [property] Loading BASE/src/clj/clojure/version.properties  
Override ignored for property "clojure.version.qualifier"  
Override ignored for property "clojure.version.major"  
Override ignored for property "clojure.version.interim"  
Override ignored for property "clojure.version.incremental"  
Override ignored for property "clojure.version.minor"  
Override ignored for property "clojure.version.incremental.label"  
Override ignored for property "clojure.version.qualifier.label"  
Override ignored for property "clojure.version.interim.label"  
Override ignored for property "clojure.version.label"  
Override ignored for property "clojure_noversion_jar"  
Override ignored for property "slim_noversion_jar"  
Override ignored for property "src_noversion_jar"  
Override ignored for property "clojure_jar"  
Override ignored for property "slim_jar"  
Override ignored for property "src_jar"  
Override ignored for property "snapshot.repo.dir"  
Override ignored for property "stable.repo.dir"  
Build sequence for target(s) 'init-version' is [init-version]  
Complete build sequence is  
    [init-version, setup-maven, clean, init, compile-java,  
     compile-clojure, build, compile-tests, test, clojure-jar,  
     clojure-jar-slim, clojure-jar-sources, all, ci-build,  
     nightly-build, debug, dist, release, ]  
    [antcall] Entering BASE/build.xml...  
Build sequence for target(s) 'init-version' is [init-version]  
Complete build sequence is  
    [init-version, setup-maven, clean, init, compile-java,  
     compile-clojure, build, compile-tests, test, clojure-jar,
```

```
clojure-jar-slim, clojure-jar-sources, all, ci-build,
nightly-build, debug, dist, release, ]  
  
init-version:  
  [copy] Copying 1 file to BASE  
  [copy] Copying BASE/pom-template.xml to BASE/pom.xml  
Replacing: @clojure-version@ -> 1.3.0-master-SNAPSHOT  
  [chmod] Current OS is Linux  
  [chmod] Executing 'chmod' with arguments:  
  [chmod] 'ugo-w'  
  [chmod] 'BASE/pom.xml'  
  [chmod]  
  [chmod] The ' characters around the executable and arguments are  
  [chmod] not part of the command.  
  [chmod] Applied chmod to 1 file and 0 directories.  
[antcall] Exiting BASE/build.xml.  
  
compile-java:  
  [javac] clojure/asm/AnnotationVisitor.java added as  
         clojure/asm/AnnotationVisitor.class doesn't exist.  
  [javac] clojure/asm/AnnotationWriter.java added as  
         clojure/asm/AnnotationWriter.class doesn't exist.  
  [javac] clojure/asm/Attribute.java added as  
         clojure/asm/Attribute.class doesn't exist.  
  [javac] clojure/asm/ByteVector.java added as  
         clojure/asm/ByteVector.class doesn't exist.  
  [javac] clojure/asm/ClassAdapter.java added as  
         clojure/asm/ClassAdapter.class doesn't exist.  
  [javac] clojure/asm/ClassReader.java added as  
         clojure/asm/ClassReader.class doesn't exist.  
  [javac] clojure/asm/ClassVisitor.java added as  
         clojure/asm/ClassVisitor.class doesn't exist.  
  [javac] clojure/asm/ClassWriter.java added as  
         clojure/asm/ClassWriter.class doesn't exist.  
  [javac] clojure/asm/Edge.java added as  
         clojure/asm/Edge.class doesn't exist.  
  [javac] clojure/asm/FieldVisitor.java added as  
         clojure/asm/FieldVisitor.class doesn't exist.  
  [javac] clojure/asm/FieldWriter.java added as  
         clojure/asm/FieldWriter.class doesn't exist.  
  [javac] clojure/asm/Frame.java added as  
         clojure/asm/Frame.class doesn't exist.  
  [javac] clojure/asm/Handler.java added as  
         clojure/asm/Handler.class doesn't exist.  
  [javac] clojure/asm/Item.java added as  
         clojure/asm/Item.class doesn't exist.  
  [javac] clojure/asm/Label.java added as  
         clojure/asm/Label.class doesn't exist.  
  [javac] clojure/asm/MethodAdapter.java added as  
         clojure/asm/MethodAdapter.class doesn't exist.
```

```
[javac] clojure/asm/MethodVisitor.java added as
clojure/asm/MethodVisitor.class doesn't exist.
[javac] clojure/asm/MethodWriter.java added as
clojure/asm/MethodWriter.class doesn't exist.
[javac] clojure/asm/Opcodes.java added as
clojure/asm/Opcodes.class doesn't exist.
[javac] clojure/asm/Type.java added as
clojure/asm/Type.class doesn't exist.
[javac] clojure/asm/commons/AdviceAdapter.java added as
clojure/asm/commons/AdviceAdapter.class doesn't exist.
[javac] clojure/asm/commons/AnalyzerAdapter.java added as
clojure/asm/commons/AnalyzerAdapter.class doesn't exist.
[javac] clojure/asm/commons/CodeSizeEvaluator.java added as
clojure/asm/commons/CodeSizeEvaluator.class doesn't exist.
[javac] clojure/asm/commons/EmptyVisitor.java added as
clojure/asm/commons/EmptyVisitor.class doesn't exist.
[javac] clojure/asm/commons/GeneratorAdapter.java added as
clojure/asm/commons/GeneratorAdapter.class doesn't exist.
[javac] clojure/asm/commons/LocalVariablesSorter.java added as
clojure/asm/commons/LocalVariablesSorter.class doesn't exist.
[javac] clojure/asm/commons/Method.java added as
clojure/asm/commons/Method.class doesn't exist.
[javac] clojure/asm/commons/SerialVersionUIDAdder.java added as
clojure/asm/commons/SerialVersionUIDAdder.class
doesn't exist.
[javac] clojure/asm/commons/StaticInitMerger.java added as
clojure/asm/commons/StaticInitMerger.class doesn't exist.
[javac] clojure/asm/commons/TableSwitchGenerator.java added as
clojure/asm/commons/TableSwitchGenerator.class doesn't exist.
[javac] BASE/src/jvm/clojure/asm/commons/package.html
skipped - don't know how to handle it
[javac] BASE/src/jvm/clojure/asm/package.html
skipped - don't know how to handle it
[javac] clojure/lang/AFn.java added as
clojure/lang/AFn.class doesn't exist.
[javac] clojure/lang/AFunction.java added as
clojure/lang/AFunction.class doesn't exist.
[javac] clojure/lang/AMapEntry.java added as
clojure/lang/AMapEntry.class doesn't exist.
[javac] clojure/lang/APersistentMap.java added as
clojure/lang/APersistentMap.class doesn't exist.
[javac] clojure/lang/APersistentSet.java added as
clojure/lang/APersistentSet.class doesn't exist.
[javac] clojure/lang/APersistentVector.java added as
clojure/lang/APersistentVector.class doesn't exist.
[javac] clojure/lang/ARef.java added as
clojure/lang/ARef.class doesn't exist.
[javac] clojure/lang/AReference.java added as
clojure/lang/AReference.class doesn't exist.
[javac] clojure/lang/ASeq.java added as
```

```
clojure/lang/ASeq.class doesn't exist.  
[javac] clojure/lang/ATransientMap.java added as  
clojure/lang/ATransientMap.class doesn't exist.  
[javac] clojure/lang/ATransientSet.java added as  
clojure/lang/ATransientSet.class doesn't exist.  
[javac] clojure/lang/Agent.java added as  
clojure/lang/Agent.class doesn't exist.  
[javac] clojure/lang/ArityException.java added as  
clojure/lang/ArityException.class doesn't exist.  
[javac] clojure/lang/ArrayChunk.java added as  
clojure/lang/ArrayChunk.class doesn't exist.  
[javac] clojure/lang/ArraySeq.java added as  
clojure/lang/ArraySeq.class doesn't exist.  
[javac] clojure/lang/Associative.java added as  
clojure/lang/Associative.class doesn't exist.  
[javac] clojure/lang/Atom.java added as  
clojure/lang/Atom.class doesn't exist.  
[javac] clojure/lang/BigInt.java added as  
clojure/lang/BigInt.class doesn't exist.  
[javac] clojure/lang/Binding.java added as  
clojure/lang/Binding.class doesn't exist.  
[javac] clojure/lang/Box.java added as  
clojure/lang/Box.class doesn't exist.  
[javac] clojure/lang/ChunkBuffer.java added as  
clojure/lang/ChunkBuffer.class doesn't exist.  
[javac] clojure/lang/ChunkedCons.java added as  
clojure/lang/ChunkedCons.class doesn't exist.  
[javac] clojure/lang/Compile.java added as  
clojure/lang/Compile.class doesn't exist.  
[javac] clojure/lang/Compiler.java added as  
clojure/lang/Compiler.class doesn't exist.  
[javac] clojure/lang/Cons.java added as  
clojure/lang/Cons.class doesn't exist.  
[javac] clojure/lang/Counted.java added as  
clojure/lang/Counted.class doesn't exist.  
[javac] clojure/lang/Delay.java added as  
clojure/lang/Delay.class doesn't exist.  
[javac] clojure/lang/DynamicClassLoader.java added as  
clojure/lang/DynamicClassLoader.class doesn't exist.  
[javac] clojure/lang/EnumerationSeq.java added as  
clojure/lang/EnumerationSeq.class doesn't exist.  
[javac] clojure/lang/Fn.java added as  
clojure/lang/Fn.class doesn't exist.  
[javac] clojure/lang/IChunk.java added as  
clojure/lang/IChunk.class doesn't exist.  
[javac] clojure/lang/IChunkedSeq.java added as  
clojure/lang/IChunkedSeq.class doesn't exist.  
[javac] clojure/lang/IDeref.java added as  
clojure/lang/IDeref.class doesn't exist.  
[javac] clojure/lang/IEditableCollection.java added as
```

```
clojure/lang/IEditableCollection.class doesn't exist.  
[javac] clojure/lang/IFn.java added as  
clojure/lang/IFn.class doesn't exist.  
[javac] clojure/lang/IKeywordLookup.java added as  
clojure/lang/IKeywordLookup.class doesn't exist.  
[javac] clojure/lang/ILookup.java added as  
clojure/lang/ILookup.class doesn't exist.  
[javac] clojure/lang/ILookupSite.java added as  
clojure/lang/ILookupSite.class doesn't exist.  
[javac] clojure/lang/ILookupThunk.java added as  
clojure/lang/ILookupThunk.class doesn't exist.  
[javac] clojure/lang/IMapEntry.java added as  
clojure/lang/IMapEntry.class doesn't exist.  
[javac] clojure/lang/IMeta.java added as  
clojure/lang/IMeta.class doesn't exist.  
[javac] clojure/lang/IObj.java added as  
clojure/lang/IObj.class doesn't exist.  
[javac] clojure/lang/IPersistentCollection.java added as  
clojure/lang/IPersistentCollection.class doesn't exist.  
[javac] clojure/lang/IPersistentList.java added as  
clojure/lang/IPersistentList.class doesn't exist.  
[javac] clojure/lang/IPersistentMap.java added as  
clojure/lang/IPersistentMap.class doesn't exist.  
[javac] clojure/lang/IPersistentSet.java added as  
clojure/lang/IPersistentSet.class doesn't exist.  
[javac] clojure/lang/IPersistentStack.java added as  
clojure/lang/IPersistentStack.class doesn't exist.  
[javac] clojure/lang/IPersistentVector.java added as  
clojure/lang/IPersistentVector.class doesn't exist.  
[javac] clojure/lang/IPromiseImpl.java added as  
clojure/lang/IPromiseImpl.class doesn't exist.  
[javac] clojure/lang/IProxy.java added as  
clojure/lang/IProxy.class doesn't exist.  
[javac] clojure/lang/IReduce.java added as  
clojure/lang/IReduce.class doesn't exist.  
[javac] clojure/lang/IRef.java added as  
clojure/lang/IRef.class doesn't exist.  
[javac] clojure/lang/IReference.java added as  
clojure/lang/IReference.class doesn't exist.  
[javac] clojure/lang/ISeq.java added as  
clojure/lang/ISeq.class doesn't exist.  
[javac] clojure/lang/ITransientAssociative.java added as  
clojure/lang/ITransientAssociative.class doesn't exist.  
[javac] clojure/lang/ITransientCollection.java added as  
clojure/lang/ITransientCollection.class doesn't exist.  
[javac] clojure/lang/ITransientMap.java added as  
clojure/lang/ITransientMap.class doesn't exist.  
[javac] clojure/lang/ITransientSet.java added as  
clojure/lang/ITransientSet.class doesn't exist.  
[javac] clojure/lang/ITransientVector.java added as
```

```
clojure/lang/ITransientVector.class doesn't exist.  
[javac] clojure/lang/Indexed.java added as  
clojure/lang/Indexed.class doesn't exist.  
[javac] clojure/lang/IndexedSeq.java added as  
clojure/lang/IndexedSeq.class doesn't exist.  
[javac] clojure/lang/IteratorSeq.java added as  
clojure/lang/IteratorSeq.class doesn't exist.  
[javac] clojure/lang/Keyword.java added as  
clojure/lang/Keyword.class doesn't exist.  
[javac] clojure/lang/KeywordLookupSite.java added as  
clojure/lang/KeywordLookupSite.class doesn't exist.  
[javac] clojure/lang/LazilyPersistentVector.java added as  
clojure/lang/LazilyPersistentVector.class doesn't exist.  
[javac] clojure/lang/LazySeq.java added as  
clojure/lang/LazySeq.class doesn't exist.  
[javac] clojure/lang/LineNumberingPushbackReader.java added as  
clojure/lang/LineNumberingPushbackReader.class doesn't exist.  
[javac] clojure/lang/LispReader.java added as  
clojure/lang/LispReader.class doesn't exist.  
[javac] clojure/lang/LockingTransaction.java added as  
clojure/lang/LockingTransaction.class doesn't exist.  
[javac] clojure/lang/MapEntry.java added as  
clojure/lang/MapEntry.class doesn't exist.  
[javac] clojure/lang/MapEquivalence.java added as  
clojure/lang/MapEquivalence.class doesn't exist.  
[javac] clojure/lang/MethodImplCache.java added as  
clojure/lang/MethodImplCache.class doesn't exist.  
[javac] clojure/lang/MultiFn.java added as  
clojure/lang/MultiFn.class doesn't exist.  
[javac] clojure/lang/Named.java added as  
clojure/lang/Named.class doesn't exist.  
[javac] clojure/lang/Namespace.java added as  
clojure/lang/Namespace.class doesn't exist.  
[javac] clojure/lang/Numbers.java added as  
clojure/lang/Numbers.class doesn't exist.  
[javac] clojure/lang/Obj.java added as  
clojure/lang/Obj.class doesn't exist.  
[javac] clojure/lang/PersistentArrayMap.java added as  
clojure/lang/PersistentArrayMap.class doesn't exist.  
[javac] clojure/lang/PersistentHashMap.java added as  
clojure/lang/PersistentHashMap.class doesn't exist.  
[javac] clojure/lang/PersistentHashSet.java added as  
clojure/lang/PersistentHashSet.class doesn't exist.  
[javac] clojure/lang/PersistentList.java added as  
clojure/lang/PersistentList.class doesn't exist.  
[javac] clojure/lang/PersistentQueue.java added as  
clojure/lang/PersistentQueue.class doesn't exist.  
[javac] clojure/lang/PersistentStructMap.java added as  
clojure/lang/PersistentStructMap.class doesn't exist.  
[javac] clojure/lang/PersistentTreeMap.java added as
```

```
clojure/lang/PersistentTreeMap.class doesn't exist.  
[javac] clojure/lang/PersistentTreeSet.java added as  
clojure/lang/PersistentTreeSet.class doesn't exist.  
[javac] clojure/lang/PersistentVector.java added as  
clojure/lang/PersistentVector.class doesn't exist.  
[javac] clojure/lang/ProxyHandler.java added as  
clojure/lang/ProxyHandler.class doesn't exist.  
[javac] clojure/lang/RT.java added as  
clojure/lang/RT.class doesn't exist.  
[javac] clojure/lang/Range.java added as  
clojure/lang/Range.class doesn't exist.  
[javac] clojure/lang/Ratio.java added as  
clojure/lang/Ratio.class doesn't exist.  
[javac] clojure/lang/Ref.java added as  
clojure/lang/Ref.class doesn't exist.  
[javac] clojure/lang/Reflector.java added as  
clojure/lang/Reflector.class doesn't exist.  
[javac] clojure/lang/Repl.java added as  
clojure/lang/Repl.class doesn't exist.  
[javac] clojure/lang/RestFn.java added as  
clojure/lang/RestFn.class doesn't exist.  
[javac] clojure/lang/Reversible.java added as  
clojure/lang/Reversible.class doesn't exist.  
[javac] clojure/lang/Script.java added as  
clojure/lang/Script.class doesn't exist.  
[javac] clojure/lang/SeqEnumeration.java added as  
clojure/lang/SeqEnumeration.class doesn't exist.  
[javac] clojure/lang/SeqIterator.java added as  
clojure/lang/SeqIterator.class doesn't exist.  
[javac] clojure/lang/Seqable.java added as  
clojure/lang/Seqable.class doesn't exist.  
[javac] clojure/lang/Sequential.java added as  
clojure/lang/Sequential.class doesn't exist.  
[javac] clojure/lang/Settable.java added as  
clojure/lang/Settable.class doesn't exist.  
[javac] clojure/lang/Sorted.java added as  
clojure/lang/Sorted.class doesn't exist.  
[javac] clojure/lang/StringSeq.java added as  
clojure/lang/StringSeq.class doesn't exist.  
[javac] clojure/lang/Symbol.java added as  
clojure/lang/Symbol.class doesn't exist.  
[javac] clojure/lang/TransactionalHashMap.java added as  
clojure/lang/TransactionalHashMap.class doesn't exist.  
[javac] clojure/lang/Util.java added as  
clojure/lang/Util.class doesn't exist.  
[javac] clojure/lang/Var.java added as  
clojure/lang/Var.class doesn't exist.  
[javac] clojure/lang/XMLHandler.java added as  
clojure/lang/XMLHandler.class doesn't exist.  
[javac] clojure/main.java added as
```

```
clojure/main.class doesn't exist.  
[javac] Compiling 139 source files to BASE/classes  
[javac] Using modern compiler  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/jre/jre/lib/rt.jar from path  
    as it doesn't exist  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/Classes/jce.jar from path  
    as it doesn't exist  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/Classes/jsse.jar from path  
    as it doesn't exist  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/jre/lib/core.jar from path  
    as it doesn't exist  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/jre/lib/graphics.jar from path  
    as it doesn't exist  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/jre/lib/security.jar from path  
    as it doesn't exist  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/jre/lib/server.jar from path  
    as it doesn't exist  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/jre/lib/xml.jar from path  
    as it doesn't exist  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/Classes/classes.jar from path  
    as it doesn't exist  
dropping /usr/lib/jvm/java-6-sun-1.6.0.13/Classes/ui.jar from path  
    as it doesn't exist  
[javac] Compilation arguments:  
[javac] '-d'  
[javac] 'BASE/classes'  
[javac] '-classpath'  
[javac] 'BASE/classes:  
    /usr/share/ant/lib/ant-launcher.jar:  
    /usr/share/java/xmlParserAPIs.jar:  
    /usr/share/java/xercesImpl.jar:  
    /usr/share/ant/lib/ant-antlr.jar:  
    /usr/share/ant/lib/velocity.jar:  
    /usr/share/ant/lib/ant.jar:  
    /usr/share/ant/lib/commons-collections.jar:  
    /usr/share/ant/lib/werkenn.xpath.jar:  
    /usr/share/ant/lib/ant-jsch.jar:  
    /usr/share/ant/lib/ant-apache-regexp.jar:  
    /usr/share/ant/lib/jsch.jar:  
    /usr/share/ant/lib/ant-apache-oro.jar:  
    /usr/share/ant/lib/ant-jdepend.jar:  
    /usr/share/ant/lib/bcel.jar:  
    /usr/share/ant/lib/ant-jmf.jar:  
    /usr/share/ant/lib/ant-javamail.jar:  
    /usr/share/ant/lib/ant-commons-net.jar:  
    /usr/share/ant/lib/jdom0.jar:  
    /usr/share/ant/lib/ant-apache-bsf.jar:  
    /usr/share/ant/lib/ant-swing.jar:  
    /usr/share/ant/lib/logkit.jar:  
    /usr/share/ant/lib/ant-commons-logging.jar:
```

```
/usr/share/ant/lib/ant-apache-log4j.jar:  
/usr/share/ant/lib/ant-apache-resolver.jar:  
/usr/share/ant/lib/ant-trax.jar:  
/usr/share/ant/lib/ant-apache-bcel.jar:  
/usr/share/ant/lib/ant-junit.jar:  
/usr/share/ant/lib/ant-nodeps.jar:  
/usr/lib/jvm/java-6-sun-1.6.0.13/lib/tools.jar:  
/usr/lib/jvm/java-6-sun-1.6.0.13/jre/lib/rt.jar:  
/usr/lib/jvm/java-6-sun-1.6.0.13/jre/lib/jce.jar:  
/usr/lib/jvm/java-6-sun-1.6.0.13/jre/lib/jsse.jar'  
[javac] '-sourcepath'  
[javac] 'BASE/src/jvm'  
[javac] '-target'  
[javac] '1.5'  
[javac] '-g'  
[javac]  
[javac] The ' characters around the executable and arguments are  
[javac] not part of the command.  
[javac] Files to be compiled:  
    BASE/src/jvm/clojure/asm/AnnotationVisitor.java  
    BASE/src/jvm/clojure/asm/AnnotationWriter.java  
    BASE/src/jvm/clojure/asm/Attribute.java  
    BASE/src/jvm/clojure/asm/ByteVector.java  
    BASE/src/jvm/clojure/asm/ClassAdapter.java  
    BASE/src/jvm/clojure/asm/ClassReader.java  
    BASE/src/jvm/clojure/asm/ClassVisitor.java  
    BASE/src/jvm/clojure/asm/ClassWriter.java  
    BASE/src/jvm/clojure/asm/Edge.java  
    BASE/src/jvm/clojure/asm/FieldVisitor.java  
    BASE/src/jvm/clojure/asm/FieldWriter.java  
    BASE/src/jvm/clojure/asm/Frame.java  
    BASE/src/jvm/clojure/asm/Handler.java  
    BASE/src/jvm/clojure/asm/Item.java  
    BASE/src/jvm/clojure/asm/Label.java  
    BASE/src/jvm/clojure/asm/MethodAdapter.java  
    BASE/src/jvm/clojure/asm/MethodVisitor.java  
    BASE/src/jvm/clojure/asm/MethodWriter.java  
    BASE/src/jvm/clojure/asm/Opcodes.java  
    BASE/src/jvm/clojure/asm/Type.java  
    BASE/src/jvm/clojure/asm/commons/AdviceAdapter.java  
    BASE/src/jvm/clojure/asm/commons/AnalyzerAdapter.java  
    BASE/src/jvm/clojure/asm/commons/CodeSizeEvaluator.java  
    BASE/src/jvm/clojure/asm/commons/EmptyVisitor.java  
    BASE/src/jvm/clojure/asm/commons/GeneratorAdapter.java  
    BASE/src/jvm/clojure/asm/commons/LocalVariablesSorter.java  
    BASE/src/jvm/clojure/asm/commons/Method.java  
    BASE/src/jvm/clojure/asm/commons/SerialVersionUIDAdder.java  
    BASE/src/jvm/clojure/asm/commons/StaticInitMerger.java  
    BASE/src/jvm/clojure/asm/commons/TableSwitchGenerator.java  
    BASE/src/jvm/clojure/lang/AFn.java
```

```
BASE/src/jvm/clojure/lang/AFunction.java
BASE/src/jvm/clojure/lang/AMapEntry.java
BASE/src/jvm/clojure/lang/APersistentMap.java
BASE/src/jvm/clojure/lang/APersistentSet.java
BASE/src/jvm/clojure/lang/APersistentVector.java
BASE/src/jvm/clojure/lang/ARef.java
BASE/src/jvm/clojure/lang/AReference.java
BASE/src/jvm/clojure/lang/ASeq.java
BASE/src/jvm/clojure/lang/ATransientMap.java
BASE/src/jvm/clojure/lang/ATransientSet.java
BASE/src/jvm/clojure/lang/Agent.java
BASE/src/jvm/clojure/lang/ArityException.java
BASE/src/jvm/clojure/lang/ArrayChunk.java
BASE/src/jvm/clojure/lang/ArraySeq.java
BASE/src/jvm/clojure/lang/Associative.java
BASE/src/jvm/clojure/lang/Atom.java
BASE/src/jvm/clojure/lang/BigInt.java
BASE/src/jvm/clojure/lang/Binding.java
BASE/src/jvm/clojure/lang/Box.java
BASE/src/jvm/clojure/lang/ChunkBuffer.java
BASE/src/jvm/clojure/lang/ChunkedCons.java
BASE/src/jvm/clojure/lang/Compile.java
BASE/src/jvm/clojure/lang/Compiler.java
BASE/src/jvm/clojure/lang/Cons.java
BASE/src/jvm/clojure/lang/Counted.java
BASE/src/jvm/clojure/lang/Delay.java
BASE/src/jvm/clojure/lang/DynamicClassLoader.java
BASE/src/jvm/clojure/lang/EnumerationSeq.java
BASE/src/jvm/clojure/lang/Fn.java
BASE/src/jvm/clojure/lang/IChunk.java
BASE/src/jvm/clojure/lang/IChunkedSeq.java
BASE/src/jvm/clojure/lang/IDeref.java
BASE/src/jvm/clojure/lang/IEditableCollection.java
BASE/src/jvm/clojure/lang/IFn.java
BASE/src/jvm/clojure/lang/IKeywordLookup.java
BASE/src/jvm/clojure/lang/ILookup.java
BASE/src/jvm/clojure/lang/ILookupSite.java
BASE/src/jvm/clojure/lang/ILookupThunk.java
BASE/src/jvm/clojure/lang/IMapEntry.java
BASE/src/jvm/clojure/lang/IMeta.java
BASE/src/jvm/clojure/lang/IObj.java
BASE/src/jvm/clojure/lang/IPersistentCollection.java
BASE/src/jvm/clojure/lang/IPersistentList.java
BASE/src/jvm/clojure/lang/IPersistentMap.java
BASE/src/jvm/clojure/lang/IPersistentSet.java
BASE/src/jvm/clojure/lang/IPersistentStack.java
BASE/src/jvm/clojure/lang/IPersistentVector.java
BASE/src/jvm/clojure/lang/IPromiseImpl.java
BASE/src/jvm/clojure/lang/IProxy.java
BASE/src/jvm/clojure/lang/IReduce.java
```

```
BASE/src/jvm/clojure/lang/IRef.java  
BASE/src/jvm/clojure/lang/IReference.java  
BASE/src/jvm/clojure/lang/ISeq.java  
BASE/src/jvm/clojure/lang/ITransientAssociative.java  
BASE/src/jvm/clojure/lang/ITransientCollection.java  
BASE/src/jvm/clojure/lang/ITransientMap.java  
BASE/src/jvm/clojure/lang/ITransientSet.java  
BASE/src/jvm/clojure/lang/ITransientVector.java  
BASE/src/jvm/clojure/lang/Indexed.java  
BASE/src/jvm/clojure/lang/IndexedSeq.java  
BASE/src/jvm/clojure/lang/IteratorSeq.java  
BASE/src/jvm/clojure/lang/Keyword.java  
BASE/src/jvm/clojure/lang/KeywordLookupSite.java  
BASE/src/jvm/clojure/lang/LazilyPersistentVector.java  
BASE/src/jvm/clojure/lang/LazySeq.java  
BASE/src/jvm/clojure/lang/LineNumberingPushbackReader.java  
BASE/src/jvm/clojure/lang/LispReader.java  
BASE/src/jvm/clojure/lang/LockingTransaction.java  
BASE/src/jvm/clojure/lang/MapEntry.java  
BASE/src/jvm/clojure/lang/MapEquivalence.java  
BASE/src/jvm/clojure/lang/MethodImplCache.java  
BASE/src/jvm/clojure/lang/MultiFn.java  
BASE/src/jvm/clojure/lang/Named.java  
BASE/src/jvm/clojure/lang/Namespace.java  
BASE/src/jvm/clojure/lang/Numbers.java  
BASE/src/jvm/clojure/lang/Obj.java  
BASE/src/jvm/clojure/lang/PersistentArrayMap.java  
BASE/src/jvm/clojure/lang/PersistentHashMap.java  
BASE/src/jvm/clojure/lang/PersistentHashSet.java  
BASE/src/jvm/clojure/lang/PersistentList.java  
BASE/src/jvm/clojure/lang/PersistentQueue.java  
BASE/src/jvm/clojure/lang/PersistentStructMap.java  
BASE/src/jvm/clojure/lang/PersistentTreeMap.java  
BASE/src/jvm/clojure/lang/PersistentTreeSet.java  
BASE/src/jvm/clojure/lang/PersistentVector.java  
BASE/src/jvm/clojure/lang/ProxyHandler.java  
BASE/src/jvm/clojure/lang/RT.java  
BASE/src/jvm/clojure/lang/Range.java  
BASE/src/jvm/clojure/lang/Ratio.java  
BASE/src/jvm/clojure/lang/Ref.java  
BASE/src/jvm/clojure/lang/Reflector.java  
BASE/src/jvm/clojure/lang/Repl.java  
BASE/src/jvm/clojure/lang/RestFn.java  
BASE/src/jvm/clojure/lang/Reversible.java  
BASE/src/jvm/clojure/lang/Script.java  
BASE/src/jvm/clojure/lang/SeqEnumeration.java  
BASE/src/jvm/clojure/lang/SeqIterator.java  
BASE/src/jvm/clojure/lang/Seqable.java  
BASE/src/jvm/clojure/lang/Sequential.java  
BASE/src/jvm/clojure/lang/Settable.java
```

```
BASE/src/jvm/clojure/lang/Sorted.java
BASE/src/jvm/clojure/lang/StringSeq.java
BASE/src/jvm/clojure/lang/Symbol.java
BASE/src/jvm/clojure/lang/TransactionalHashMap.java
BASE/src/jvm/clojure/lang/Util.java
BASE/src/jvm/clojure/lang/Var.java
BASE/src/jvm/clojure/lang/XMLHandler.java
BASE/src/jvm/clojure/main.java
[javac] Note: Some input files use unchecked or unsafe operations.
[javac] Note: Recompile with -Xlint:unchecked for details.

compile-clojure:
[java] Executing '/usr/lib/jvm/java-6-sun-1.6.0.13/jre/bin/java'
      with arguments:
[java] '-Dclojure.compile.path=BASE/classes'
[java] '-classpath'
[java] 'BASE/classes:BASE/src/clj'
[java] 'clojure.lang.Compile'
[java] 'clojure.core'
[java] 'clojure.core.protocols'
[java] 'clojure.main'
[java] 'clojure.set'
[java] 'clojure.xml'
[java] 'clojure.zip'
[java] 'clojure.inspector'
[java] 'clojure.walk'
[java] 'clojure.stacktrace'
[java] 'clojure.template'
[java] 'clojure.test'
[java] 'clojure.test.tap'
[java] 'clojure.test.junit'
[java] 'clojure.pprint'
[java] 'clojure.java.io'
[java] 'clojure.repl'
[java] 'clojure.java/browse'
[java] 'clojure.java.javadoc'
[java] 'clojure.java.shell'
[java] 'clojure.java/browse-ui'
[java] 'clojure.string'
[java] 'clojure.data'
[java] 'clojure.reflect'
[java]
[java] The ' characters around the executable and arguments are
[java] not part of the command.
[java] Compiling clojure.core to BASE/classes
[java] Compiling clojure.core.protocols to BASE/classes
[java] Compiling clojure.main to BASE/classes
[java] Compiling clojure.set to BASE/classes
[java] Compiling clojure.xml to BASE/classes
[java] Compiling clojure.zip to BASE/classes
```

```
[java] Compiling clojure.inspector to BASE/classes
[java] Compiling clojure.walk to BASE/classes
[java] Compiling clojure.stacktrace to BASE/classes
[java] Compiling clojure.template to BASE/classes
[java] Compiling clojure.test to BASE/classes
[java] Compiling clojure.test.tap to BASE/classes
[java] Compiling clojure.test.junit to BASE/classes
[java] Compiling clojure.pprint to BASE/classes
[java] Compiling clojure.java.io to BASE/classes
[java] Compiling clojure.repl to BASE/classes
[java] Compiling clojure.java/browse to BASE/classes
[java] Compiling clojure.java/javadoc to BASE/classes
[java] Compiling clojure.java.shell to BASE/classes
[java] Compiling clojure.java/browse-ui to BASE/classes
[java] Compiling clojure.string to BASE/classes
[java] Compiling clojure.data to BASE/classes
[java] Compiling clojure.reflect to BASE/classes

build:

BUILD SUCCESSFUL
Total time: 33 seconds
```

Chapter 7

jvm/clojure/asm/

7.1 AnnotationVisitor.java

— AnnotationVisitor.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * A visitor to visit a Java annotation. The methods of this interface
 * must be called in the following order: (<tt>visit<tt> |
 * <tt>visitEnum<tt> | <tt>visitAnnotation<tt> | <tt>visitArray<tt>)*
 * <tt>visitEnd<tt>.
 *
 * @author Eric Bruneton
 * @author Eugene Kuleshov
 */
public interface AnnotationVisitor{

/**
 * Visits a primitive value of the annotation.
 *
 * @param name the value name.
 * @param value the actual value, whose type must be {@link Byte},
 *             {@link Boolean}, {@link Character}, {@link Short},
 *             {@link Integer}, {@link Long}, {@link Float},
 *             {@link Double}, {@link String} or {@link Type}. This
 *             value can also be an array of byte, boolean, short,
 *             char, int, long, float or double values (this is
 *             equivalent to using {@link #visitArray visitArray}
 *             and visiting each array element in turn, but is more
 *             convenient).
 */

}
```

```
/*
void visit(String name, Object value);

/**
 * Visits an enumeration value of the annotation.
 *
 * @param name the value name.
 * @param desc the class descriptor of the enumeration class.
 * @param value the actual enumeration value.
 */
void visitEnum(String name, String desc, String value);

/**
 * Visits a nested annotation value of the annotation.
 *
 * @param name the value name.
 * @param desc the class descriptor of the nested annotation class.
 * @return a visitor to visit the actual nested annotation value, or
 *         <tt>null</tt> if this visitor is not interested in visiting
 *         this nested annotation. <i>The nested annotation value must be
 *         fully visited before calling other methods on this annotation
 *         visitor</i>.
 */
AnnotationVisitor visitAnnotation(String name, String desc);

/**
 * Visits an array value of the annotation. Note that arrays of primitive
 * types (such as byte, boolean, short, char, int, long, float or double)
 * can be passed as value to {@link #visit visit}. This is what
 * {@link ClassReader} does.
 *
 * @param name the value name.
 * @return a visitor to visit the actual array value elements, or
 *         <tt>null</tt> if this visitor is not interested in visiting
 *         these values. The 'name' parameters passed to the methods of
 *         this visitor are ignored. <i>All the array values must be
 *         visited before calling other methods on this annotation
 *         visitor</i>.
 */
AnnotationVisitor visitArray(String name);

/**
 * Visits the end of the annotation.
 */
void visitEnd();
}
```

7.2 AnnotationWriter.java

(AnnotationVisitor [163])
— AnnotationWriter.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * An {@link AnnotationVisitor} that generates annotations in
 * bytecode form.
 *
 * @author Eric Bruneton
 * @author Eugene Kuleshov
 */
final class AnnotationWriter implements AnnotationVisitor{

    /**
     * The class writer to which this annotation must be added.
     */
    private final ClassWriter cw;

    /**
     * The number of values in this annotation.
     */
    private int size;

    /**
     * <tt>true</tt> if values are named, <tt>false</tt> otherwise. Annotation
     * writers used for annotation default and annotation arrays use unnamed
     * values.
     */
    private final boolean named;

    /**
     * The annotation values in bytecode form. This byte vector only contains
     * the values themselves, i.e. the number of values must be stored as a
     * unsigned short just before these bytes.
     */
    private final ByteVector bv;

    /**
     * The byte vector to be used to store the number of values of this
     * annotation. See {@link #bv}.
     */
    private final ByteVector parent;

    /**
     * Where the number of values of this annotation must be stored in
```

```
* {@link #parent}.
```

```
*/
```

```
private final int offset;
```

```
/**
```

```
* Next annotation writer. This field is used to store annotation lists.
```

```
*/
```

```
AnnotationWriter next;
```

```
/**
```

```
* Previous annotation writer. This field is used to store annotation
```

```
* lists.
```

```
*/
```

```
AnnotationWriter prev;
```

```
// -----
```

```
// Constructor
```

```
// -----
```

```
/**
```

```
* Constructs a new {@link AnnotationWriter}.
```

```
*
```

```
* @param cw      the class writer to which this annotation must be added.
```

```
* @param named   <tt>true</tt> if values are named,
```

```
*                 <tt>false</tt> otherwise.
```

```
* @param bv      where the annotation values must be stored.
```

```
* @param parent  where the number of annotation values must be stored.
```

```
* @param offset  where in <tt>parent</tt> the number of annotation values
```

```
*                 must be stored.
```

```
*/
```

```
AnnotationWriter(
```

```
    final ClassWriter cw,
```

```
    final boolean named,
```

```
    final ByteVector bv,
```

```
    final ByteVector parent,
```

```
    final int offset){
```

```
    this.cw = cw;
```

```
    this.named = named;
```

```
    this.bv = bv;
```

```
    this.parent = parent;
```

```
    this.offset = offset;
```

```
}
```

```
// -----
```

```
// Implementation of the AnnotationVisitor interface
```

```
// -----
```

```
public void visit(final String name, final Object value){
```

```
    ++size;
```

```
    if(named)
```

```
{  
    bv.putShort(cw.newUTF8(name));  
}  
if(value instanceof String)  
{  
    bv.put12('s', cw.newUTF8((String) value));  
}  
else if(value instanceof Byte)  
{  
    bv.put12('B', cw.newInteger(((Byte) value).byteValue()).index);  
}  
else if(value instanceof Boolean)  
{  
    int v = ((Boolean) value).booleanValue() ? 1 : 0;  
    bv.put12('Z', cw.newInteger(v).index);  
}  
else if(value instanceof Character)  
{  
    bv.put12('C',  
        cw.newInteger(((Character) value).charValue()).index);  
}  
else if(value instanceof Short)  
{  
    bv.put12('S', cw.newInteger(((Short) value).shortValue()).index);  
}  
else if(value instanceof Type)  
{  
    bv.put12('c', cw.newUTF8(((Type) value).getDescriptor()));  
}  
else if(value instanceof byte[])  
{  
    byte[] v = (byte[]) value;  
    bv.put12('[', v.length);  
    for(int i = 0; i < v.length; i++)  
    {  
        bv.put12('B', cw.newInteger(v[i]).index);  
    }  
}  
else if(value instanceof boolean[])  
{  
    boolean[] v = (boolean[]) value;  
    bv.put12('[', v.length);  
    for(int i = 0; i < v.length; i++)  
    {  
        bv.put12('Z', cw.newInteger(v[i] ? 1 : 0).index);  
    }  
}  
else if(value instanceof short[])  
{  
    short[] v = (short[]) value;
```

```

bv.put12('[, v.length);
for(int i = 0; i < v.length; i++)
{
    bv.put12('S', cw.newInteger(v[i]).index);
}
else if(value instanceof char[])
{
    char[] v = (char[]) value;
    bv.put12('[, v.length);
    for(int i = 0; i < v.length; i++)
    {
        bv.put12('C', cw.newInteger(v[i]).index);
    }
}
else if(value instanceof int[])
{
    int[] v = (int[]) value;
    bv.put12('[, v.length);
    for(int i = 0; i < v.length; i++)
    {
        bv.put12('I', cw.newInteger(v[i]).index);
    }
}
else if(value instanceof long[])
{
    long[] v = (long[]) value;
    bv.put12('[, v.length);
    for(int i = 0; i < v.length; i++)
    {
        bv.put12('J', cw.newLong(v[i]).index);
    }
}
else if(value instanceof float[])
{
    float[] v = (float[]) value;
    bv.put12('[, v.length);
    for(int i = 0; i < v.length; i++)
    {
        bv.put12('F', cw.newFloat(v[i]).index);
    }
}
else if(value instanceof double[])
{
    double[] v = (double[]) value;
    bv.put12('[, v.length);
    for(int i = 0; i < v.length; i++)
    {
        bv.put12('D', cw.newDouble(v[i]).index);
    }
}

```

```

        }
    else
    {
        Item i = cw.newConstItem(value);
        bv.putInt2("".s.IFJDGS".charAt(i.type), i.index);
    }
}

public void visitEnum(
    final String name,
    final String desc,
    final String value){
    ++size;
    if(named)
    {
        bv.putShort(cw.newUTF8(name));
    }
    bv.putInt2('e', cw.newUTF8(desc)).putShort(cw.newUTF8(value));
}

public AnnotationVisitor visitAnnotation(
    final String name,
    final String desc){
    ++size;
    if(named)
    {
        bv.putShort(cw.newUTF8(name));
    }
    // write tag and type, and reserve space for values count
    bv.putInt2('0', cw.newUTF8(desc)).putShort(0);
    return new AnnotationWriter(cw, true, bv, bv, bv.length - 2);
}

public AnnotationVisitor visitArray(final String name){
    ++size;
    if(named)
    {
        bv.putShort(cw.newUTF8(name));
    }
    // write tag, and reserve space for array size
    bv.putInt2('[', 0);
    return new AnnotationWriter(cw, false, bv, bv, bv.length - 2);
}

public void visitEnd(){
    if(parent != null)
    {
        byte[] data = parent.data;
        data[offset] = (byte) (size >> 8);
        data[offset + 1] = (byte) size;
    }
}

```

```
        }
    }

// -----
// Utility methods
// -----


/***
 * Returns the size of this annotation writer list.
 *
 * @return the size of this annotation writer list.
 */
int getSize(){
    int size = 0;
    AnnotationWriter aw = this;
    while(aw != null)
    {
        size += aw.bv.length;
        aw = aw.next;
    }
    return size;
}

/***
 * Puts the annotations of this annotation writer list into the given
 * byte vector.
 *
 * @param out where the annotations must be put.
 */
void put(final ByteVector out){
    int n = 0;
    int size = 2;
    AnnotationWriter aw = this;
    AnnotationWriter last = null;
    while(aw != null)
    {
        ++n;
        size += aw.bv.length;
        aw.visitEnd(); // in case user forgot to call visitEnd
        aw.prev = last;
        last = aw;
        aw = aw.next;
    }
    out.putInt(size);
    out.putShort(n);
    aw = last;
    while(aw != null)
    {
        out.putByteArray(aw.bv.data, 0, aw.bv.length);
        aw = aw.prev;
    }
}
```

```

        }
    }

    /**
     * Puts the given annotation lists into the given byte vector.
     *
     * @param panns an array of annotation writer lists.
     * @param out   where the annotations must be put.
     */
    static void put(final AnnotationWriter[] panns, final ByteVector out){
        int size = 1 + 2 * panns.length;
        for(int i = 0; i < panns.length; ++i)
        {
            size += panns[i] == null ? 0 : panns[i].getSize();
        }
        out.putInt(size).putByte(panns.length);
        for(int i = 0; i < panns.length; ++i)
        {
            AnnotationWriter aw = panns[i];
            AnnotationWriter last = null;
            int n = 0;
            while(aw != null)
            {
                ++n;
                aw.visitEnd(); // in case user forgot to call visitEnd
                aw.prev = last;
                last = aw;
                aw = aw.next;
            }
            out.putShort(n);
            aw = last;
            while(aw != null)
            {
                out.putByteArray(aw.bv.data, 0, aw.bv.length);
                aw = aw.prev;
            }
        }
    }
}

```

7.3 Attribute.java

— Attribute.java —

\getchunk{France Telecom Copyright}

```
package clojure.asm;

/**
 * A non standard class, field, method or code attribute.
 *
 * @author Eric Bruneton
 * @author Eugene Kuleshov
 */
public class Attribute{

    /**
     * The type of this attribute.
     */
    public final String type;

    /**
     * The raw value of this attribute, used only for unknown attributes.
     */
    byte[] value;

    /**
     * The next attribute in this attribute list. May be <tt>null</tt>.
     */
    Attribute next;

    /**
     * Constructs a new empty attribute.
     *
     * @param type the type of the attribute.
     */
    protected Attribute(final String type){
        this.type = type;
    }

    /**
     * Returns <tt>true</tt> if this type of attribute is unknown.
     * The default implementation of this method always returns
     * <tt>true</tt>.
     *
     * @return <tt>true</tt> if this type of attribute is unknown.
     */
    public boolean isUnknown(){
        return true;
    }

    /**
     * Returns <tt>true</tt> if this type of attribute is a code attribute.
     *
     * @return <tt>true</tt> if this type of attribute is a code attribute.
     */
}
```

```

public boolean isCodeAttribute(){
    return false;
}

/**
 * Returns the labels corresponding to this attribute.
 *
 * @return the labels corresponding to this attribute, or
 *         <tt>null</tt> if this attribute is not a code attribute
 *         that contains labels.
 */
protected Label[] getLabels(){
    return null;
}

/**
 * Reads a {@link #type type} attribute. This method must return
 * <i>new</i> {@link Attribute} object, of type {@link #type type},
 * corresponding to the <tt>len</tt> bytes starting at the given offset,
 * in the given class reader.
 *
 * @param cr      the class that contains the attribute to be read.
 * @param off     index of the first byte of the attribute's content
 *               in {@link ClassReader#b cr.b}. The 6 attribute header
 *               bytes, containing the type and the length of the
 *               attribute, are not taken into account
 *               here.
 * @param len     the length of the attribute's content.
 * @param buf     buffer to be used to call
 *               {@link ClassReader#readUTF8 readUTF8},
 *               {@link ClassReader#readClass(int,char[]) readClass} or
 *               {@link ClassReader#readConst readConst}.
 * @param codeOff index of the first byte of code's attribute content in
 *               {@link ClassReader#b cr.b}, or -1 if the attribute to
 *               be read is not a code attribute. The 6 attribute header
 *               bytes, containing the type and the length of the
 *               attribute, are not taken into account here.
 * @param labels  the labels of the method's code, or <tt>null</tt>
 *               if the attribute to be read is not a code attribute.
 * @return a <i>new</i> {@link Attribute} object corresponding to the
 *         given bytes.
 */
protected Attribute read(
    final ClassReader cr,
    final int off,
    final int len,
    final char[] buf,
    final int codeOff,
    final Label[] labels){
    Attribute attr = new Attribute(type);
}

```

```

        attr.value = new byte[len];
        System.arraycopy(cr.b, off, attr.value, 0, len);
        return attr;
    }

    /**
     * Returns the byte array form of this attribute.
     *
     * @param cw           the class to which this attribute must be added.
     *                     This parameter can be used to add to the constant
     *                     pool of this class the items that corresponds to
     *                     this attribute.
     * @param code         the bytecode of the method corresponding to this code
     *                     attribute, or <tt>null</tt> if this attribute is not
     *                     a code attributes.
     * @param len          the length of the bytecode of the method
     *                     corresponding to this code attribute, or
     *                     <tt>null</tt> if this attribute is not a code
     *                     attribute.
     * @param maxStack    the maximum stack size of the method corresponding to
     *                     this code attribute, or -1 if this attribute is not
     *                     a code attribute.
     * @param maxLocals   the maximum number of local variables of the method
     *                     corresponding to this code attribute, or -1 if this
     *                     attribute is not a code attribute.
     * @return the byte array form of this attribute.
     */
    protected ByteVector write(
        final ClassWriter cw,
        final byte[] code,
        final int len,
        final int maxStack,
        final int maxLocals){
        ByteVector v = new ByteVector();
        v.data = value;
        v.length = value.length;
        return v;
    }

    /**
     * Returns the length of the attribute list that begins with this
     *       attribute.
     *
     * @return the length of the attribute list that begins with this
     *       attribute.
     */
    final int getCount(){
        int count = 0;
        Attribute attr = this;
        while(attr != null)

```

```

        {
        count += 1;
        attr = attr.next;
    }
    return count;
}

/**
 * Returns the size of all the attributes in this attribute list.
 *
 * @param cw      the class writer to be used to convert the attributes
 *                into byte arrays, with the {@link #write write}
 *                method.
 * @param code    the bytecode of the method corresponding to these
 *                code attributes, or <tt>null</tt> if these
 *                attributes are not code attributes.
 * @param len     the length of the bytecode of the method
 *                corresponding to these code attributes, or
 *                <tt>null</tt> if these attributes are not code
 *                attributes.
 * @param maxStack the maximum stack size of the method corresponding to
 *                these code attributes, or -1 if these attributes are
 *                not code attributes.
 * @param maxLocals the maximum number of local variables of the method
 *                corresponding to these code attributes, or -1 if
 *                these attributes are not code attributes.
 * @return the size of all the attributes in this attribute list.
 *         This size includes the size of the attribute headers.
 */
final int getSize(
    final ClassWriter cw,
    final byte[] code,
    final int len,
    final int maxStack,
    final int maxLocals){
    Attribute attr = this;
    int size = 0;
    while(attr != null)
    {
        cw.newUTF8(attr.type);
        size +=
            attr.write(cw,code,len,maxStack,maxLocals).length + 6;
        attr = attr.next;
    }
    return size;
}

/**
 * Writes all the attributes of this attribute list in the given byte
 * vector.

```

```

*
* @param cw      the class writer to be used to convert the attributes
*                into byte arrays, with the {@link #write write} method.
*
* @param code    the bytecode of the method corresponding to these
*                code attributes, or <tt>null</tt> if these attributes
*                are not code attributes.
*
* @param len     the length of the bytecode of the method
*                corresponding to these code attributes, or
*                <tt>null</tt> if these attributes are not code
*                attributes.
*
* @param maxStack the maximum stack size of the method corresponding to
*                these code attributes, or -1 if these attributes are
*                not code attributes.
*
* @param maxLocals the maximum number of local variables of the method
*                corresponding to these code attributes, or -1 if
*                these attributes are not code attributes.
*
* @param out      where the attributes must be written.
*/
final void put(
    final ClassWriter cw,
    final byte[] code,
    final int len,
    final int maxStack,
    final int maxLocals,
    final ByteVector out){
    Attribute attr = this;
    while(attr != null)
    {
        ByteVector b = attr.write(cw, code, len, maxStack, maxLocals);
        out.putShort(cw.newUTF8(attr.type)).putInt(b.length);
        out.putByteArray(b.data, 0, b.length);
        attr = attr.next;
    }
}

```

7.4 ByteVector.java

— ByteVector.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm;

/**
```

```
* A dynamically extensible vector of bytes. This class is roughly
* equivalent to a DataOutputStream on top of a ByteArrayOutputStream,
* but is more efficient.
*
* @author Eric Bruneton
*/
public class ByteVector{

    /**
     * The content of this vector.
     */
    byte[] data;

    /**
     * Actual number of bytes in this vector.
     */
    int length;

    /**
     * Constructs a new {@link ByteVector ByteVector} with a default initial
     * size.
     */
    public ByteVector(){
        data = new byte[64];
    }

    /**
     * Constructs a new {@link ByteVector ByteVector} with the given initial
     * size.
     *
     * @param initialSize the initial size of the byte vector to be
     *                    constructed.
     */
    public ByteVector(final int initialSize){
        data = new byte[initialSize];
    }

    /**
     * Puts a byte into this byte vector. The byte vector is automatically
     * enlarged if necessary.
     *
     * @param b a byte.
     * @return this byte vector.
     */
    public ByteVector putByte(final int b){
        int length = this.length;
        if(length + 1 > data.length)
        {
            enlarge(1);
        }
    }
}
```

```
        data[length++] = (byte) b;
        this.length = length;
        return this;
    }

    /**
     * Puts two bytes into this byte vector. The byte vector is automatically
     * enlarged if necessary.
     *
     * @param b1 a byte.
     * @param b2 another byte.
     * @return this byte vector.
     */
    ByteVector put11(final int b1, final int b2){
        int length = this.length;
        if(length + 2 > data.length)
        {
            enlarge(2);
        }
        byte[] data = this.data;
        data[length++] = (byte) b1;
        data[length++] = (byte) b2;
        this.length = length;
        return this;
    }

    /**
     * Puts a short into this byte vector. The byte vector is automatically
     * enlarged if necessary.
     *
     * @param s a short.
     * @return this byte vector.
     */
    public ByteVector putShort(final int s){
        int length = this.length;
        if(length + 2 > data.length)
        {
            enlarge(2);
        }
        byte[] data = this.data;
        data[length++] = (byte) (s >> 8);
        data[length++] = (byte) s;
        this.length = length;
        return this;
    }

    /**
     * Puts a byte and a short into this byte vector. The byte vector is
     * automatically enlarged if necessary.
     *
```

```
* @param b a byte.
* @param s a short.
* @return this byte vector.
*/
ByteVector put12(final int b, final int s){
    int length = this.length;
    if(length + 3 > data.length)
    {
        enlarge(3);
    }
    byte[] data = this.data;
    data[length++] = (byte) b;
    data[length++] = (byte) (s >>> 8);
    data[length++] = (byte) s;
    this.length = length;
    return this;
}

/**
 * Puts an int into this byte vector. The byte vector is automatically
 * enlarged if necessary.
 *
 * @param i an int.
 * @return this byte vector.
 */
public ByteVector.putInt(final int i){
    int length = this.length;
    if(length + 4 > data.length)
    {
        enlarge(4);
    }
    byte[] data = this.data;
    data[length++] = (byte) (i >>> 24);
    data[length++] = (byte) (i >>> 16);
    data[length++] = (byte) (i >>> 8);
    data[length++] = (byte) i;
    this.length = length;
    return this;
}

/**
 * Puts a long into this byte vector. The byte vector is automatically
 * enlarged if necessary.
 *
 * @param l a long.
 * @return this byte vector.
 */
public ByteVector.putLong(final long l){
    int length = this.length;
    if(length + 8 > data.length)
```

```

    {
        enlarge(8);
    }
    byte[] data = this.data;
    int i = (int) (l >>> 32);
    data[length++] = (byte) (i >>> 24);
    data[length++] = (byte) (i >>> 16);
    data[length++] = (byte) (i >>> 8);
    data[length++] = (byte) i;
    i = (int) l;
    data[length++] = (byte) (i >>> 24);
    data[length++] = (byte) (i >>> 16);
    data[length++] = (byte) (i >>> 8);
    data[length++] = (byte) i;
    this.length = length;
    return this;
}

/**
 * Puts an UTF8 string into this byte vector. The byte vector is
 * automatically enlarged if necessary.
 *
 * @param s a String.
 * @return this byte vector.
 */
public ByteVector putUTF8(final String s){
    int charLength = s.length();
    if(length + 2 + charLength > data.length)
    {
        enlarge(2 + charLength);
    }
    int len = length;
    byte[] data = this.data;
    // optimistic algorithm: instead of computing the byte length
    // and then serializing the string (which requires two loops),
    // we assume the byte length is equal to char length (which is
    // the most frequent case), and we start serializing the string
    // right away. During the serialization, if we find that this
    // assumption is wrong, we continue with the general method.
    data[len++] = (byte) (charLength >>> 8);
    data[len++] = (byte) charLength;
    for(int i = 0; i < charLength; ++i)
    {
        char c = s.charAt(i);
        if(c >= '\u001' && c <= '\u177')
        {
            data[len++] = (byte) c;
        }
        else
        {

```

```

int byteLength = i;
for(int j = i; j < charLength; ++j)
{
    c = s.charAt(j);
    if(c >= '\001' && c <= '\177')
    {
        byteLength++;
    }
    else if(c > '\u07FF')
    {
        byteLength += 3;
    }
    else
    {
        byteLength += 2;
    }
}
data[length] = (byte) (byteLength >>> 8);
data[length + 1] = (byte) byteLength;
if(length + 2 + byteLength > data.length)
{
    length = len;
    enlarge(2 + byteLength);
    data = this.data;
}
for(int j = i; j < charLength; ++j)
{
    c = s.charAt(j);
    if(c >= '\001' && c <= '\177')
    {
        data[len++] = (byte) c;
    }
    else if(c > '\u07FF')
    {
        data[len++] =
            (byte) (0xE0 | c >> 12 & 0xF);
        data[len++] =
            (byte) (0x80 | c >> 6 & 0x3F);
        data[len++] =
            (byte) (0x80 | c & 0x3F);
    }
    else
    {
        data[len++] =
            (byte) (0xC0 | c >> 6 & 0x1F);
        data[len++] =
            (byte) (0x80 | c & 0x3F);
    }
}
break;
}

```

```

        }
    }
    length = len;
    return this;
}

/***
 * Puts an array of bytes into this byte vector. The byte vector is
 * automatically enlarged if necessary.
 *
 * @param b    an array of bytes. May be <tt>null</tt> to put <tt>len</tt>
 *             null bytes into this byte vector.
 * @param off index of the fist byte of b that must be copied.
 * @param len number of bytes of b that must be copied.
 * @return this byte vector.
 */
public ByteVector putByteArray(final byte[] b, final int off,
                               final int len){
    if(length + len > data.length)
    {
        enlarge(len);
    }
    if(b != null)
    {
        System.arraycopy(b, off, data, length, len);
    }
    length += len;
    return this;
}

/***
 * Enlarge this byte vector so that it can receive n more bytes.
 *
 * @param size number of additional bytes that this byte vector should be
 *             able to receive.
 */
private void enlarge(final int size){
    int length1 = 2 * data.length;
    int length2 = length + size;
    byte[] newData = new byte[length1 > length2 ? length1 : length2];
    System.arraycopy(data, 0, newData, 0, length);
    data = newData;
}
}

```

7.5 ClassAdapter.java

```
(ClassVisitor [229])
— ClassAdapter.java —

\getchunk{France Telecom Copyright}
package clojure.asm;


public class ClassAdapter implements ClassVisitor{

    /**
     * An empty {@link ClassVisitor} that delegates to another
     * {@link ClassVisitor}. This class can be used as a super class to
     * quickly implement useful class adapter classes, just by overriding
     * the necessary methods.
     *
     * @author Eric Bruneton
     */
    protected ClassVisitor cv;

    /**
     * Constructs a new {@link ClassAdapter} object.
     *
     * @param cv the class visitor to which this adapter must delegate calls.
     */
    public ClassAdapter(final ClassVisitor cv){
        this.cv = cv;
    }

    public void visit(
        final int version,
        final int access,
        final String name,
        final String signature,
        final String superName,
        final String[] interfaces){
        cv.visit(version, access, name, signature, superName, interfaces);
    }

    public void visitSource(final String source, final String debug){
        cv.visitSource(source, debug);
    }

    public void visitOuterClass(
        final String owner,
        final String name,
```

```
        final String desc){
    cv.visitOuterClass(owner, name, desc);
}

public AnnotationVisitor visitAnnotation(
    final String desc,
    final boolean visible){
    return cv.visitAnnotation(desc, visible);
}

public void visitAttribute(final Attribute attr){
    cv.visitAttribute(attr);
}

public void visitInnnerClass(
    final String name,
    final String outerName,
    final String innerName,
    final int access){
    cv.visitInnnerClass(name, outerName, innerName, access);
}

public FieldVisitor visitField(
    final int access,
    final String name,
    final String desc,
    final String signature,
    final Object value){
    return cv.visitField(access, name, desc, signature, value);
}

public MethodVisitor visitMethod(
    final int access,
    final String name,
    final String desc,
    final String signature,
    final String[] exceptions){
    return cv.visitMethod(access, name, desc, signature, exceptions);
}

public void visitEnd(){
    cv.visitEnd();
}
```

7.6 ClassReader.java

— ClassReader.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm;

import java.io.InputStream;
import java.io.IOException;

/**
 * A Java class parser to make a {@link ClassVisitor} visit an
 * existing class. This class parses a byte array conforming to
 * the Java class file format and calls the appropriate visit
 * methods of a given class visitor for each field, method and
 * bytecode instruction encountered.
 *
 * @author Eric Bruneton
 * @author Eugene Kuleshov
 */
public class ClassReader{

    /**
     * Flag to skip method code. If this class is set <code>CODE</code>
     * attribute won't be visited. This can be used, for example, to retrieve
     * annotations for methods and method parameters.
     */
    public final static int SKIP_CODE = 1;

    /**
     * Flag to skip the debug information in the class. If this flag is set
     * the debug information of the class is not visited, i.e. the
     * {@link MethodVisitor#visitLocalVariable visitLocalVariable} and
     * {@link MethodVisitor#visitLineNumber visitLineNumber} methods
     * will not be called.
     */
    public final static int SKIP_DEBUG = 2;

    /**
     * Flag to skip the stack map frames in the class. If this flag is set
     * the stack map frames of the class is not visited, i.e. the
     * {@link MethodVisitor#visitFrame visitFrame} method will not be called.
     * This flag is useful when the {@link ClassWriter#COMPUTE_FRAMES} option
     * is used: it avoids visiting frames that will be ignored and
     * recomputed from scratch in the class writer.
     */
    public final static int SKIP_FRAMES = 4;
```

```

/**
 * Flag to expand the stack map frames. By default stack map frames are
 * visited in their original format (i.e. "expanded" for classes whose
 * version is less than V1_6, and "compressed" for the other classes).
 * If this flag is set, stack map frames are always visited in
 * expanded format (this option adds a decompression/recompression
 * step in ClassReader and ClassWriter which degrades performances
 * quite a lot).
 */
public final static int EXPAND_FRAMES = 8;

/**
 * The class to be parsed. <i>The content of this array must not be
 * modified. This field is intended for {@link Attribute} sub classes,
 * and is normally not needed by class generators or adapters.</i>
 */
public final byte[] b;

/**
 * The start index of each constant pool item in {@link #b b}, plus one.
 * The one byte offset skips the constant pool item tag that indicates
 * its type.
 */
private final int[] items;

/**
 * The String objects corresponding to the CONSTANT_Utf8 items.
 * This cache avoids multiple parsing of a given CONSTANT_Utf8
 * constant pool item, which GREATLY improves performances (by a
 * factor 2 to 3). This caching strategy could be extended to all
 * constant pool items, but its benefit would not be so great for
 * these items (because they are much less expensive to parse than
 * CONSTANT_Utf8 items).
 */
private final String[] strings;

/**
 * Maximum length of the strings contained in the constant pool of the
 * class.
 */
private final int maxStringLength;

/**
 * Start index of the class header information (access, name...) in
 * {@link #b b}.
 */
public final int header;

// -----
// Constructors

```

```
// -----  
  
/**  
 * Constructs a new {@link ClassReader} object.  
 *  
 * @param b the bytecode of the class to be read.  
 */  
public ClassReader(final byte[] b){  
    this(b, 0, b.length);  
}  
  
/**  
 * Constructs a new {@link ClassReader} object.  
 *  
 * @param b the bytecode of the class to be read.  
 * @param off the start offset of the class data.  
 * @param len the length of the class data.  
 */  
public ClassReader(final byte[] b, final int off, final int len){  
    this.b = b;  
    // parses the constant pool  
    items = new int[readUnsignedShort(off + 8)];  
    int n = items.length;  
    strings = new String[n];  
    int max = 0;  
    int index = off + 10;  
    for(int i = 1; i < n; ++i)  
    {  
        items[i] = index + 1;  
        int size;  
        switch(b[index])  
        {  
            case ClassWriter.FIELD:  
            case ClassWriter.METH:  
            case ClassWriter.IMETH:  
            case ClassWriter.INT:  
            case ClassWriter.FLOAT:  
            case ClassWriter.NAME_TYPE:  
                size = 5;  
                break;  
            case ClassWriter.LONG:  
            case ClassWriter.DOUBLE:  
                size = 9;  
                ++i;  
                break;  
            case ClassWriter.UTF8:  
                size = 3 + readUnsignedShort(index + 1);  
                if(size > max)  
                {  
                    max = size;  
                }  
        }  
    }  
}
```

```

        }
        break;
        // case ClassWriter.CLASS:
        // case ClassWriter.STR:
    default:
        size = 3;
        break;
    }
    index += size;
}
maxStringLength = max;
// the class header information starts just after the constant pool
header = index;
}

/**
 * Returns the class's access flags (see {@link Opcodes}). This value may
 * not reflect Deprecated and Synthetic flags when bytecode is before 1.5
 * and those flags are represented by attributes.
 *
 * @return the class access flags
 * @see ClassVisitor#visit(int,int,String,String,String[])
 */
public int getAccess(){
    return readUnsignedShort(header);
}

/**
 * Returns the internal name of the class (see
 * {@link Type#getInternalName()} getInternalName}).
 *
 * @return the internal class name
 * @see ClassVisitor#visit(int,int,String,String,String[])
 */
public String getClassName(){
    return readClass(header + 2, new char[maxStringLength]);
}

/**
 * Returns the internal of name of the super class (see
 * {@link Type#getInternalName()} getInternalName}). For interfaces, the
 * super class is {@link Object}.
 *
 * @return the internal name of super class, or <tt>null</tt> for
 *         {@link Object} class.
 * @see ClassVisitor#visit(int,int,String,String,String[])
 */
public String getSuperName(){
    int n = items[readUnsignedShort(header + 4)];
    return n == 0 ? null : readUTF8(n, new char[maxStringLength]);
}

```

```

}


 * Returns the internal names of the class's interfaces (see
 * {@link Type#getInternalName() getInternalName}).
 *
 * @return the array of internal names for all implemented interfaces or
 *         <tt>null</tt>.
 * @see ClassVisitor#visit(int,int,String,String, String[])
 */
public String[] getInterfaces(){
    int index = header + 6;
    int n = readUnsignedShort(index);
    String[] interfaces = new String[n];
    if(n > 0)
    {
        char[] buf = new char[maxStringLength];
        for(int i = 0; i < n; ++i)
        {
            index += 2;
            interfaces[i] = readClass(index, buf);
        }
    }
    return interfaces;
}


 * Copies the constant pool data into the given {@link ClassWriter}.
 * Should be called before the {@link #accept(ClassVisitor,int)} method.
 *
 * @param classWriter the {@link ClassWriter} to copy constant pool into.
 */
void copyPool(final ClassWriter classWriter){
    char[] buf = new char[maxStringLength];
    int ll = items.length;
    Item[] items2 = new Item[ll];
    for(int i = 1; i < ll; i++)
    {
        int index = items[i];
        int tag = b[index - 1];
        Item item = new Item(i);
        int nameType;
        switch(tag)
        {
            case ClassWriter.FIELD:
            case ClassWriter.METH:
            case ClassWriter.IMETH:
                nameType = items[readUnsignedShort(index + 2)];
                item.set(tag,
                    readClass(index, buf),


```

```
        readUTF8(nameType, buf),
        readUTF8(nameType + 2, buf));
    break;

case ClassWriter.INT:
    item.set(readInt(index));
    break;

case ClassWriter.FLOAT:
    item.set(Float.intBitsToFloat(readInt(index)));
    break;

case ClassWriter.NAME_TYPE:
    item.set(tag,
            readUTF8(index, buf),
            readUTF8(index + 2, buf),
            null);
    break;

case ClassWriter.LONG:
    item.set(readLong(index));
    ++i;
    break;

case ClassWriter.DOUBLE:
    item.set(Double.longBitsToDouble(readLong(index)));
    ++i;
    break;

case ClassWriter.UTF8:
{
    String s = strings[i];
    if(s == null)
    {
        index = items[i];
        s = strings[i] = readUTF(index + 2,
                                 readUnsignedShort(index),
                                 buf);
    }
    item.set(tag, s, null, null);
}
break;

// case ClassWriter.STR:
// case ClassWriter.CLASS:
default:
    item.set(tag, readUTF8(index, buf), null, null);
    break;
}
```

```
        int index2 = item.hashCode % items2.length;
        item.next = items2[index2];
        items2[index2] = item;
    }

    int off = items[1] - 1;
    classWriter.pool.putByteArray(b, off, header - off);
    classWriter.items = items2;
    classWriter.threshold = (int) (0.75d * ll);
    classWriter.index = ll;
}


```

```

        if(n == -1)
        {
            if(len < b.length)
            {
                byte[] c = new byte[len];
                System.arraycopy(b, 0, c, 0, len);
                b = c;
            }
            return b;
        }
        len += n;
        if(len == b.length)
        {
            byte[] c = new byte[b.length + 1000];
            System.arraycopy(b, 0, c, 0, len);
            b = c;
        }
    }
}

// -----
// Public methods
// -----


/**
 * Makes the given visitor visit the Java class of this
 * {@link ClassReader}. This class is the one specified in the
 * constructor (see {@link #ClassReader(byte[]) ClassReader}).
 *
 * @param classVisitor the visitor that must visit this class.
 * @param flags          option flags that can be used to modify
 *                       the default behavior of this class. See
 *                       {@link #SKIP_DEBUG}, {@link #EXPAND_FRAMES}.
 */
public void accept(final ClassVisitor classVisitor, final int flags){
    accept(classVisitor, new Attribute[0], flags);
}

/**
 * Makes the given visitor visit the Java class of this
 * {@link ClassReader}. This class is the one specified in the
 * constructor (see {@link #ClassReader(byte[]) ClassReader}).
 *
 * @param classVisitor the visitor that must visit this class.
 * @param attrs         prototypes of the attributes that must be parsed
 *                      during the visit of the class. Any attribute whose
 *                      type is not equal to the type of one the
 *                      prototypes will not be parsed: its byte array
 *                      value will be passed unchanged to the ClassWriter.
 *                      <i>This may corrupt it if this value contains

```

```

*
*           references to the constant pool, or has syntactic
*           or semantic links with a class element that has
*           been transformed by a class adapter between the
*           reader and the writer</i>.
*
* @param flags      option flags that can be used to modify the
*                   default behavior of this class. See
*                   {@link #SKIP_DEBUG}, {@link #EXPAND_FRAMES}.
*/
public void accept(
    final ClassVisitor classVisitor,
    final Attribute[] attrs,
    final int flags){
    byte[] b = this.b; // the bytecode array
    char[] c = new char[maxStringLength]; // buffer used to read strings
    int i, j, k; // loop variables
    int u, v, w; // indexes in b
    Attribute attr;

    int access;
    String name;
    String desc;
    String attrName;
    String signature;
    int anns = 0;
    int ianns = 0;
    Attribute cattrs = null;

    // visits the header
    u = header;
    access = readUnsignedShort(u);
    name = readClass(u + 2, c);
    v = items[readUnsignedShort(u + 4)];
    String superClassNome = v == 0 ? null : readUTF8(v, c);
    String[] implementedItfs = new String[readUnsignedShort(u + 6)];
    w = 0;
    u += 8;
    for(i = 0; i < implementedItfs.length; ++i)
    {
        implementedItfs[i] = readClass(u, c);
        u += 2;
    }

    boolean skipCode = (flags & SKIP_CODE) != 0;
    boolean skipDebug = (flags & SKIP_DEBUG) != 0;
    boolean unzip = (flags & EXPAND_FRAMES) != 0;

    // skips fields and methods
    v = u;
    i = readUnsignedShort(v);
    v += 2;
}

```

```

for(; i > 0; --i)
{
    j = readUnsignedShort(v + 6);
    v += 8;
    for(; j > 0; --j)
    {
        v += 6 + readInt(v + 2);
    }
}
i = readUnsignedShort(v);
v += 2;
for(; i > 0; --i)
{
    j = readUnsignedShort(v + 6);
    v += 8;
    for(; j > 0; --j)
    {
        v += 6 + readInt(v + 2);
    }
}
// reads the class's attributes
signature = null;
String sourceFile = null;
String sourceDebug = null;
String enclosingOwner = null;
String enclosingName = null;
String enclosingDesc = null;

i = readUnsignedShort(v);
v += 2;
for(; i > 0; --i)
{
    attrName = readUTF8(v, c);
    // tests are sorted in decreasing frequency order
    // (based on frequencies observed on typical classes)
    if(attrName.equals("SourceFile"))
    {
        sourceFile = readUTF8(v + 6, c);
    }
    else if(attrName.equals("InnerClasses"))
    {
        w = v + 6;
    }
    else if(attrName.equals("EnclosingMethod"))
    {
        enclosingOwner = readClass(v + 6, c);
        int item = readUnsignedShort(v + 8);
        if(item != 0)
        {
            enclosingName = readUTF8(items[item], c);
        }
    }
}

```

```

        enclosingDesc = readUTF8(items[item] + 2, c);
    }
}
else if(attrName.equals("Signature"))
{
    signature = readUTF8(v + 6, c);
}
else if(attrName.equals("RuntimeVisibleAnnotations"))
{
    anns = v + 6;
}
else if(attrName.equals("Deprecated"))
{
    access |= Opcodes.ACC_DEPRECATED;
}
else if(attrName.equals("Synthetic"))
{
    access |= Opcodes.ACC_SYNTHETIC;
}
else if(attrName.equals("SourceDebugExtension"))
{
    int len = readInt(v + 2);
    sourceDebug = readUTF(v + 6, len, new char[len]);
}
else if(attrName.equals("RuntimeInvisibleAnnotations"))
{
    ianns = v + 6;
}
else
{
    attr = readAttribute(attrs,
                         attrName,
                         v + 6,
                         readInt(v + 2),
                         c,
                         -1,
                         null);
    if(attr != null)
    {
        attr.next = cattrs;
        cattrs = attr;
    }
}
v += 6 + readInt(v + 2);
}
// calls the visit method
classVisitor.visit(readInt(4),
                  access,
                  name,
                  signature,

```

```

        superClassNames,
        implementedItfs);

// calls the visitSource method
if(!skipDebug && (sourceFile != null || sourceDebug != null))
{
    classVisitor.visitSource(sourceFile, sourceDebug);
}

// calls the visitOuterClass method
if(enclosingOwner != null)
{
    classVisitor.visitOuterClass(enclosingOwner,
                                enclosingName,
                                enclosingDesc);
}

// visits the class annotations
for(i = 1; i >= 0; --i)
{
    v = i == 0 ? ianns : anns;
    if(v != 0)
    {
        j = readUnsignedShort(v);
        v += 2;
        for(; j > 0; --j)
        {
            v = readAnnotationValues(v + 2, c, true,
                                      classVisitor.visitAnnotation(readUTF8(v, c), i != 0));
        }
    }
}

// visits the class attributes
while(cattrs != null)
{
    attr = cattrs.next;
    cattrs.next = null;
    classVisitor.visitAttribute(cattrs);
    cattrs = attr;
}

// calls the visitInnerClass method
if(w != 0)
{
    i = readUnsignedShort(w);
    w += 2;
    for(; i > 0; --i)
    {
        classVisitor.visitInnerClass(

```

```

        readUnsignedShort(w) == 0
    ? null
    : readClass(w, c), readUnsignedShort(w + 2) == 0
    ? null
    : readClass(w + 2, c), readUnsignedShort(w + 4) == 0
    ? null
    : readUTF8(w + 4, c),
        readUnsignedShort(w + 6));
w += 8;
}
}

// visits the fields
i = readUnsignedShort(u);
u += 2;
for(; i > 0; --i)
{
access = readUnsignedShort(u);
name = readUTF8(u + 2, c);
desc = readUTF8(u + 4, c);
// visits the field's attributes and looks for a ConstantValue
// attribute
int fieldValueItem = 0;
signature = null;
anns = 0;
ianns = 0;
cattrs = null;

j = readUnsignedShort(u + 6);
u += 8;
for(; j > 0; --j)
{
attrName = readUTF8(u, c);
// tests are sorted in decreasing frequency order
// (based on frequencies observed on typical classes)
if(attrName.equals("ConstantValue"))
{
fieldValueItem = readUnsignedShort(u + 6);
}
else if(attrName.equals("Signature"))
{
signature = readUTF8(u + 6, c);
}
else if(attrName.equals("Deprecated"))
{
access |= Opcodes.ACC_DEPRECATED;
}
else if(attrName.equals("Synthetic"))
{
access |= Opcodes.ACC_SYNTHETIC;
}
}
}

```

```

        }
    else if(attrName.equals("RuntimeVisibleAnnotations"))
    {
        anns = u + 6;
    }
    else if(attrName.equals("RuntimeInvisibleAnnotations"))
    {
        ianns = u + 6;
    }
    else
    {
        attr = readAttribute(attrs,
                             attrName,
                             u + 6,
                             readInt(u + 2),
                             c,
                             -1,
                             null);
        if(attr != null)
        {
            attr.next = cattrs;
            cattrs = attr;
        }
    }
    u += 6 + readInt(u + 2);
}
// visits the field
FieldVisitor fv =
classVisitor.visitField(access, name, desc, signature,
fieldValueItem == 0 ? null : readConst(fieldValueItem, c));
// visits the field annotations and attributes
if(fv != null)
{
    for(j = 1; j >= 0; --j)
    {
        v = j == 0 ? ianns : anns;
        if(v != 0)
        {
            k = readUnsignedShort(v);
            v += 2;
            for(; k > 0; --k)
            {
                v = readAnnotationValues(v + 2, c, true,
                                         fv.visitAnnotation(readUTF8(v, c), j != 0));
            }
        }
    }
    while(cattrs != null)
    {
        attr = cattrs.next;
    }
}

```

```

        cattrs.next = null;
        fv.visitAttribute(cattr);
        cattrs = attr;
    }
    fv.visitEnd();
}
}

// visits the methods
i = readUnsignedShort(u);
u += 2;
for(; i > 0; --i)
{
    int u0 = u + 6;
    access = readUnsignedShort(u);
    name = readUTF8(u + 2, c);
    desc = readUTF8(u + 4, c);
    signature = null;
    anns = 0;
    ianns = 0;
    int dann = 0;
    int mpanns = 0;
    int impanns = 0;
    cattr = null;
    v = 0;
    w = 0;

    // looks for Code and Exceptions attributes
    j = readUnsignedShort(u + 6);
    u += 8;
    for(; j > 0; --j)
    {
        attrName = readUTF8(u, c);
        int attrSize = readInt(u + 2);
        u += 6;
        // tests are sorted in decreasing frequency order
        // (based on frequencies observed on typical classes)
        if(attrName.equals("Code"))
        {
            if(!skipCode)
            {
                v = u;
            }
        }
        else if(attrName.equals("Exceptions"))
        {
            w = u;
        }
        else if(attrName.equals("Signature"))
        {

```

```

        signature = readUTF8(u, c);
    }
    else if(attrName.equals("Deprecated"))
    {
        access |= Opcodes.ACC_DEPRECATED;
    }
    else if(attrName.equals("RuntimeVisibleAnnotations"))
    {
        anns = u;
    }
    else if(attrName.equals("AnnotationDefault"))
    {
        dann = u;
    }
    else if(attrName.equals("Synthetic"))
    {
        access |= Opcodes.ACC_SYNTHETIC;
    }
    else if(attrName.equals("RuntimeInvisibleAnnotations"))
    {
        ianns = u;
    }
    else if(attrName.equals(
            "RuntimeVisibleParameterAnnotations"))
    {
        mpanns = u;
    }
    else if(attrName.equals(
            "RuntimeInvisibleParameterAnnotations"))
    {
        impanns = u;
    }
    else
    {
        attr = readAttribute(attrs,
                            attrName,
                            u,
                            attrSize,
                            c,
                            -1,
                            null);
        if(attr != null)
        {
            attr.next = cattrs;
            cattrs = attr;
        }
    }
    u += attrSize;
}
// reads declared exceptions

```

```

String[] exceptions;
if(w == 0)
{
    exceptions = null;
}
else
{
    exceptions = new String[readUnsignedShort(w)];
    w += 2;
    for(j = 0; j < exceptions.length; ++j)
    {
        exceptions[j] = readClass(w, c);
        w += 2;
    }
}

// visits the method's code, if any
MethodVisitor mv = classVisitor.visitMethod(access,
                                             name,
                                             desc,
                                             signature,
                                             exceptions);

if(mv != null)
{
/*
 * if the returned MethodVisitor is in fact a MethodWriter, it
 * means there is no method adapter between the reader and the
 * writer. If, in addition, the writer's constant pool was
 * copied from this reader (mw.cw.cr == this), and the
 * signature and exceptions of the method have not been
 * changed, then it is possible to skip all visit events and
 * just copy the original code of the method to the writer
 * (the access, name and descriptor can have been changed,
 * this is not important since they are not copied as is from
 * the reader).
*/
    if(mv instanceof MethodWriter)
    {
        MethodWriter mw = (MethodWriter) mv;
        if(mw.cw.cr == this)
        {
            if(signature == mw.signature)
            {
                boolean sameExceptions = false;
                if(exceptions == null)
                {
                    sameExceptions = mw.exceptionCount == 0;
                }
            }
        }
    }
}

```

```

    {
    if(exceptions.length == mw.exceptionCount)
    {
        sameExceptions = true;
        for(j = exceptions.length - 1; j >= 0; --j)
        {
            w -= 2;
            if(mw.exceptions[j] != readUnsignedShort(w))
            {
                sameExceptions = false;
                break;
            }
        }
    }
    if(sameExceptions)
    {
        /*
         * we do not copy directly the code into
         * MethodWriter to save a byte array copy
         * operation. The real copy will be done in
         * ClassWriter.toByteArray().
         */
        mw.classReaderOffset = u0;
        mw.classReaderLength = u - u0;
        continue;
    }
}
}

if(dann != 0)
{
    AnnotationVisitor dv = mv.visitAnnotationDefault();
    readAnnotationValue(dann, c, null, dv);
    if(dv != null)
    {
        dv.visitEnd();
    }
}
for(j = 1; j >= 0; --j)
{
    w = j == 0 ? ianns : anns;
    if(w != 0)
    {
        k = readUnsignedShort(w);
        w += 2;
        for(; k > 0; --k)
        {

```

```

        w = readAnnotationValues(w + 2, c, true,
            mv.visitAnnotation(readUTF8(w, c), j != 0));
    }
}
if(mpans != 0)
{
    readParameterAnnotations(mpans, c, true, mv);
}
if(impanns != 0)
{
    readParameterAnnotations(impanns, c, false, mv);
}
while(cattr != null)
{
    attr = cattr.next;
    cattr.next = null;
    mv.visitAttribute(cattr);
    cattr = attr;
}
}

if(mv != null && v != 0)
{
    int maxStack = readUnsignedShort(v);
    int maxLocals = readUnsignedShort(v + 2);
    int codeLength = readInt(v + 4);
    v += 8;

    int codeStart = v;
    int codeEnd = v + codeLength;

    mv.visitCode();

    // 1st phase: finds the labels
    int label;
    Label[] labels = new Label[codeLength + 1];
    while(v < codeEnd)
    {
        int opcode = b[v] & 0xFF;
        switch(ClassWriter.TYPE[opcode])
        {
            case ClassWriter.NOARG_INSN:
            case ClassWriter.IMPLVAR_INSN:
                v += 1;
                break;
            case ClassWriter.LABEL_INSN:
                label = v - codeStart + readShort(v + 1);
                if(labels[label] == null)
                {

```

```

        labels[label] = new Label();
    }
    v += 3;
    break;
case ClassWriter.LABELW_INSN:
    label = v - codeStart + readInt(v + 1);
    if(labels[label] == null)
    {
        labels[label] = new Label();
    }
    v += 5;
    break;
case ClassWriter.WIDE_INSN:
    opcode = b[v + 1] & 0xFF;
    if(opcode == Opcodes.IINC)
    {
        v += 6;
    }
    else
    {
        v += 4;
    }
    break;
case ClassWriter.TABL_INSN:
    // skips 0 to 3 padding bytes
    w = v - codeStart;
    v = v + 4 - (w & 3);
    // reads instruction
    label = w + readInt(v);
    if(labels[label] == null)
    {
        labels[label] = new Label();
    }
    j = readInt(v + 8) - readInt(v + 4) + 1;
    v += 12;
    for(; j > 0; --j)
    {
        label = w + readInt(v);
        v += 4;
        if(labels[label] == null)
        {
            labels[label] = new Label();
        }
    }
    break;
case ClassWriter.LOOK_INSN:
    // skips 0 to 3 padding bytes
    w = v - codeStart;
    v = v + 4 - (w & 3);
    // reads instruction

```

```

        label = w + readInt(v);
        if(labels[label] == null)
        {
            labels[label] = new Label();
        }
        j = readInt(v + 4);
        v += 8;
        for(; j > 0; --j)
        {
            label = w + readInt(v + 4);
            v += 8;
            if(labels[label] == null)
            {
                labels[label] = new Label();
            }
        }
        break;
    case ClassWriter.VAR_INSN:
    case ClassWriter.SBYTE_INSN:
    case ClassWriter.LDC_INSN:
        v += 2;
        break;
    case ClassWriter.SHORT_INSN:
    case ClassWriter.LDCW_INSN:
    case ClassWriter.FIELDORMETH_INSN:
    case ClassWriter.TYPE_INSN:
    case ClassWriter.IINC_INSN:
        v += 3;
        break;
    case ClassWriter.ITFMETH_INSN:
        v += 5;
        break;
        // case MANA_INSN:
    default:
        v += 4;
        break;
    }
}
// parses the try catch entries
j = readUnsignedShort(v);
v += 2;
for(; j > 0; --j)
{
    label = readUnsignedShort(v);
    Label start = labels[label];
    if(start == null)
    {
        labels[label] = start = new Label();
    }
    label = readUnsignedShort(v + 2);
}

```

```

Label end = labels[label];
if(end == null)
{
    labels[label] = end = new Label();
}
label = readUnsignedShort(v + 4);
Label handler = labels[label];
if(handler == null)
{
    labels[label] = handler = new Label();
}
int type = readUnsignedShort(v + 6);
if(type == 0)
{
    mv.visitTryCatchBlock(start, end, handler, null);
}
else
{
    mv.visitTryCatchBlock(start,
                         end,
                         handler,
                         readUTF8(items[type], c));
}
v += 8;
}

// parses the local variable, line number tables, and code
// attributes
int varTable = 0;
int varTypeTable = 0;
int stackMap = 0;
int frameCount = 0;
int frameMode = 0;
int frameOffset = 0;
int frameLocalCount = 0;
int frameLocalDiff = 0;
int frameStackCount = 0;
Object[] frameLocal = null;
Object[] frameStack = null;
boolean zip = true;
cattrs = null;
j = readUnsignedShort(v);
v += 2;
for(; j > 0; --j)
{
    attrName = readUTF8(v, c);
    if(attrName.equals("LocalVariableTable"))
    {
        if(!skipDebug)
        {
            varTable = v + 6;
        }
    }
}

```

```

        k = readUnsignedShort(v + 6);
        w = v + 8;
        for(; k > 0; --k)
        {
            label = readUnsignedShort(w);
            if(labels[label] == null)
            {
                labels[label] = new Label(true);
            }
            label += readUnsignedShort(w + 2);
            if(labels[label] == null)
            {
                labels[label] = new Label(true);
            }
            w += 10;
        }
    }
    else if(attrName.equals("LocalVariableTypeTable"))
    {
        varTypeTable = v + 6;
    }
    else if(attrName.equals("LineNumberTable"))
    {
        if(!skipDebug)
        {
            k = readUnsignedShort(v + 6);
            w = v + 8;
            for(; k > 0; --k)
            {
                label = readUnsignedShort(w);
                if(labels[label] == null)
                {
                    labels[label] = new Label(true);
                }
                labels[label].line=readUnsignedShort(w + 2);
                w += 4;
            }
        }
    }
    else if(attrName.equals("StackMapTable"))
    {
        if((flags & SKIP_FRAMES) == 0)
        {
            stackMap = v + 8;
            frameCount = readUnsignedShort(v + 6);
        }
    }
/*
 * here we do not extract the labels corresponding to
 * the attribute content. This would require a full

```

```

* parsing of the attribute, which would need to be
* repeated in the second phase (see below). Instead the
* content of the attribute is read one frame at a time
* (i.e. after a frame has been visited, the next frame
* is read), and the labels it contains are also
* extracted one frame at a time. Thanks to the ordering
* of frames, having only a "one frame lookahead" is not
* a problem, i.e. it is not possible to see an offset
* smaller than the offset of the current insn and for
* which no Label exist.
*/
// TODO true for frame offsets,
// but for UNINITIALIZED type offsets?
}
else if(attrName.equals("StackMap"))
{
    if((flags & SKIP_FRAMES) == 0)
    {
        stackMap = v + 8;
        frameCount = readUnsignedShort(v + 6);
        zip = false;
    }
/*
 * IMPORTANT! here we assume that the frames are
 * ordered, as in the StackMapTable attribute,
 * although this is not guaranteed by the
 * attribute format.
 */
}
else
{
    for(k = 0; k < attrs.length; ++k)
    {
        if(attrs[k].type.equals(attrName))
        {
            attr = attrs[k].read(this,
                                v + 6,
                                readInt(v + 2),
                                c,
                                codeStart - 8,
                                labels);

            if(attr != null)
            {
                attr.next = cattrs;
                cattrs = attr;
            }
        }
    }
}
v += 6 + readInt(v + 2);

```

```

    }

    // 2nd phase: visits each instruction
    if(stackMap != 0)
    {
        // creates the very first (implicit) frame from the
        // method descriptor
        frameLocal = new Object[maxLocals];
        frameStack = new Object[maxStack];
        if(unzip)
        {
            int local = 0;
            if((access & Opcodes.ACC_STATIC) == 0)
            {
                if(name.equals("<init>"))
                {
                    frameLocal[local++] = Opcodes.UNINITIALIZED_THIS;
                }
                else
                {
                    frameLocal[local++] = readClass(header + 2, c);
                }
            }
            j = 1;
            loop:
            while(true)
            {
                k = j;
                switch(desc.charAt(j++))
                {
                    case'Z':
                    case'C':
                    case'B':
                    case'S':
                    case'I':
                        frameLocal[local++] = Opcodes.INTEGER;
                        break;
                    case'F':
                        frameLocal[local++] = Opcodes.FLOAT;
                        break;
                    case'J':
                        frameLocal[local++] = Opcodes.LONG;
                        break;
                    case'D':
                        frameLocal[local++] = Opcodes.DOUBLE;
                        break;
                    case '[':
                        while(desc.charAt(j) == '[')
                        {
                            ++j;

```

```

        }
        if(desc.charAt(j) == 'L')
        {
            ++j;
            while(desc.charAt(j) != ';')
            {
                ++j;
            }
        }
        frameLocal[local++] =
            desc.substring(k, ++j);
        break;
    case'L':
        while(desc.charAt(j) != ';')
        {
            ++j;
        }
        frameLocal[local++] =
            desc.substring(k + 1, j++);
        break;
    default:
        break loop;
    }
}
frameLocalCount = local;
}
*/
/* for the first explicit frame the offset is not
 * offset_delta + 1 but only offset_delta; setting the
 * implicit frame offset to -1 allow the use of the
 * "offset_delta + 1" rule in all cases
 */
frameOffset = -1;
}
v = codeStart;
Label l;
while(v < codeEnd)
{
    w = v - codeStart;

    l = labels[w];
    if(l != null)
    {
        mv.visitLabel(l);
        if(!skipDebug && l.line > 0)
        {
            mv.visitLineNumber(l.line, l);
        }
    }
}

```

```

        while(frameLocal != null
              && (frameOffset == w || frameOffset == -1))
        {
            // if there is a frame for this offset,
            // makes the visitor visit it,
            // and reads the next frame if there is one.
            if(!zip || unzip)
            {
                mv.visitFrame(Opcodes.F_NEW,
                             frameLocalCount,
                             frameLocal,
                             frameStackCount,
                             frameStack);
            }
            else if(frameOffset != -1)
            {
                mv.visitFrame(frameMode,
                             frameLocalDiff,
                             frameLocal,
                             frameStackCount,
                             frameStack);
            }
        }

        if(frameCount > 0)
        {
            int tag, delta, n;
            if(zip)
            {
                tag = b[stackMap++] & 0xFF;
            }
            else
            {
                tag = MethodWriter.FULL_FRAME;
                frameOffset = -1;
            }
            frameLocalDiff = 0;
            if(tag <
               MethodWriter.SAME_LOCALS_1_STACK_ITEM_FRAME)
            {
                delta = tag;
                frameMode = Opcodes.F_SAME;
                frameStackCount = 0;
            }
            else if(tag < MethodWriter.RESERVED)
            {
                delta = tag
                - MethodWriter.SAME_LOCALS_1_STACK_ITEM_FRAME;
                stackMap = readFrameType(frameStack,
                                         0,
                                         stackMap,
                                         0);
            }
        }
    }
}

```

```

        c,
        labels);
frameMode = Opcodes.F_SAME1;
frameStackCount = 1;
}
else
{
    delta = readUnsignedShort(stackMap);
    stackMap += 2;
    if(tag ==
MethodWriter.SAME_LOCALS_1_STACK_ITEM_FRAME_EXTENDED)
{
    stackMap = readFrameType(frameStack,
                                0,
                                stackMap,
                                c,
                                labels);
    frameMode = Opcodes.F_SAME1;
    frameStackCount = 1;
}
else if(tag >= MethodWriter.CHOP_FRAME &&
        tag < MethodWriter.SAME_FRAME_EXTENDED)
{
    frameMode = Opcodes.F_CHOP;
    frameLocalDiff =
        MethodWriter.SAME_FRAME_EXTENDED
        - tag;
    frameLocalCount -= frameLocalDiff;
    frameStackCount = 0;
}
else if(tag ==
        MethodWriter.SAME_FRAME_EXTENDED)
{
    frameMode = Opcodes.F_SAME;
    frameStackCount = 0;
}
else if(tag < MethodWriter.FULL_FRAME)
{
    j = unzip ? frameLocalCount : 0;
    for(k = tag
        - MethodWriter.SAME_FRAME_EXTENDED;
        k > 0; k--)
    {
        stackMap = readFrameType(frameLocal,
                                j++,
                                stackMap,
                                c,
                                labels);
    }
    frameMode = Opcodes.F_APPEND;
}

```

```

        frameLocalDiff = tag
            - MethodWriter.SAME_FRAME_EXTENDED;
        frameLocalCount += frameLocalDiff;
        frameStackCount = 0;
    }
}
else
{ // if (tag == FULL_FRAME) {
    frameMode = Opcodes.F_FULL;
    n = frameLocalDiff
        = frameLocalCount
        = readUnsignedShort(stackMap);
    stackMap += 2;
    for(j = 0; n > 0; n--)
    {
        stackMap = readFrameType(frameLocal,
                                  j++,
                                  stackMap,
                                  c,
                                  labels);
    }
    n = frameStackCount
        = readUnsignedShort(stackMap);
    stackMap += 2;
    for(j = 0; n > 0; n--)
    {
        stackMap = readFrameType(frameStack,
                                  j++,
                                  stackMap,
                                  c,
                                  labels);
    }
}
frameOffset += delta + 1;
if(labels[frameOffset] == null)
{
    labels[frameOffset] = new Label();
}

--frameCount;
}
else
{
    frameLocal = null;
}
}

int opcode = b[v] & 0xFF;
switch(ClassWriter.TYPE[opcode])
{

```

```

case ClassWriter.NOARG_INSN:
    mv.visitInsn(opcode);
    v += 1;
    break;
case ClassWriter.IMPLVAR_INSN:
    if(opcode > Opcodes.ISTORE)
    {
        opcode -= 59; // ISTORE_0
        mv.visitVarInsn(
            Opcodes.ISTORE + (opcode >> 2),
            opcode & 0x3);
    }
    else
    {
        opcode -= 26; // ILOAD_0
        mv.visitVarInsn(
            Opcodes.ILOAD + (opcode >> 2),
            opcode & 0x3);
    }
    v += 1;
    break;
case ClassWriter.LABEL_INSN:
    mv.visitJumpInsn(opcode,
                     labels[w + readShort(v + 1)]);
    v += 3;
    break;
case ClassWriter.LABELW_INSN:
    mv.visitJumpInsn(opcode - 33,
                     labels[w + readInt(v + 1)]);
    v += 5;
    break;
case ClassWriter.WIDE_INSN:
    opcode = b[v + 1] & 0xFF;
    if(opcode == Opcodes.IINC)
    {
        mv.visitIincInsn(readUnsignedShort(v + 2),
                         readShort(v + 4));
        v += 6;
    }
    else
    {
        mv.visitVarInsn(opcode,
                        readUnsignedShort(v + 2));
        v += 4;
    }
    break;
case ClassWriter.TABL_INSN:
    // skips 0 to 3 padding bytes
    v = v + 4 - (w & 3);
    // reads instruction

```

```

        label = w + readInt(v);
        int min = readInt(v + 4);
        int max = readInt(v + 8);
        v += 12;
        Label[] table = new Label[max - min + 1];
        for(j = 0; j < table.length; ++j)
        {
            table[j] = labels[w + readInt(v)];
            v += 4;
        }
        mv.visitTableSwitchInsn(min,
                               max,
                               labels[label],
                               table);
        break;
    case ClassWriter.LOOK_INSN:
        // skips 0 to 3 padding bytes
        v = v + 4 - (w & 3);
        // reads instruction
        label = w + readInt(v);
        j = readInt(v + 4);
        v += 8;
        int[] keys = new int[j];
        Label[] values = new Label[j];
        for(j = 0; j < keys.length; ++j)
        {
            keys[j] = readInt(v);
            values[j] = labels[w + readInt(v + 4)];
            v += 8;
        }
        mv.visitLookupSwitchInsn(labels[label],
                               keys,
                               values);
        break;
    case ClassWriter.VAR_INSN:
        mv.visitVarInsn(opcode, b[v + 1] & 0xFF);
        v += 2;
        break;
    case ClassWriter.SBYTE_INSN:
        mv.visitIntInsn(opcode, b[v + 1]);
        v += 2;
        break;
    case ClassWriter.SHORT_INSN:
        mv.visitIntInsn(opcode, readShort(v + 1));
        v += 3;
        break;
    case ClassWriter.LDC_INSN:
        mv.visitLdcInsn(readConst(b[v + 1] & 0xFF, c));
        v += 2;
        break;
    
```

```

case ClassWriter.LDCW_INSN:
    mv.visitLdcInsn(
        readConst(readUnsignedShort(v + 1), c));
    v += 3;
    break;
case ClassWriter.FIELDORMETH_INSN:
case ClassWriter.ITFMETH_INSN:
    int cpIndex = items[readUnsignedShort(v + 1)];
    String iowner = readClass(cpIndex, c);
    cpIndex = items[readUnsignedShort(cpIndex + 2)];
    String iname = readUTF8(cpIndex, c);
    String idesc = readUTF8(cpIndex + 2, c);
    if(opcode < Opcodes.INVOKEVIRTUAL)
    {
        mv.visitFieldInsn(opcode, iowner, iname, idesc);
    }
    else
    {
        mv.visitMethodInsn(opcode, iowner, iname, idesc);
    }
    if(opcode == Opcodes.INVOKEINTERFACE)
    {
        v += 5;
    }
    else
    {
        v += 3;
    }
    break;
case ClassWriter.TYPE_INSN:
    mv.visitTypeInsn(opcode, readClass(v + 1, c));
    v += 3;
    break;
case ClassWriter.IINC_INSN:
    mv.visitIincInsn(b[v + 1] & 0xFF, b[v + 2]);
    v += 3;
    break;
    // case MANA_INSN:
default:
    mv.visitMultiANewArrayInsn(readClass(v + 1, c),
                                b[v + 3] & 0xFF);
    v += 4;
    break;
}
}
l = labels[codeEnd - codeStart];
if(l != null)
{
    mv.visitLabel(l);
}

```

```

// visits the local variable tables
if(!skipDebug && varTable != 0)
{
    int[] typeTable = null;
    if(varTypeTable != 0)
    {
        k = readUnsignedShort(varTypeTable) * 3;
        w = varTypeTable + 2;
        typeTable = new int[k];
        while(k > 0)
        {
            typeTable[--k] = w + 6; // signature
            typeTable[--k] = readUnsignedShort(w + 8); //index
            typeTable[--k] = readUnsignedShort(w); // start
            w += 10;
        }
    }
    k = readUnsignedShort(varTable);
    w = varTable + 2;
    for(; k > 0; --k)
    {
        int start = readUnsignedShort(w);
        int length = readUnsignedShort(w + 2);
        int index = readUnsignedShort(w + 8);
        String vsignature = null;
        if(typeTable != null)
        {
            for(int a = 0; a < typeTable.length; a += 3)
            {
                if(typeTable[a] == start
                   && typeTable[a + 1] == index)
                {
                    vsignature=readUTF8(typeTable[a + 2], c);
                    break;
                }
            }
        }
        mv.visitLocalVariable(readUTF8(w + 4, c),
                             readUTF8(w + 6, c),
                             vsignature,
                             labels[start],
                             labels[start + length],
                             index);
        w += 10;
    }
}
// visits the other attributes
while(cattrs != null)
{
    attr = cattrs.next;
}

```

```

        cattrs.next = null;
        mv.visitAttribute(cattrs);
        cattrs = attr;
    }
    // visits the max stack and max locals values
    mv.visitMaxs(maxStack, maxLocals);
}

if(mv != null)
{
    mv.visitEnd();
}
}

// visits the end of the class
classVisitor.visitEnd();
}

<**
 * Reads parameter annotations and makes the given visitor visit them.
 *
 * @param v      start offset in {@link #b b} of the annotations to
 *               be read.
 * @param buf    buffer to be used to call {@link #readUTF8 readUTF8},
 *               {@link #readClass(int,char[]) readClass} or
 *               {@link #readConst readConst}.
 * @param visible <tt>true</tt> if the annotations to be read are visible
 *               at runtime.
 * @param mv     the visitor that must visit the annotations.
 */
private void readParameterAnnotations(
    int v,
    final char[] buf,
    final boolean visible,
    final MethodVisitor mv){
    int n = b[v++] & 0xFF;
    for(int i = 0; i < n; ++i)
    {
        int j = readUnsignedShort(v);
        v += 2;
        for(; j > 0; --j)
        {
            v = readAnnotationValues(v + 2, buf, true,
                mv.visitParameterAnnotation(i,readUTF8(v, buf),visible));
        }
    }
}

<**
 * Reads the values of an annotation and makes the given visitor

```

```

* visit them.
*
* @param v      the start offset in {@link #b b} of the values to be
*               read (including the unsigned short that gives the
*               number of values).
* @param buf    buffer to be used to call {@link #readUTF8 readUTF8},
*               {@link #readClass(int,char[]) readClass} or
*               {@link #readConst readConst}.
* @param named if the annotation values are named or not.
* @param av     the visitor that must visit the values.
* @return the end offset of the annotation values.
*/
private int readAnnotationValues(
    int v,
    final char[] buf,
    final boolean named,
    final AnnotationVisitor av){
    int i = readUnsignedShort(v);
    v += 2;
    if(named)
    {
        for(; i > 0; --i)
        {
            v = readAnnotationValue(v + 2, buf, readUTF8(v, buf), av);
        }
    }
    else
    {
        for(; i > 0; --i)
        {
            v = readAnnotationValue(v, buf, null, av);
        }
    }
    if(av != null)
    {
        av.visitEnd();
    }
    return v;
}

/**
 * Reads a value of an annotation and makes the given visitor visit it.
 *
* @param v      the start offset in {@link #b b} of the value to be
*               read (<i>not including the value name constant pool
*               index</i>).
* @param buf    buffer to be used to call {@link #readUTF8 readUTF8},
*               {@link #readClass(int,char[]) readClass} or
*               {@link #readConst readConst}.
* @param name   the name of the value to be read.

```

```

* @param av  the visitor that must visit the value.
* @return the end offset of the annotation value.
*/
private int readAnnotationValue(
    int v,
    final char[] buf,
    final String name,
    final AnnotationVisitor av){
    int i;
    if(av == null)
    {
        switch(b[v] & 0xFF)
        {
            case'e': // enum_const_value
                return v + 5;
            case'@': // annotation_value
                return readAnnotationValues(v + 3, buf, true, null);
            case '[': // array_value
                return readAnnotationValues(v + 1, buf, false, null);
            default:
                return v + 3;
        }
    }
    switch(b[v++] & 0xFF)
    {
        case'I': // pointer to CONSTANT_Integer
        case'J': // pointer to CONSTANT_Long
        case'F': // pointer to CONSTANT_Float
        case'D': // pointer to CONSTANT_Double
            av.visit(name, readConst(readUnsignedShort(v), buf));
            v += 2;
            break;
        case'B': // pointer to CONSTANT_Byte
            av.visit(name,
                    new Byte((byte)
                            readInt(items[readUnsignedShort(v)])));
            v += 2;
            break;
        case'Z': // pointer to CONSTANT_Boolean
            av.visit(name, readInt(items[readUnsignedShort(v)]) == 0
                    ? Boolean.FALSE
                    : Boolean.TRUE);
            v += 2;
            break;
        case'S': // pointer to CONSTANT_Short
            av.visit(name,
                    new Short((short)
                            readInt(items[readUnsignedShort(v)])));
            v += 2;
            break;
    }
}

```

```

case'C': // pointer to CONSTANT_Char
    av.visit(name,
              new Character((char)
                  readInt(items[readUnsignedShort(v)])));
    v += 2;
    break;
case's': // pointer to CONSTANT_Utf8
    av.visit(name, readUTF8(v, buf));
    v += 2;
    break;
case'e': // enum_const_value
    av.visitEnum(name, readUTF8(v, buf), readUTF8(v + 2, buf));
    v += 4;
    break;
case'c': // class_info
    av.visit(name, Type.getType(readUTF8(v, buf)));
    v += 2;
    break;
case'@': // annotation_value
    v = readAnnotationValues(v + 2, buf, true,
                             av.visitAnnotation(name, readUTF8(v, buf)));
    break;
case '[': // array_value
    int size = readUnsignedShort(v);
    v += 2;
    if(size == 0)
    {
        return readAnnotationValues(v - 2,
                                     buf,
                                     false,
                                     av.visitArray(name));
    }
    switch(this.b[v++] & 0xFF)
    {
        case'B':
            byte[] bv = new byte[size];
            for(i = 0; i < size; i++)
            {
                bv[i] =
                    (byte) readInt(items[readUnsignedShort(v)]);
                v += 3;
            }
            av.visit(name, bv);
            --v;
            break;
        case'Z':
            boolean[] zv = new boolean[size];
            for(i = 0; i < size; i++)
            {
                zv[i] =

```

```

        readInt(items[readUnsignedShort(v)]) != 0;
        v += 3;
    }
    av.visit(name, zv);
    --v;
    break;
case'S':
    short[] sv = new short[size];
    for(i = 0; i < size; i++)
    {
        sv[i] =
            (short) readInt(items[readUnsignedShort(v)]);
        v += 3;
    }
    av.visit(name, sv);
    --v;
    break;
case'C':
    char[] cv = new char[size];
    for(i = 0; i < size; i++)
    {
        cv[i] =
            (char) readInt(items[readUnsignedShort(v)]);
        v += 3;
    }
    av.visit(name, cv);
    --v;
    break;
case'I':
    int[] iv = new int[size];
    for(i = 0; i < size; i++)
    {
        iv[i] = readInt(items[readUnsignedShort(v)]);
        v += 3;
    }
    av.visit(name, iv);
    --v;
    break;
case'J':
    long[] lv = new long[size];
    for(i = 0; i < size; i++)
    {
        lv[i] = readLong(items[readUnsignedShort(v)]);
        v += 3;
    }
    av.visit(name, lv);
    --v;
    break;
case'F':
    float[] fv = new float[size];

```

```

        for(i = 0; i < size; i++)
        {
            fv[i] =
                Float.intBitsToFloat(
                    readInt(items[readUnsignedShort(v)]));
            v += 3;
        }
        av.visit(name, fv);
        --v;
        break;
    case'D':
        double[] dv = new double[size];
        for(i = 0; i < size; i++)
        {
            dv[i] =
                Double.longBitsToDouble(
                    readLong(items[readUnsignedShort(v)]));
            v += 3;
        }
        av.visit(name, dv);
        --v;
        break;
    default:
        v = readAnnotationValues(v - 3,
                                buf,
                                false,
                                av.visitArray(name));
}
return v;
}

private int readFrameType(
    final Object[] frame,
    final int index,
    int v,
    final char[] buf,
    final Label[] labels){
    int type = b[v++] & 0xFF;
    switch(type)
    {
        case 0:
            frame[index] = Opcodes.TOP;
            break;
        case 1:
            frame[index] = Opcodes.INTEGER;
            break;
        case 2:
            frame[index] = Opcodes.FLOAT;
            break;
    }
}

```

```

        case 3:
            frame[index] = Opcodes.DOUBLE;
            break;
        case 4:
            frame[index] = Opcodes.LONG;
            break;
        case 5:
            frame[index] = Opcodes.NULL;
            break;
        case 6:
            frame[index] = Opcodes.UNINITIALIZED_THIS;
            break;
        case 7: // Object
            frame[index] = readClass(v, buf);
            v += 2;
            break;
        default: // Uninitialized
            int offset = readUnsignedShort(v);
            if(labels[offset] == null)
            {
                labels[offset] = new Label();
            }
            frame[index] = labels[offset];
            v += 2;
        }
    return v;
}

/**
 * Reads an attribute in {@link #b b}.
 *
 * @param attrs prototypes of the attributes that must be parsed during
 *               the visit of the class. Any attribute whose type is not
 *               equal to the type of one the prototypes is ignored
 *               (i.e. an empty {@link Attribute} instance is returned).
 * @param type   the type of the attribute.
 * @param off    index of the first byte of the attribute's content in
 *               {@link #b b}. The 6 attribute header bytes, containing
 *               the type and the length of the attribute, are not taken
 *               into account here (they have already been read).
 * @param len    the length of the attribute's content.
 * @param buf    buffer to be used to call {@link #readUTF8 readUTF8},
 *               {@link #readClass(int,char[]) readClass} or
 *               {@link #readConst readConst}.
 * @param codeOff index of the first byte of code's attribute content in
 *               {@link #b b}, or -1 if the attribute to be read is not
 *               a code attribute. The 6 attribute header bytes,
 *               containing the type and the length of the attribute,
 *               are not taken into account here.
 * @param labels the labels of the method's code, or <tt>null</tt> if

```

```

*           the attribute to be read is not a code attribute.
* @return the attribute that has been read, or <tt>null</tt> to skip
*         this attribute.
*/
private Attribute readAttribute(
    final Attribute[] attrs,
    final String type,
    final int off,
    final int len,
    final char[] buf,
    final int codeOff,
    final Label[] labels){
    for(int i = 0; i < attrs.length; ++i)
    {
        if(attrs[i].type.equals(type))
        {
            return attrs[i].read(this, off, len, buf, codeOff, labels);
        }
    }
    return new Attribute(type).read(this, off, len, null, -1, null);
}

// -----
// Utility methods: low level parsing
// -----


/**
 * Returns the start index of the constant pool item in {@link #b b},
 * plus one. <i>This method is intended for {@link Attribute} sub
 * classes, and is normally not needed by class generators or
 * adapters.</i>
 *
 * @param item the index a constant pool item.
 * @return the start index of the constant pool item in {@link #b b},
 *         plus one.
 */
public int getItem(final int item){
    return items[item];
}

/**
 * Reads a byte value in {@link #b b}. <i>This method is intended for
 * {@link Attribute} sub classes, and is normally not needed by class
 * generators or adapters.</i>
 *
 * @param index the start index of the value to be read in {@link #b b}.
 * @return the read value.
 */
public int readByte(final int index){
    return b[index] & 0xFF;
}

```

```
}

/***
 * Reads an unsigned short value in {@link #b b}. <i>This method is
 * intended for {@link Attribute} sub classes, and is normally not
 * needed by class generators or adapters.</i>
 *
 * @param index the start index of the value to be read in {@link #b b}.
 * @return the read value.
 */
public int readUnsignedShort(final int index){
    byte[] b = this.b;
    return ((b[index] & 0xFF) << 8) | (b[index + 1] & 0xFF);
}

/***
 * Reads a signed short value in {@link #b b}. <i>This method is intended
 * for {@link Attribute} sub classes, and is normally not needed by class
 * generators or adapters.</i>
 *
 * @param index the start index of the value to be read in {@link #b b}.
 * @return the read value.
 */
public short readShort(final int index){
    byte[] b = this.b;
    return (short) (((b[index] & 0xFF) << 8) | (b[index + 1] & 0xFF));
}

/***
 * Reads a signed int value in {@link #b b}. <i>This method is intended
 * for {@link Attribute} sub classes, and is normally not needed by
 * class generators or adapters.</i>
 *
 * @param index the start index of the value to be read in {@link #b b}.
 * @return the read value.
 */
public int readInt(final int index){
    byte[] b = this.b;
    return ((b[index] & 0xFF) << 24) | ((b[index + 1] & 0xFF) << 16)
        | ((b[index + 2] & 0xFF) << 8) | (b[index + 3] & 0xFF);
}

/***
 * Reads a signed long value in {@link #b b}. <i>This method is intended
 * for {@link Attribute} sub classes, and is normally not needed by class
 * generators or adapters.</i>
 *
 * @param index the start index of the value to be read in {@link #b b}.
 * @return the read value.
 */
```

```

public long readLong(final int index){
    long l1 = readInt(index);
    long l0 = readInt(index + 4) & 0xFFFFFFFFL;
    return (l1 << 32) | l0;
}

/**
 * Reads an UTF8 string constant pool item in {@link #b b}. <i>This
 * method is intended for {@link Attribute} sub classes, and is normally
 * not needed by class generators or adapters.</i>
 *
 * @param index the start index of an unsigned short value in
 *              {@link #b b}, whose value is the index of an UTF8
 *              constant pool item.
 * @param buf   buffer to be used to read the item. This buffer must be
 *              sufficiently large. It is not automatically resized.
 * @return the String corresponding to the specified UTF8 item.
 */
public String readUTF8(int index, final char[] buf){
    int item = readUnsignedShort(index);
    String s = strings[item];
    if(s != null)
    {
        return s;
    }
    index = items[item];
    return strings[item] =
        readUTF(index + 2, readUnsignedShort(index), buf);
}

/**
 * Reads UTF8 string in {@link #b b}.
 *
 * @param index start offset of the UTF8 string to be read.
 * @param utfLen length of the UTF8 string to be read.
 * @param buf   buffer to be used to read the string. This buffer must
 *              be sufficiently large. It is not automatically resized.
 * @return the String corresponding to the specified UTF8 string.
 */
private String readUTF(int index, final int utfLen, final char[] buf){
    int endIndex = index + utfLen;
    byte[] b = this.b;
    int strLen = 0;
    int c, d, e;
    while(index < endIndex)
    {
        c = b[index++] & 0xFF;
        switch(c >> 4)
        {
            case 0:

```

```

        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6:
        case 7:
            // 0xxxxxxx
            buf[strLen++] = (char) c;
            break;
        case 12:
        case 13:
            // 110x xxxx 10xx xxxx
            d = b[index++];
            buf[strLen++] = (char) (((c & 0x1F) << 6) | (d & 0x3F));
            break;
        default:
            // 1110 xxxx 10xx xxxx 10xx xxxx
            d = b[index++];
            e = b[index++];
            buf[strLen++] =
                (char) (((c & 0x0F) << 12)
                    | ((d & 0x3F) << 6) | (e & 0x3F));
            break;
        }
    }
    return new String(buf, 0, strLen);
}

/**
 * Reads a class constant pool item in {@link #b b}. <i>This method
 * is intended for {@link Attribute} sub classes, and is normally not
 * needed by class generators or adapters.</i>
 *
 * @param index the start index of an unsigned short value in
 *              {@link #b b}, whose value is the index of a class
 *              constant pool item.
 * @param buf   buffer to be used to read the item. This buffer must be
 *              sufficiently large. It is not automatically resized.
 * @return the String corresponding to the specified class item.
 */
public String readClass(final int index, final char[] buf){
    // computes the start index of the CONSTANT_Class item in b
    // and reads the CONSTANT_Utf8 item designated by
    // the first two bytes of this CONSTANT_Class item
    return readUTF8(items[readUnsignedShort(index)], buf);
}

/**
 * Reads a numeric or string constant pool item in {@link #b b}.

```

```

* <i>This method is intended for {@link Attribute} sub classes,
* and is normally not needed by class generators or adapters.</i>
*
* @param item the index of a constant pool item.
* @param buf buffer to be used to read the item. This buffer must be
*            sufficiently large. It is not automatically resized.
* @return the {@link Integer}, {@link Float}, {@link Long},
*         {@link Double}, {@link String} or {@link Type} corresponding
*         to the given constant pool item.
*/
public Object readConst(final int item, final char[] buf){
    int index = items[item];
    switch(b[index - 1])
    {
        case ClassWriter.INT:
            return new Integer(readInt(index));
        case ClassWriter.FLOAT:
            return new Float(Float.intBitsToFloat(readInt(index)));
        case ClassWriter.LONG:
            return new Long(readLong(index));
        case ClassWriter.DOUBLE:
            return new Double(Double.longBitsToDouble(readLong(index)));
        case ClassWriter.CLASS:
            String s = readUTF8(index, buf);
            return s.charAt(0) == '['
                ? Type.getType(s)
                : Type.getObjectType(s);
            // case ClassWriter.STR:
        default:
            return readUTF8(index, buf);
    }
}
}

```

7.7 ClassVisitor.java

— ClassVisitor.java —

```

\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * A visitor to visit a Java class. The methods of this interface
 * must be called in the following order: <tt>visit</tt>
 * [ <tt>visitSource</tt> ] [ <tt>visitOuterClass</tt> ]

```

```

* ( <tt>visitAnnotation</tt> | <tt>visitAttribute</tt> )*
* (<tt>visitInnerClass</tt> | <tt>visitField</tt> |
* <tt>visitMethod</tt> )* <tt>visitEnd</tt>.
*
* @author Eric Bruneton
*/
public interface ClassVisitor{

/**
 * Visits the header of the class.
 *
 * @param version      the class version.
 * @param access       the class's access flags (see {@link Opcodes}). This
 *                     parameter also indicates if the class is deprecated.
 * @param name         the internal name of the class (see
 *                     {@link Type#getInternalName() getInternalName}).
 * @param signature    the signature of this class. May be <tt>null</tt> if
 *                     the class is not a generic one, and does not extend
 *                     or implement generic classes or interfaces.
 * @param superName   the internal of name of the super class (see
 *                     {@link Type#getInternalName() getInternalName}).
 * @param interfaces  For interfaces, the super class is {@link Object}.
 *                     May be <tt>null</tt>, but only for the
 *                     {@link Object} class.
 * @param interfaces  the internal names of the class's interfaces (see
 *                     {@link Type#getInternalName() getInternalName}).
 *                     May be <tt>null</tt>.
 */
void visit(
    int version,
    int access,
    String name,
    String signature,
    String superName,
    String[] interfaces);

/**
 * Visits the source of the class.
 *
 * @param source      the name of the source file from which the class was
 *                   compiled. May be <tt>null</tt>.
 * @param debug       additional debug information to compute the
 *                   correspondance between source and compiled elements of
 *                   the class. May be <tt>null</tt>.
 */
void visitSource(String source, String debug);

/**
 * Visits the enclosing class of the class. This method must be called
 * only if the class has an enclosing class.

```

```

*
* @param owner internal name of the enclosing class of the class.
* @param name  the name of the method that contains the class, or
*               <tt>null</tt> if the class is not enclosed in a method
*               of its enclosing class.
* @param desc   the descriptor of the method that contains the class, or
*               <tt>null</tt> if the class is not enclosed in a method of
*               its enclosing class.
*/
void visitOuterClass(String owner, String name, String desc);

/**
 * Visits an annotation of the class.
 *
 * @param desc   the class descriptor of the annotation class.
 * @param visible <tt>true</tt> if the annotation is visible at runtime.
 * @return a visitor to visit the annotation values, or <tt>null</tt> if
 *         this visitor is not interested in visiting this annotation.
 */
AnnotationVisitor visitAnnotation(String desc, boolean visible);

/**
 * Visits a non standard attribute of the class.
 *
 * @param attr an attribute.
 */
void visitAttribute(Attribute attr);

/**
 * Visits information about an inner class. This inner class is not
 * necessarily a member of the class being visited.
 *
 * @param name      the internal name of an inner class (see
 *                  {@link Type#getInternalName() getInternalName}).
 * @param outerName the internal name of the class to which the inner
 *                  class belongs (see {@link Type#getInternalName()
 *                  getInternalName}). May be <tt>null</tt> for not
 *                  member classes.
 * @param innerName the (simple) name of the inner class inside its
 *                  enclosing class. May be <tt>null</tt> for anonymous
 *                  inner classes.
 * @param access    the access flags of the inner class as originally
 *                  declared in the enclosing class.
 */
void visitInnerClass(
    String name,
    String outerName,
    String innerName,
    int access);

```

```

/**
 * Visits a field of the class.
 *
 * @param access      the field's access flags (see {@link Opcodes}).
 *                    This parameter also indicates if the field is
 *                    synthetic and/or deprecated.
 * @param name        the field's name.
 * @param desc         the field's descriptor (see {@link Type Type}).
 * @param signature   the field's signature. May be <tt>null</tt> if the
 *                    field's type does not use generic types.
 * @param value        the field's initial value. This parameter, which may
 *                    be <tt>null</tt> if the field does not have an
 *                    initial value, must be an {@link Integer}, a
 *                    {@link Float}, a {@link Long}, a {@link Double} or
 *                    a {@link String} (for <tt>int</tt>, <tt>float</tt>,
 *                    <tt>long</tt> or <tt>String</tt> fields
 *                    respectively). <i>This parameter is only used for
 *                    static fields</i>. Its value is ignored for non
 *                    static fields, which must be initialized through
 *                    bytecode instructions in constructors or methods.
 * @return a visitor to visit field annotations and attributes, or
 *         <tt>null</tt> if this class visitor is not interested in
 *         visiting these annotations and attributes.
 */
FieldVisitor visitField(
    int access,
    String name,
    String desc,
    String signature,
    Object value);

/**
 * Visits a method of the class. This method <i>must</i> return a new
 * {@link MethodVisitor} instance (or <tt>null</tt>) each time it is
 * called, i.e., it should not return a previously returned visitor.
 *
 * @param access      the method's access flags (see {@link Opcodes}).
 *                    This parameter also indicates if the method is
 *                    synthetic and/or deprecated.
 * @param name        the method's name.
 * @param desc         the method's descriptor (see {@link Type Type}).
 * @param signature   the method's signature. May be <tt>null</tt> if the
 *                    method parameters, return type and exceptions do not
 *                    use generic types.
 * @param exceptions the internal names of the method's exception classes
 *                    (see
 *                    {@link Type#getInternalName() getInternalName}).
 *                    May be <tt>null</tt>.
 * @return an object to visit the byte code of the method, or
 *         <tt>null</tt> if this class visitor is not interested in

```

```

*           visiting the code of this method.
*/
MethodVisitor visitMethod(
    int access,
    String name,
    String desc,
    String signature,
    String[] exceptions);

/**
 * Visits the end of the class. This method, which is the last one to be
 * called, is used to inform the visitor that all the fields and methods
 * of the class have been visited.
 */
void visitEnd();
}

```

7.8 ClassWriter.java

(ClassVisitor [229])
— ClassWriter.java —

```

\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * A {@link ClassVisitor} that generates classes in bytecode form.
 * More precisely this visitor generates a byte array conforming to
 * the Java class file format. It can be used alone, to generate a
 * Java class "from scratch", or with one or more
 * {@link ClassReader ClassReader} and adapter class visitor
 * to generate a modified class from one or more existing Java classes.
 *
 * @author Eric Bruneton
 */
public class ClassWriter implements ClassVisitor{

/**
 * Flag to automatically compute the maximum stack size and the
 * maximum number of local variables of methods. If this flag is set,
 * then the arguments of the {@link MethodVisitor#visitMaxs visitMaxs}
 * method of the {@link MethodVisitor} returned by the
 * {@link #visitMethod visitMethod} method will be ignored, and
 * computed automatically from the signature and the bytecode of
 * each method.
*

```

```
* @see #ClassWriter(int)
*/
public final static int COMPUTE_MAXS = 1;

/**
 * Flag to automatically compute the stack map frames of methods from
 * scratch. If this flag is set, then the calls to the
 * {@link MethodVisitor#visitFrame} method are ignored, and the stack map
 * frames are recomputed from the methods bytecode. The arguments of the
 * {@link MethodVisitor#visitMaxs visitMaxs} method are also ignored and
 * recomputed from the bytecode. In other words, computeFrames implies
 * computeMaxs.
*
* @see #ClassWriter(int)
*/
public final static int COMPUTE_FRAMES = 2;

/**
 * The type of instructions without any argument.
*/
final static int NOARG_INSN = 0;

/**
 * The type of instructions with an signed byte argument.
*/
final static int SBYTE_INSN = 1;

/**
 * The type of instructions with an signed short argument.
*/
final static int SHORT_INSN = 2;

/**
 * The type of instructions with a local variable index argument.
*/
final static int VAR_INSN = 3;

/**
 * The type of instructions with an implicit local variable index
 * argument.
*/
final static int IMPLVAR_INSN = 4;

/**
 * The type of instructions with a type descriptor argument.
*/
final static int TYPE_INSN = 5;

/**
 * The type of field and method invocations instructions.
```

```
 */
final static int FIELDORMETH_INSN = 6;

/**
 * The type of the INVOKEINTERFACE instruction.
 */
final static int ITFMETH_INSN = 7;

/**
 * The type of instructions with a 2 bytes bytecode offset label.
 */
final static int LABEL_INSN = 8;

/**
 * The type of instructions with a 4 bytes bytecode offset label.
 */
final static int LABELW_INSN = 9;

/**
 * The type of the LDC instruction.
 */
final static int LDC_INSN = 10;

/**
 * The type of the LDC_W and LDC2_W instructions.
 */
final static int LDCW_INSN = 11;

/**
 * The type of the IINC instruction.
 */
final static int IINC_INSN = 12;

/**
 * The type of the TABLESWITCH instruction.
 */
final static int TABL_INSN = 13;

/**
 * The type of the LOOKUPSWITCH instruction.
 */
final static int LOOK_INSN = 14;

/**
 * The type of the MULTIANEWARRAY instruction.
 */
final static int MANA_INSN = 15;

/**
 * The type of the WIDE instruction.
```

```
/*
final static int WIDE_INSN = 16;

/**
 * The instruction types of all JVM opcodes.
 */
static byte[] TYPE;

/**
 * The type of CONSTANT_Class constant pool items.
 */
final static int CLASS = 7;

/**
 * The type of CONSTANT_Fieldref constant pool items.
 */
final static int FIELD = 9;

/**
 * The type of CONSTANT_Methodref constant pool items.
 */
final static int METH = 10;

/**
 * The type of CONSTANT_InterfaceMethodref constant pool items.
 */
final static int IMETH = 11;

/**
 * The type of CONSTANT_String constant pool items.
 */
final static int STR = 8;

/**
 * The type of CONSTANT_Integer constant pool items.
 */
final static int INT = 3;

/**
 * The type of CONSTANT_Float constant pool items.
 */
final static int FLOAT = 4;

/**
 * The type of CONSTANT_Long constant pool items.
 */
final static int LONG = 5;

/**
 * The type of CONSTANT_Double constant pool items.

```

```
 */
final static int DOUBLE = 6;

/**
 * The type of CONSTANT_NameAndType constant pool items.
 */
final static int NAME_TYPE = 12;

/**
 * The type of CONSTANT_Utf8 constant pool items.
 */
final static int UTF8 = 1;

/**
 * Normal type Item stored in the ClassWriter
 * {@link ClassWriter#typeTable}, instead of the constant pool,
 * in order to avoid clashes with normal constant pool items in
 * the ClassWriter constant pool's hash table.
 */
final static int TYPE_NORMAL = 13;

/**
 * Uninitialized type Item stored in the ClassWriter
 * {@link ClassWriter#typeTable}, instead of the constant pool, in
 * order to avoid clashes with normal constant pool items in the
 * ClassWriter constant pool's hash table.
 */
final static int TYPE_UNINIT = 14;

/**
 * Merged type Item stored in the ClassWriter
 * {@link ClassWriter#typeTable}, instead of the constant pool, in
 * order to avoid clashes with normal constant pool items in the
 * ClassWriter constant pool's hash table.
 */
final static int TYPE_MERGED = 15;

/**
 * The class reader from which this class writer was constructed, if any.
 */
ClassReader cr;

/**
 * Minor and major version numbers of the class to be generated.
 */
int version;

/**
 * Index of the next item to be added in the constant pool.
 */
```

```
int index;

/**
 * The constant pool of this class.
 */
ByteVector pool;

/**
 * The constant pool's hash table data.
 */
Item[] items;

/**
 * The threshold of the constant pool's hash table.
 */
int threshold;

/**
 * A reusable key used to look for items in the {@link #items} hash
 * table.
 */
Item key;

/**
 * A reusable key used to look for items in the {@link #items} hash
 * table.
 */
Item key2;

/**
 * A reusable key used to look for items in the {@link #items} hash
 * table.
 */
Item key3;

/**
 * A type table used to temporarily store internal names that will
 * not necessarily be stored in the constant pool. This type table
 * is used by the control flow and data flow analysis algorithm
 * used to compute stack map frames from scratch. This array
 * associates to each index <tt>i</tt> the Item whose index is
 * <tt>i</tt>. All Item objects stored in this array are also stored
 * in the {@link #items} hash table. These two arrays allow to retrieve
 * an Item from its index or, conversely, to get the index of an Item
 * from its value. Each Item stores an internal name in its
 * {@link Item#strVal1} field.
 */
Item[] typeTable;

/**
```

```
* Number of elements in the {@link #typeTable} array.  
*/  
private short typeCount; // TODO int?  
  
/**  
 * The access flags of this class.  
 */  
private int access;  
  
/**  
 * The constant pool item that contains the internal name of this  
 * class.  
 */  
private int name;  
  
/**  
 * The internal name of this class.  
 */  
String thisName;  
  
/**  
 * The constant pool item that contains the signature of this class.  
 */  
private int signature;  
  
/**  
 * The constant pool item that contains the internal name of the super  
 * class of this class.  
 */  
private int superName;  
  
/**  
 * Number of interfaces implemented or extended by this class or  
 * interface.  
 */  
private int interfaceCount;  
  
/**  
 * The interfaces implemented or extended by this class or interface.  
 * More precisely, this array contains the indexes of the constant  
 * pool items that contain the internal names of these interfaces.  
 */  
private int[] interfaces;  
  
/**  
 * The index of the constant pool item that contains the name of the  
 * source file from which this class was compiled.  
 */  
private int sourceFile;
```

```
/**  
 * The SourceDebug attribute of this class.  
 */  
private ByteVector sourceDebug;  
  
/**  
 * The constant pool item that contains the name of the enclosing class  
 * of this class.  
 */  
private int enclosingMethodOwner;  
  
/**  
 * The constant pool item that contains the name and descriptor of the  
 * enclosing method of this class.  
 */  
private int enclosingMethod;  
  
/**  
 * The runtime visible annotations of this class.  
 */  
private AnnotationWriter anns;  
  
/**  
 * The runtime invisible annotations of this class.  
 */  
private AnnotationWriter iannts;  
  
/**  
 * The non standard attributes of this class.  
 */  
private Attribute attrs;  
  
/**  
 * The number of entries in the InnerClasses attribute.  
 */  
private int innerClassesCount;  
  
/**  
 * The InnerClasses attribute.  
 */  
private ByteVector innerClasses;  
  
/**  
 * The fields of this class. These fields are stored in a linked list  
 * of {@link FieldWriter} objects, linked to each other by their  
 * {@link FieldWriter#next} field. This field stores the first element  
 * of this list.  
 */  
FieldWriter firstField;
```

```
/**  
 * The fields of this class. These fields are stored in a linked list  
 * of {@link FieldWriter} objects, linked to each other by their  
 * {@link FieldWriter#next} field. This field stores the last element  
 * of this list.  
 */  
FieldWriter lastField;  
  
/**  
 * The methods of this class. These methods are stored in a linked list  
 * of {@link MethodWriter} objects, linked to each other by their  
 * {@link MethodWriter#next} field. This field stores the first element  
 * of this list.  
 */  
MethodWriter firstMethod;  
  
/**  
 * The methods of this class. These methods are stored in a linked list  
 * of {@link MethodWriter} objects, linked to each other by their  
 * {@link MethodWriter#next} field. This field stores the last element  
 * of this list.  
 */  
MethodWriter lastMethod;  
  
/**  
 * <tt>true</tt> if the maximum stack size and number of local variables  
 * must be automatically computed.  
 */  
private boolean computeMaxs;  
  
/**  
 * <tt>true</tt> if the stack map frames must be recomputed from scratch.  
 */  
private boolean computeFrames;  
  
/**  
 * <tt>true</tt> if the stack map tables of this class are invalid. The  
 * {@link MethodWriter#resizeInstructions} method cannot transform  
 * existing stack map tables, and so produces potentially invalid  
 * classes when it is executed. In this case the class is reread  
 * and rewritten with the {@link #COMPUTE_FRAMES} option (the  
 * resizeInstructions method can resize stack map tables when this  
 * option is used).  
 */  
boolean invalidFrames;  
  
// -----  
// Static initializer  
// -----
```

```

/**
 * Computes the instruction types of JVM opcodes.
 */
static
{
    int i;
    byte[] b = new byte[220];
    String s =
        "AAAAAAAAAAAAABCKLDDDDDEEEEEEEEEEAAAAADD"
        + "DDDEEEEEEEEEEAAAAA"
        + "AAAAAAAAAAAAAMAAAAAAAIIIIIIIIIIIDNOAA"
        + "AAAAGGGGGGHAFBFAAFFAQPTIJJIIEEEEEEEE";
    for(i = 0; i < b.length; ++i)
    {
        b[i] = (byte) (s.charAt(i) - 'A');
    }
    TYPE = b;

    // code to generate the above string
    //
    // // SBYTE_INSN instructions
    // b[Constants.NEWARRAY] = SBYTE_INSN;
    // b[Constants.BIPUSH] = SBYTE_INSN;
    //
    // // SHORT_INSN instructions
    // b[Constants.SIPUSH] = SHORT_INSN;
    //
    // // (IMPL)VAR_INSN instructions
    // b[Constants.RET] = VAR_INSN;
    // for (i = Constants.ILOAD; i <= Constants.ALOAD; ++i) {
    // b[i] = VAR_INSN;
    // }
    // for (i = Constants.ISTORE; i <= Constants.ASTORE; ++i) {
    // b[i] = VAR_INSN;
    // }
    // for (i = 26; i <= 45; ++i) { // ILOAD_0 to ALOAD_3
    // b[i] = IMPLVAR_INSN;
    // }
    // for (i = 59; i <= 78; ++i) { // ISTORE_0 to ASTORE_3
    // b[i] = IMPLVAR_INSN;
    // }
    //
    // // TYPE_INSN instructions
    // b[Constants.NEW] = TYPE_INSN;
    // b[Constants.ANEWARRAY] = TYPE_INSN;
    // b[Constants.CHECKCAST] = TYPE_INSN;
    // b[Constants.INSTANCEOF] = TYPE_INSN;
    //
    // // (Set)FIELDORMETH_INSN instructions
    // for (i = Constants.GETSTATIC; i <= Constants.INVOKESTATIC; ++i) {

```

```

// b[i] = FIELDORMETH_INSN;
// }
// b[Constants.INVOKEINTERFACE] = ITFMETH_INSN;
//
// // LABEL(W)_INSN instructions
// for (i = Constants.IFEQ; i <= Constants.JSR; ++i) {
// b[i] = LABEL_INSN;
// }
// b[Constants.IFNULL] = LABEL_INSN;
// b[Constants.IFNONNULL] = LABEL_INSN;
// b[200] = LABELW_INSN; // GOTO_W
// b[201] = LABELW_INSN; // JSR_W
// // temporary opcodes used internally by ASM - see Label and
// MethodWriter
// for (i = 202; i < 220; ++i) {
// b[i] = LABEL_INSN;
// }
//
// // LDC(_W) instructions
// b[Constants.LDC] = LDC_INSN;
// b[19] = LDCW_INSN; // LDC_W
// b[20] = LDCW_INSN; // LDC2_W
//
// // special instructions
// b[Constants.IINC] = IINC_INSN;
// b[Constants.TABLESWITCH] = TABL_INSN;
// b[Constants.LOOKUPSWITCH] = LOOK_INSN;
// b[Constants.MULTIANEWARRAY] = MANA_INSN;
// b[196] = WIDE_INSN; // WIDE
//
// for (i = 0; i < b.length; ++i) {
// System.err.print((char)('A' + b[i]));
// }
// System.err.println();
}

// -----
// Constructor
// -----


/**
 * Constructs a new {@link ClassWriter} object.
 *
 * @param flags option flags that can be used to modify the default
 * behavior of this class. See {@link #COMPUTE_MAXS},
 * {@link #COMPUTE_FRAMES}.
 */
public ClassWriter(final int flags){
    index = 1;
    pool = new ByteVector();
}

```

```

        items = new Item[256];
        threshold = (int) (0.75d * items.length);
        key = new Item();
        key2 = new Item();
        key3 = new Item();
        this.computeMaxs = (flags & COMPUTE_MAXS) != 0;
        this.computeFrames = (flags & COMPUTE_FRAMES) != 0;
    }

    /**
     * Constructs a new {@link ClassWriter} object and enables optimizations
     * for "mostly add" bytecode transformations. These optimizations are
     * the following:
     * <p/>
     * <ul> <li>The constant pool from the original class is copied as is
     * in the new class, which saves time. New constant pool entries will
     * be added at the end if necessary, but unused constant pool entries
     * <i>won't be removed</i>. </li> <li>Methods that are not transformed
     * are copied as is in the new class, directly from the original class
     * bytecode (i.e. without emitting visit events for all the method
     * instructions), which saves a <i>lot</i> of time. Untransformed
     * methods are detected by the fact that the {@link ClassReader}
     * receives {@link MethodVisitor} objects that come from a
     * {@link ClassWriter} (and not from a custom {@link ClassAdapter}
     * or any other {@link ClassVisitor} instance).</li> </ul>
     *
     * @param classReader the {@link ClassReader} used to read the original
     *                    class. It will be used to copy the entire
     *                    constant pool from the original class and also
     *                    to copy other fragments of original bytecode
     *                    where applicable.
     * @param flags      option flags that can be used to modify the
     *                    default behavior of this class. See
     *                    {@link #COMPUTE_MAXS}, {@link #COMPUTE_FRAMES}.
     */
    public ClassWriter(final ClassReader classReader, final int flags){
        this(flags);
        classReader.copyPool(this);
        this.cr = classReader;
    }

    // -----
    // Implementation of the ClassVisitor interface
    // -----
    public void visit(
        final int version,
        final int access,
        final String name,
        final String signature,

```

```
    final String superName,
    final String[] interfaces){
this.version = version;
this.access = access;
this.name = newClass(name);
thisName = name;
if(signature != null)
{
    this.signature = newUTF8(signature);
}
this.superName = superName == null ? 0 : newClass(superName);
if(interfaces != null && interfaces.length > 0)
{
    interfaceCount = interfaces.length;
    this.interfaces = new int[interfaceCount];
    for(int i = 0; i < interfaceCount; ++i)
    {
        this.interfaces[i] = newClass(interfaces[i]);
    }
}
}

public void visitSource(final String file, final String debug){
    if(file != null)
    {
        sourceFile = newUTF8(file);
    }
    if(debug != null)
    {
        sourceDebug = new ByteVector().putUTF8(debug);
    }
}

public void visitOuterClass(
    final String owner,
    final String name,
    final String desc){
enclosingMethodOwner = newClass(owner);
if(name != null && desc != null)
{
    enclosingMethod = newNameType(name, desc);
}
}

public AnnotationVisitor visitAnnotation(
    final String desc,
    final boolean visible){
ByteVector bv = new ByteVector();
// write type, and reserve space for values count
bv.putShort(newUTF8(desc)).putShort(0);
```

```

AnnotationWriter aw = new AnnotationWriter(this, true, bv, bv, 2);
if(visible)
{
    {
        aw.next = anns;
        anns = aw;
    }
}
else
{
    {
        aw.next = ianns;
        ianns = aw;
    }
}
return aw;
}

public void visitAttribute(final Attribute attr){
    attr.next = attrs;
    attrs = attr;
}

public void visitInnerClass(
    final String name,
    final String outerName,
    final String innerName,
    final int access){
    if(innerClasses == null)
    {
        innerClasses = new ByteVector();
    }
    ++innerClassesCount;
    innerClasses.putShort(name == null ? 0 : newClass(name));
    innerClasses.putShort(outerName == null ? 0 : newClass(outerName));
    innerClasses.putShort(innerName == null ? 0 : newUTF8(innerName));
    innerClasses.putShort(access);
}

public FieldVisitor visitField(
    final int access,
    final String name,
    final String desc,
    final String signature,
    final Object value){
    return new FieldWriter(this, access, name, desc, signature, value);
}

public MethodVisitor visitMethod(
    final int access,
    final String name,
    final String desc,
    final String signature,
    final String[] exceptions){
}

```

```
        return new MethodWriter(this,
                               access,
                               name,
                               desc,
                               signature,
                               exceptions,
                               computeMaxs,
                               computeFrames);
    }

    public void visitEnd(){
    }

    // -----
    // Other public methods
    // -----
}

/** 
 * Returns the bytecode of the class that was build with this class
 * writer.
 *
 * @return the bytecode of the class that was build with this class
 *         writer.
 */
public byte[] toByteArray(){
    // computes the real size of the bytecode of this class
    int size = 24 + 2 * interfaceCount;
    int nbFields = 0;
    FieldWriter fb = firstField;
    while(fb != null)
    {
        ++
        nbFields;
        size += fb.getSize();
        fb = fb.next;
    }
    int nbMethods = 0;
    MethodWriter mb = firstMethod;
    while(mb != null)
    {
        ++
        nbMethods;
        size += mb.getSize();
        mb = mb.next;
    }
    int attributeCount = 0;
    if(signature != 0)
    {
        ++
        attributeCount;
        size += 8;
        newUTF8("Signature");
    }
}
```

```
if(sourceFile != 0)
{
    ++
    attributeCount;
    size += 8;
    newUTF8("SourceFile");
}
if(sourceDebug != null)
{
    ++
    attributeCount;
    size += sourceDebug.length + 4;
    newUTF8("SourceDebugExtension");
}
if(enclosingMethodOwner != 0)
{
    ++
    attributeCount;
    size += 10;
    newUTF8("EnclosingMethod");
}
if((access & Opcodes.ACC_DEPRECATED) != 0)
{
    ++
    attributeCount;
    size += 6;
    newUTF8("Deprecated");
}
if((access & Opcodes.ACC_SYNTHETIC) != 0
    && (version & 0xffff) < Opcodes.V1_5)
{
    ++
    attributeCount;
    size += 6;
    newUTF8("Synthetic");
}
if(innerClasses != null)
{
    ++
    attributeCount;
    size += 8 + innerClasses.length;
    newUTF8("InnerClasses");
}
if(anns != null)
{
    ++
    attributeCount;
    size += 8 + anns.getSize();
    newUTF8("RuntimeVisibleAnnotations");
}
if(ianns != null)
{
    ++
    attributeCount;
    size += 8 + ianns.getSize();
    newUTF8("RuntimeInvisibleAnnotations");
}
if(attrs != null)
```

```
{  
    attributeCount += attrs.getCount();  
    size += attrs.getSize(this, null, 0, -1, -1);  
}  
size += pool.length;  
// allocates a byte vector of this size, in order to  
// avoid unnecessary arraycopy operations in the  
// ByteVector.enlarge() method  
ByteVector out = new ByteVector(size);  
out.putInt(0xCAFEBABE).putInt(version);  
out.putShort(index).putByteArray(pool.data, 0, pool.length);  
out.putShort(access).putShort(name).putShort(superName);  
out.putShort(interfaceCount);  
for(int i = 0; i < interfaceCount; ++i)  
{  
    out.putShort(interfaces[i]);  
}  
out.putShort(nbFields);  
fb = firstField;  
while(fb != null)  
{  
    fb.put(out);  
    fb = fb.next;  
}  
out.putShort(nbMethods);  
mb = firstMethod;  
while(mb != null)  
{  
    mb.put(out);  
    mb = mb.next;  
}  
out.putShort(attributeCount);  
if(signature != 0)  
{  
    out.putShort(newUTF8("Signature"))  
        .putInt(2)  
        .putShort(signature);  
}  
if(sourceFile != 0)  
{  
    out.putShort(newUTF8("SourceFile"))  
        .putInt(2)  
        .putShort(sourceFile);  
}  
if(sourceDebug != null)  
{  
    int len = sourceDebug.length - 2;  
    out.putShort(newUTF8("SourceDebugExtension"))  
        .putInt(len);  
    out.putByteArray(sourceDebug.data, 2, len);  
}
```

```

        }
        if(enclosingMethodOwner != 0)
        {
            out.putShort(newUTF8("EnclosingMethod"))
                .putInt(4);
            out.putShort(enclosingMethodOwner)
                .putShort(enclosingMethod);
        }
        if((access & Opcodes.ACC_DEPRECATED) != 0)
        {
            out.putShort(newUTF8("Deprecated")).putInt(0);
        }
        if((access & Opcodes.ACC_SYNTHETIC) != 0
            && (version & 0xffff) < Opcodes.V1_5)
        {
            out.putShort(newUTF8("Synthetic")).putInt(0);
        }
        if(innerClasses != null)
        {
            out.putShort(newUTF8("InnerClasses"));
            out.putInt(innerClasses.length + 2)
                .putShort(innerClassesCount);
            out.putByteArray(innerClasses.data,
                            0, innerClasses.length);
        }
        if(anns != null)
        {
            out.putShort(newUTF8("RuntimeVisibleAnnotations"));
            anns.put(out);
        }
        if(ianns != null)
        {
            out.putShort(newUTF8("RuntimeInvisibleAnnotations"));
            ianns.put(out);
        }
        if(attrs != null)
        {
            attrs.put(this, null, 0, -1, -1, out);
        }
        if(invalidFrames)
        {
            ClassWriter cw = new ClassWriter(COMPUTE_FRAMES);
            new ClassReader(out.data).accept(cw, ClassReader.SKIP_FRAMES);
            return cw.toByteArray();
        }
        return out.data;
    }

    // -----
    // Utility methods: constant pool management
}

```

```
// -----  
  
/**  
 * Adds a number or string constant to the constant pool of the class  
 * being build. Does nothing if the constant pool already contains  
 * a similar item.  
 *  
 * @param cst the value of the constant to be added to the constant  
 * pool. This parameter must be an {@link Integer},  
 * a {@link Float}, a {@link Long}, a {@link Double},  
 * a {@link String} or a {@link Type}.  
 * @return a new or already existing constant item with the given value.  
 */  
Item newConstItem(final Object cst){  
    if(cst instanceof Integer)  
    {  
        int val = ((Integer) cst).intValue();  
        return newInteger(val);  
    }  
    else if(cst instanceof Byte)  
    {  
        int val = ((Byte) cst).intValue();  
        return newInteger(val);  
    }  
    else if(cst instanceof Character)  
    {  
        int val = ((Character) cst).charValue();  
        return newInteger(val);  
    }  
    else if(cst instanceof Short)  
    {  
        int val = ((Short) cst).intValue();  
        return newInteger(val);  
    }  
    else if(cst instanceof Boolean)  
    {  
        int val = ((Boolean) cst).booleanValue() ? 1 : 0;  
        return newInteger(val);  
    }  
    else if(cst instanceof Float)  
    {  
        float val = ((Float) cst).floatValue();  
        return newFloat(val);  
    }  
    else if(cst instanceof Long)  
    {  
        long val = ((Long) cst).longValue();  
        return newLong(val);  
    }  
    else if(cst instanceof Double)
```

```

    {
        double val = ((Double) cst).doubleValue();
        return newDouble(val);
    }
    else if(cst instanceof String)
    {
        return newString((String) cst);
    }
    else if(cst instanceof Type)
    {
        Type t = (Type) cst;
        return newClassItem(t.getSort() == Type.OBJECT
            ? t.getInternalName()
            : t.getDescriptor());
    }
    else
    {
        throw new IllegalArgumentException("value " + cst);
    }
}

/**
 * Adds a number or string constant to the constant pool of the class
 * being build. Does nothing if the constant pool already contains a
 * similar item. <i>This method is intended for {@link Attribute}
 * sub classes, and is normally not needed by class generators or
 * adapters.</i>
 *
 * @param cst the value of the constant to be added to the constant pool.
 *           This parameter must be an {@link Integer}, a {@link Float},
 *           a {@link Long}, a {@link Double} or a {@link String}.
 * @return the index of a new or already existing constant item with the
 *         given value.
 */
public int newConst(final Object cst){
    return newConstItem(cst).index;
}

/**
 * Adds an UTF8 string to the constant pool of the class being build.
 * Does nothing if the constant pool already contains a similar item.
 * <i>This method is intended for {@link Attribute} sub classes, and
 * is normally not needed by class generators or adapters.</i>
 *
 * @param value the String value.
 * @return the index of a new or already existing UTF8 item.
 */
public int newUTF8(final String value){
    key.set(UTF8, value, null, null);
    Item result = get(key);
}

```

```
    if(result == null)
    {
        pool.putByte(UTF8).putUTF8(value);
        result = new Item(index++, key);
        put(result);
    }
    return result.index;
}

<**
 * Adds a class reference to the constant pool of the class being build.
 * Does nothing if the constant pool already contains a similar item.
 * <i>This method is intended for {@link Attribute} sub classes, and is
 * normally not needed by class generators or adapters.</i>
 *
 * @param value the internal name of the class.
 * @return a new or already existing class reference item.
 */
Item newClassItem(final String value){
    key2.set(CLASS, value, null, null);
    Item result = get(key2);
    if(result == null)
    {
        pool.put12(CLASS, newUTF8(value));
        result = new Item(index++, key2);
        put(result);
    }
    return result;
}

<**
 * Adds a class reference to the constant pool of the class being build.
 * Does nothing if the constant pool already contains a similar item.
 * <i>This method is intended for {@link Attribute} sub classes, and is
 * normally not needed by class generators or adapters.</i>
 *
 * @param value the internal name of the class.
 * @return the index of a new or already existing class reference item.
 */
public int newClass(final String value){
    return newClassItem(value).index;
}

<**
 * Adds a field reference to the constant pool of the class being build.
 * Does nothing if the constant pool already contains a similar item.
 *
 * @param owner the internal name of the field's owner class.
 * @param name the field's name.
 * @param desc the field's descriptor.
 */
```

```

 * @return a new or already existing field reference item.
 */
Item newFieldItem(final String owner,
                  final String name,
                  final String desc){
    key3.set(FIELD, owner, name, desc);
    Item result = get(key3);
    if(result == null)
    {
        put122(FIELD, newClass(owner), newNameType(name, desc));
        result = new Item(index++, key3);
        put(result);
    }
    return result;
}

/**
 * Adds a field reference to the constant pool of the class being build.
 * Does nothing if the constant pool already contains a similar item.
 * <i>This method is intended for {@link Attribute} sub classes, and is
 * normally not needed by class generators or adapters.</i>
 *
 * @param owner the internal name of the field's owner class.
 * @param name the field's name.
 * @param desc the field's descriptor.
 * @return the index of a new or already existing field reference item.
 */
public int newField(final String owner,
                    final String name,
                    final String desc){
    return newFieldItem(owner, name, desc).index;
}

/**
 * Adds a method reference to the constant pool of the class being build.
 * Does nothing if the constant pool already contains a similar item.
 *
 * @param owner the internal name of the method's owner class.
 * @param name the method's name.
 * @param desc the method's descriptor.
 * @param itf <tt>true</tt> if <tt>owner</tt> is an interface.
 * @return a new or already existing method reference item.
 */
Item newMethodItem(
    final String owner,
    final String name,
    final String desc,
    final boolean itf){
    int type = itf ? IMETH : METH;
    key3.set(type, owner, name, desc);
}

```

```
Item result = get(key3);
if(result == null)
{
    put122(type, newClass(owner), newNameType(name, desc));
    result = new Item(index++, key3);
    put(result);
}
return result;
}

/**
 * Adds a method reference to the constant pool of the class being build.
 * Does nothing if the constant pool already contains a similar item.
 * <i>This method is intended for {@link Attribute} sub classes, and is
 * normally not needed by class generators or adapters.</i>
 *
 * @param owner the internal name of the method's owner class.
 * @param name the method's name.
 * @param desc the method's descriptor.
 * @param itf   <tt>true</tt> if <tt>owner</tt> is an interface.
 * @return the index of a new or already existing method reference item.
 */
public int newMethod(
    final String owner,
    final String name,
    final String desc,
    final boolean itf){
    return newMethodItem(owner, name, desc, itf).index;
}

/**
 * Adds an integer to the constant pool of the class being build. Does
 * nothing if the constant pool already contains a similar item.
 *
 * @param value the int value.
 * @return a new or already existing int item.
 */
Item newInteger(final int value){
    key.set(value);
    Item result = get(key);
    if(result == null)
    {
        pool.putByte(INT).putInt(value);
        result = new Item(index++, key);
        put(result);
    }
    return result;
}

/**
```

```
* Adds a float to the constant pool of the class being build. Does
* nothing if the constant pool already contains a similar item.
*
* @param value the float value.
* @return a new or already existing float item.
*/
Item newFloat(final float value){
    key.set(value);
    Item result = get(key);
    if(result == null)
    {
        pool.putByte(FLOAT).putInt(key.intValue());
        result = new Item(index++, key);
        put(result);
    }
    return result;
}

/**
* Adds a long to the constant pool of the class being build. Does
* nothing if the constant pool already contains a similar item.
*
* @param value the long value.
* @return a new or already existing long item.
*/
Item newLong(final long value){
    key.set(value);
    Item result = get(key);
    if(result == null)
    {
        pool.putByte(LONG).putLong(value);
        result = new Item(index, key);
        put(result);
        index += 2;
    }
    return result;
}

/**
* Adds a double to the constant pool of the class being build. Does
* nothing if the constant pool already contains a similar item.
*
* @param value the double value.
* @return a new or already existing double item.
*/
Item newDouble(final double value){
    key.set(value);
    Item result = get(key);
    if(result == null)
    {
```

```
        pool.putByte(DOUBLE).putLong(key.longVal);
        result = new Item(index, key);
        put(result);
        index += 2;
    }
    return result;
}

/**
 * Adds a string to the constant pool of the class being build. Does
 * nothing if the constant pool already contains a similar item.
 *
 * @param value the String value.
 * @return a new or already existing string item.
 */
private Item newString(final String value){
    key2.set(STR, value, null, null);
    Item result = get(key2);
    if(result == null)
    {
        pool.put12(STR, newUTF8(value));
        result = new Item(index++, key2);
        put(result);
    }
    return result;
}

/**
 * Adds a name and type to the constant pool of the class being build.
 * Does nothing if the constant pool already contains a similar item.
 * <i>This method is intended for {@link Attribute} sub classes, and
 * is normally not needed by class generators or adapters.</i>
 *
 * @param name a name.
 * @param desc a type descriptor.
 * @return the index of a new or already existing name and type item.
 */
public int newNameType(final String name, final String desc){
    key2.set(NAME_TYPE, name, desc, null);
    Item result = get(key2);
    if(result == null)
    {
        put122(NAME_TYPE, newUTF8(name), newUTF8(desc));
        result = new Item(index++, key2);
        put(result);
    }
    return result.index;
}

/**
```

```

* Adds the given internal name to {@link #typeTable} and returns its
* index. Does nothing if the type table already contains this internal
* name.
*
* @param type the internal name to be added to the type table.
* @return the index of this internal name in the type table.
*/
int addType(final String type){
    key.set(TYPE_NORMAL, type, null, null);
    Item result = get(key);
    if(result == null)
    {
        result = addType(key);
    }
    return result.index;
}

/**
* Adds the given "uninitialized" type to {@link #typeTable} and returns
* its index. This method is used for UNINITIALIZED types, made of an
* internal name and a bytecode offset.
*
* @param type the internal name to be added to the type table.
* @param offset the bytecode offset of the NEW instruction that created
*               this UNINITIALIZED type value.
* @return the index of this internal name in the type table.
*/
int addUninitializedType(final String type, final int offset){
    key.type = TYPE_UNINIT;
    key.intVal = offset;
    key.strVal1 = type;
    key.hashCode =
        0x7FFFFFFF & (TYPE_UNINIT + type.hashCode() + offset);
    Item result = get(key);
    if(result == null)
    {
        result = addType(key);
    }
    return result.index;
}

/**
* Adds the given Item to {@link #typeTable}.
*
* @param item the value to be added to the type table.
* @return the added Item, which a new Item instance with the same
*         value as the given Item.
*/
private Item addType(final Item item){
    ++typeCount;
}

```

```

Item result = new Item(typeCount, key);
put(result);
if(typeTable == null)
{
    typeTable = new Item[16];
}
if(typeCount == typeTable.length)
{
    Item[] newTable = new Item[2 * typeTable.length];
    System.arraycopy(typeTable, 0,
                     newTable, 0, typeTable.length);
    typeTable = newTable;
}
typeTable[typeCount] = result;
return result;
}

/**
 * Returns the index of the common super type of the two given types.
 * This method calls {@link #getCommonSuperClass} and caches the result
 * in the {@link #items} hash table to speedup future calls with the
 * same parameters.
 *
 * @param type1 index of an internal name in {@link #typeTable}.
 * @param type2 index of an internal name in {@link #typeTable}.
 * @return the index of the common super type of the two given types.
 */
int getMergedType(final int type1, final int type2){
    key2.type = TYPE_MERGED;
    key2.longVal = type1 | ((long) type2) << 32;
    key2.hashCode = 0x7FFFFFFF & (TYPE_MERGED + type1 + type2);
    Item result = get(key2);
    if(result == null)
    {
        String t = typeTable[type1].strVal1;
        String u = typeTable[type2].strVal1;
        key2.intVal = addType(getCommonSuperClass(t, u));
        result = new Item((short) 0, key2);
        put(result);
    }
    return result.intVal;
}

/**
 * Returns the common super type of the two given types. The default
 * implementation of this method <i>loads</i> the two given classes
 * and uses the java.lang.Class methods to find the common super class.
 * It can be overriden to compute this common super type in other ways,
 * in particular without actually loading any class, or to take into
 * account the class that is currently being generated by this

```

```

* ClassWriter, which can of course not be loaded since it is under
* construction.
*
* @param type1 the internal name of a class.
* @param type2 the internal name of another class.
* @return the internal name of the common super class of the two given
*         classes.
*/
protected String getCommonSuperClass(final String type1,
                                    final String type2){
    Class c, d;
    try
    {
        c = Class.forName(type1.replace('/', '.'));
        d = Class.forName(type2.replace('/', '.'));
    }
    catch(ClassNotFoundException e)
    {
        throw new RuntimeException(e);
    }
    if(c.isAssignableFrom(d))
    {
        return type1;
    }
    if(d.isAssignableFrom(c))
    {
        return type2;
    }
    if(c.isInterface() || d.isInterface())
    {
        return "java/lang/Object";
    }
    else
    {
        do
        {
            c = c.getSuperclass();
        } while(!c.isAssignableFrom(d));
        return c.getName().replace('.', '/');
    }
}

/**
 * Returns the constant pool's hash table item which is equal to the
 * given item.
 *
 * @param key a constant pool item.
 * @return the constant pool's hash table item which is equal to the
 *         given item, or <tt>null</tt> if there is no such item.
 */

```

```

private Item get(final Item key){
    Item i = items[key.hashCode % items.length];
    while(i != null && !key.isEqualTo(i))
    {
        i = i.next;
    }
    return i;
}

/**
 * Puts the given item in the constant pool's hash table. The hash
 * table <i>must</i> not already contains this item.
 *
 * @param i the item to be added to the constant pool's hash table.
 */
private void put(final Item i){
    if(index > threshold)
    {
        int ll = items.length;
        int nl = ll * 2 + 1;
        Item[] newItems = new Item[nl];
        for(int l = ll - 1; l >= 0; --l)
        {
            Item j = items[l];
            while(j != null)
            {
                int index = j.hashCode % newItems.length;
                Item k = j.next;
                j.next = newItems[index];
                newItems[index] = j;
                j = k;
            }
        }
        items = newItems;
        threshold = (int) (nl * 0.75);
    }
    int index = i.hashCode % items.length;
    i.next = items[index];
    items[index] = i;
}

/**
 * Puts one byte and two shorts into the constant pool.
 *
 * @param b a byte.
 * @param s1 a short.
 * @param s2 another short.
 */
private void put122(final int b, final int s1, final int s2){
    pool.put12(b, s1).putShort(s2);
}

```

```
 }
 }
```

—————

7.9 Edge.java

— Edge.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * An edge in the control flow graph of a method body. See
 * {@link Label Label}.
 *
 * @author Eric Bruneton
 */
class Edge{

    /**
     * Denotes a normal control flow graph edge.
     */
    final static int NORMAL = 0;

    /**
     * Denotes a control flow graph edge corresponding to an exception
     * handler. More precisely any {@link Edge} whose {@link #info} is
     * strictly positive corresponds to an exception handler. The actual
     * value of {@link #info} is the index, in the {@link ClassWriter}
     * type table, of the exception that is catched.
     */
    final static int EXCEPTION = 0x7FFFFFFF;

    /**
     * Information about this control flow graph edge. If
     * {@link ClassWriter#COMPUTE_MAXS} is used this field is the
     * (relative) stack size in the basic block from which this edge
     * originates. This size is equal to the stack size at the "jump"
     * instruction to which this edge corresponds, relatively to the
     * stack size at the beginning of the originating basic block.
     * If {@link ClassWriter#COMPUTE_FRAMES} is used, this field is
     * the kind of this control flow graph edge (i.e. NORMAL or EXCEPTION).
     */
    int info;

    /**

```

```

 * The successor block of the basic block from which this edge
 * originates.
 */
Label successor;

/**
 * The next edge in the list of successors of the originating basic
 * block. See {@link Label#successors successors}.
 */
Edge next;
}

```

7.10 FieldVisitor.java

— FieldVisitor.java —

```

\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * A visitor to visit a Java field. The methods of this interface
 * must be called in the following order: ( <tt>visitAnnotation</tt> |
 * <tt>visitAttribute</tt> )* <tt>visitEnd</tt>.
 *
 * @author Eric Bruneton
 */
public interface FieldVisitor{

/**
 * Visits an annotation of the field.
 *
 * @param desc    the class descriptor of the annotation class.
 * @param visible <tt>true</tt> if the annotation is visible at runtime.
 * @return a visitor to visit the annotation values, or <tt>null</tt> if
 *         this visitor is not interested in visiting this annotation.
 */
AnnotationVisitor visitAnnotation(String desc, boolean visible);

/**
 * Visits a non standard attribute of the field.
 *
 * @param attr an attribute.
 */
void visitAttribute(Attribute attr);

```

```
/**
 * Visits the end of the field. This method, which is the last one to be
 * called, is used to inform the visitor that all the annotations and
 * attributes of the field have been visited.
 */
void visitEnd();
}
```

7.11 FieldWriter.java

(FieldVisitor [263])
— FieldWriter.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * An {@link FieldVisitor} that generates Java fields in bytecode form.
 *
 * @author Eric Bruneton
 */
final class FieldWriter implements FieldVisitor{

/**
 * Next field writer (see {@link ClassWriter#firstField firstField}).
 */
FieldWriter next;

/**
 * The class writer to which this field must be added.
 */
private ClassWriter cw;

/**
 * Access flags of this field.
 */
private int access;

/**
 * The index of the constant pool item that contains the name of this
 * method.
 */
private int name;

/**
 * The index of the constant pool item that contains the descriptor of
```

```
* this field.  
*/  
private int desc;  
  
/**  
 * The index of the constant pool item that contains the signature of  
 * this field.  
 */  
private int signature;  
  
/**  
 * The index of the constant pool item that contains the constant value  
 * of this field.  
 */  
private int value;  
  
/**  
 * The runtime visible annotations of this field. May be <tt>null</tt>.  
 */  
private AnnotationWriter anns;  
  
/**  
 * The runtime invisible annotations of this field. May be <tt>null</tt>.  
 */  
private AnnotationWriter ianns;  
  
/**  
 * The non standard attributes of this field. May be <tt>null</tt>.  
 */  
private Attribute attrs;  
  
// -----  
// Constructor  
// -----  
  
/**  
 * Constructs a new {@link FieldWriter}.  
 *  
 * @param cw      the class writer to which this field must be added.  
 * @param access  the field's access flags (see {@link Opcodes}).  
 * @param name    the field's name.  
 * @param desc    the field's descriptor (see {@link Type}).  
 * @param signature the field's signature. May be <tt>null</tt>.  
 * @param value   the field's constant value. May be <tt>null</tt>.  
 */  
protected FieldWriter(  
    final ClassWriter cw,  
    final int access,  
    final String name,  
    final String desc,
```

```

        final String signature,
        final Object value){
    if(cw.firstField == null)
    {
        cw.firstField = this;
    }
    else
    {
        cw.lastField.next = this;
    }
    cw.lastField = this;
    this.cw = cw;
    this.access = access;
    this.name = cw.newUTF8(name);
    this.desc = cw.newUTF8(desc);
    if(signature != null)
    {
        this.signature = cw.newUTF8(signature);
    }
    if(value != null)
    {
        this.value = cw.newConstItem(value).index;
    }
}

// -----
// Implementation of the FieldVisitor interface
// -----


public AnnotationVisitor visitAnnotation(
    final String desc,
    final boolean visible){
    ByteVector bv = new ByteVector();
    // write type, and reserve space for values count
    bv.putShort(cw.newUTF8(desc)).putShort(0);
    AnnotationWriter aw = new AnnotationWriter(cw, true, bv, bv, 2);
    if(visible)
    {
        aw.next = anns;
        anns = aw;
    }
    else
    {
        aw.next = ianns;
        ianns = aw;
    }
    return aw;
}

public void visitAttribute(final Attribute attr){

```

```
        attr.next = attrs;
        attrs = attr;
    }

    public void visitEnd(){
    }

    // -----
    // Utility methods
    // -----

    /**
     * Returns the size of this field.
     *
     * @return the size of this field.
     */
    int getSize(){
        int size = 8;
        if(value != 0)
        {
            cw.newUTF8("ConstantValue");
            size += 8;
        }
        if((access & Opcodes.ACC_SYNTHETIC) != 0
           && (cw.version & 0xffff) < Opcodes.V1_5)
        {
            cw.newUTF8("Synthetic");
            size += 6;
        }
        if((access & Opcodes.ACC_DEPRECATED) != 0)
        {
            cw.newUTF8("Deprecated");
            size += 6;
        }
        if(signature != 0)
        {
            cw.newUTF8("Signature");
            size += 8;
        }
        if(anns != null)
        {
            cw.newUTF8("RuntimeVisibleAnnotations");
            size += 8 + anns.getSize();
        }
        if(ianns != null)
        {
            cw.newUTF8("RuntimeInvisibleAnnotations");
            size += 8 + ianns.getSize();
        }
        if(attrs != null)
```

```

    {
        size += attrs.getSize(cw, null, 0, -1, -1);
    }
    return size;
}

/**
 * Puts the content of this field into the given byte vector.
 *
 * @param out where the content of this field must be put.
 */
void put(final ByteVector out){
    out.putShort(access).putShort(name).putShort(desc);
    int attributeCount = 0;
    if(value != 0)
    {
        ++
        attributeCount;
    }
    if((access & Opcodes.ACC_SYNTHETIC) != 0
        && (cw.version & 0xffff) < Opcodes.V1_5)
    {
        ++
        attributeCount;
    }
    if((access & Opcodes.ACC_DEPRECATED) != 0)
    {
        ++
        attributeCount;
    }
    if(signature != 0)
    {
        ++
        attributeCount;
    }
    if(anns != null)
    {
        ++
        attributeCount;
    }
    if(ianns != null)
    {
        ++
        attributeCount;
    }
    if(attrs != null)
    {
        attributeCount += attrs.getCount();
    }
    out.putShort(attributeCount);
    if(value != 0)
    {
        out.putShort(cw.newUTF8("ConstantValue"));
        out.putInt(2).putShort(value);
    }
    if((access & Opcodes.ACC_SYNTHETIC) != 0

```

```

    && (cw.version & 0xffff) < Opcodes.V1_5)
    {
        out.putShort(cw.newUTF8("Synthetic")).putInt(0);
    }
    if((access & Opcodes.ACC_DEPRECATED) != 0)
    {
        out.putShort(cw.newUTF8("Deprecated")).putInt(0);
    }
    if(signature != 0)
    {
        out.putShort(cw.newUTF8("Signature"));
        out.putInt(2).putShort(signature);
    }
    if(anns != null)
    {
        out.putShort(cw.newUTF8("RuntimeVisibleAnnotations"));
        anns.put(out);
    }
    if(ianns != null)
    {
        out.putShort(cw.newUTF8("RuntimeInvisibleAnnotations"));
        ianns.put(out);
    }
    if(attrs != null)
    {
        attrs.put(cw, null, 0, -1, -1, out);
    }
}
}

```

—————

7.12 Frame.java

— Frame.java —

```

\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * Information about the input and output stack map frames of a basic
 * block.
 *
 * @author Eric Bruneton
 */
final class Frame{

```

```

/*
 * Frames are computed in a two steps process: during the visit
 * of each instruction, the state of the frame at the end of
 * current basic block is updated by simulating the action of
 * the instruction on the previous state of this so called
 * "output frame". In visitMaxs, a fix point algorithm is used
 * to compute the "input frame" of each basic block, i.e. the
 * stack mapframe at the begining of the basic block, starting
 * from the input frameof the first basic block (which is
 * computed from the method descriptor),and by using the
 * previously computed output frames to compute the input
 * state of the other blocks.
 *
 * All output and input frames are stored as arrays of
 * integers. Reference and array types are represented by an
 * index into a type table (which is not the same as the constant
 * pool of the class, in order to avoid adding unnecessary
 * constants in the pool - not all computed frames will end up
 * being stored in the stack map table). This allows very fast
 * type comparisons.
 *
 * Output stack map frames are computed relatively to the input
 * frame of the basic block, which is not yet known when output
 * frames are computed. It is therefore necessary to be able to
 * represent abstract types such as "the type at position x in
 * the input frame locals" or "the type at position x from the
 * top of the input frame stack" or even "the type at position
 * x in the input frame, with y more (or less) array dimensions".
 * This explains the rather complicated type format used in
 * output frames.
 *
 * This format is the following: DIM KIND VALUE (4, 4 and 24
 * bits). DIM is a signed number of array dimensions (from -8 to
 * 7). KIND is either BASE, LOCAL or STACK. BASE is used for
 * types that are not relative to the input frame. LOCAL is used
 * for types that are relative to the input local variable types.
 * STACK is used for types that are relative to the input stack
 * types. VALUE depends on KIND. For LOCAL types, it is an index
 * in the input local variable types. For STACK types, it is a
 * position relatively to the top of input frame stack. For BASE
 * types, it is either one of the constants defined in
 * FrameVisitor, or for OBJECT and UNINITIALIZED types, a tag
 * and an index in the type table.
 *
 * Output frames can contain types of any kind and with a
 * positive or negative dimension (and even unassigned types,
 * represented by 0 - which does not correspond to any valid
 * type value). Input frames can only contain BASE types of
 * positive or null dimension. In all cases the type table
 * contains only internal type names (array type descriptors

```

```
* are forbidden - dimensions must be represented through the
* DIM field).
*
* The LONG and DOUBLE types are always represented by using
* two slots (LONG + TOP or DOUBLE + TOP), for local variable
* types as well as in the operand stack. This is necessary to
* be able to simulate DUPx_y instructions, whose effect would
* be dependent on the actual type values if types were always
* represented by a single slot in the stack (and this is not
* possible, since actual type values are not always known -
* cf LOCAL and STACK type kinds).
*/
/***
 * Mask to get the dimension of a frame type. This dimension is a signed
 * integer between -8 and 7.
 */
final static int DIM = 0xF0000000;

/***
 * Constant to be added to a type to get a type with one more dimension.
 */
final static int ARRAY_OF = 0x10000000;

/***
 * Constant to be added to a type to get a type with one less dimension.
 */
final static int ELEMENT_OF = 0xF0000000;

/***
 * Mask to get the kind of a frame type.
 *
 * @see #BASE
 * @see #LOCAL
 * @see #STACK
 */
final static int KIND = 0xF0000000;

/***
 * Mask to get the value of a frame type.
 */
final static int VALUE = 0xFFFFFFF;

/***
 * Mask to get the kind of base types.
 */
final static int BASE_KIND = 0xFF000000;

/***
 * Mask to get the value of base types.
*/
```

```
/*
final static int BASE_VALUE = 0xFFFFF;

/**
 * Kind of the types that are not relative to an input stack map frame.
 */
final static int BASE = 0x1000000;

/**
 * Base kind of the base reference types. The BASE_VALUE of such types
 * is an index into the type table.
 */
final static int OBJECT = BASE | 0x700000;

/**
 * Base kind of the uninitialized base types. The BASE_VALUE of such
 * types in an index into the type table (the Item at that index
 * contains both an instruction offset and an internal class name).
 */
final static int UNINITIALIZED = BASE | 0x800000;

/**
 * Kind of the types that are relative to the local variable types of
 * an input stack map frame. The value of such types is a local
 * variable index.
 */
private final static int LOCAL = 0x2000000;

/**
 * Kind of the the types that are relative to the stack of an input
 * stack map frame. The value of such types is a position relatively
 * to the top of this stack.
 */
private final static int STACK = 0x3000000;

/**
 * The TOP type. This is a BASE type.
 */
final static int TOP = BASE | 0;

/**
 * The BOOLEAN type. This is a BASE type mainly used for array types.
 */
final static int BOOLEAN = BASE | 9;

/**
 * The BYTE type. This is a BASE type mainly used for array types.
 */
final static int BYTE = BASE | 10;
```

```
/**  
 * The CHAR type. This is a BASE type mainly used for array types.  
 */  
final static int CHAR = BASE | 11;  
  
/**  
 * The SHORT type. This is a BASE type mainly used for array types.  
 */  
final static int SHORT = BASE | 12;  
  
/**  
 * The INTEGER type. This is a BASE type.  
 */  
final static int INTEGER = BASE | 1;  
  
/**  
 * The FLOAT type. This is a BASE type.  
 */  
final static int FLOAT = BASE | 2;  
  
/**  
 * The DOUBLE type. This is a BASE type.  
 */  
final static int DOUBLE = BASE | 3;  
  
/**  
 * The LONG type. This is a BASE type.  
 */  
final static int LONG = BASE | 4;  
  
/**  
 * The NULL type. This is a BASE type.  
 */  
final static int NULL = BASE | 5;  
  
/**  
 * The UNINITIALIZED_THIS type. This is a BASE type.  
 */  
final static int UNINITIALIZED_THIS = BASE | 6;  
  
/**  
 * The stack size variation corresponding to each JVM instruction.  
 * This stack variation is equal to the size of the values produced  
 * by an instruction, minus the size of the values consumed by this  
 * instruction.  
 */  
final static int[] SIZE;  
  
/**  
 * Computes the stack size variation corresponding to each JVM
```

```

* instruction.
*/
static
{
    int i;
    int[] b = new int[202];
    String s =
        "EFFFFFFFFGGFFFGGFFEEFGFGFEEEEEEEEEEEEEEEEEDEDEDDDDD"
        + "CDCDEEEEEEEEEEEEEEEBABBBBBDCFFGGGEDCDCDCDCDCDCDC"
        + "CDCEEEEDDDDDDDCDCDCDFEFDDEEFFDEEEEBDBBDDDDCCCCCCC"
        + "DDCDCDEEEEEEEEEEFEEEEDDEEDDEE";
    for(i = 0; i < b.length; ++i)
    {
        b[i] = s.charAt(i) - 'E';
    }
    SIZE = b;

    // code to generate the above string
    //
    // int NA = 0; // not applicable (unused opcode or variable
    //               size opcode)
    //
    // b = new int[] {
    // 0, //NOP, // visitInsn
    // 1, //ACONST_NULL, // -
    // 1, //ICONST_M1, // -
    // 1, //ICONST_0, // -
    // 1, //ICONST_1, // -
    // 1, //ICONST_2, // -
    // 1, //ICONST_3, // -
    // 1, //ICONST_4, // -
    // 1, //ICONST_5, // -
    // 2, //LCONST_0, // -
    // 2, //LCONST_1, // -
    // 1, //FCONST_0, // -
    // 1, //FCONST_1, // -
    // 1, //FCONST_2, // -
    // 2, //DCONST_0, // -
    // 2, //DCONST_1, // -
    // 1, //BIPUSH, // visitIntInsn
    // 1, //SIPUSH, // -
    // 1, //LDC, // visitLdcInsn
    // NA, //LDC_W, // -
    // NA, //LDC2_W, // -
    // 1, //ILOAD, // visitVarInsn
    // 2, //LLOAD, // -
    // 1, //FLOAD, // -
    // 2, //DLOAD, // -
    // 1, //ALOAD, // -
    // NA, //ILOAD_0, // -
}

```

```
// NA, //ILOAD_1, // -
// NA, //ILOAD_2, // -
// NA, //ILOAD_3, // -
// NA, //LLOAD_0, // -
// NA, //LLOAD_1, // -
// NA, //LLOAD_2, // -
// NA, //LLOAD_3, // -
// NA, //FLOAD_0, // -
// NA, //FLOAD_1, // -
// NA, //FLOAD_2, // -
// NA, //FLOAD_3, // -
// NA, //DLOAD_0, // -
// NA, //DLOAD_1, // -
// NA, //DLOAD_2, // -
// NA, //DLOAD_3, // -
// NA, //ALOAD_0, // -
// NA, //ALOAD_1, // -
// NA, //ALOAD_2, // -
// NA, //ALOAD_3, // -
// -1, //IALOAD, // visitInsn
// 0, //LALOAD, // -
// -1, //FALOAD, // -
// 0, //DALOAD, // -
// -1, //AALOAD, // -
// -1, //BALOAD, // -
// -1, //CALOAD, // -
// -1, //SALOAD, // -
// -1, //ISTORE, // visitVarInsn
// -2, //LSTORE, // -
// -1, //FSTORE, // -
// -2, //DSTORE, // -
// -1, //ASTORE, // -
// NA, //ISTORE_0, // -
// NA, //ISTORE_1, // -
// NA, //ISTORE_2, // -
// NA, //ISTORE_3, // -
// NA, //LSTORE_0, // -
// NA, //LSTORE_1, // -
// NA, //LSTORE_2, // -
// NA, //LSTORE_3, // -
// NA, //FSTORE_0, // -
// NA, //FSTORE_1, // -
// NA, //FSTORE_2, // -
// NA, //FSTORE_3, // -
// NA, //DSTORE_0, // -
// NA, //DSTORE_1, // -
// NA, //DSTORE_2, // -
// NA, //DSTORE_3, // -
// NA, //ASTORE_0, // -
// NA, //ASTORE_1, // -
```

```
// NA, //ASTORE_2, //
// NA, //ASTORE_3, //
// -3, //IASTORE, // visitInsn
// -4, //LASTORE, //
// -3, //FASTORE, //
// -4, //DASTORE, //
// -3, //AASTORE, //
// -3, //BASTORE, //
// -3, //CASTORE, //
// -3, //SASTORE, //
// -1, //POP, //
// -2, //POP2, //
// 1, //DUP, //
// 1, //DUP_X1, //
// 1, //DUP_X2, //
// 2, //DUP2, //
// 2, //DUP2_X1, //
// 2, //DUP2_X2, //
// 0, //SWAP, //
// -1, //IADD, //
// -2, //LADD, //
// -1, //FADD, //
// -2, //DADD, //
// -1, //ISUB, //
// -2, //LSUB, //
// -1, //FSUB, //
// -2, //DSUB, //
// -1, //IMUL, //
// -2, //LMUL, //
// -1, //FMUL, //
// -2, //DMUL, //
// -1, //IDIV, //
// -2, //LDIV, //
// -1, //FDIV, //
// -2, //DDIV, //
// -1, //IREM, //
// -2, //LREM, //
// -1, //FREM, //
// -2, //DREM, //
// 0, //INEG, //
// 0, //LNEG, //
// 0, //FNEG, //
// 0, //DNEG, //
// -1, //ISHL, //
// -1, //LSHL, //
// -1, //ISHR, //
// -1, //LSHR, //
// -1, //IUSHR, //
// -1, //LUSHR, //
// -1, //IAND, //
```

```
// -2, //LAND, //
// -1, //IOR, //
// -2, //LOR, //
// -1, //IXOR, //
// -2, //LXOR, //
// 0, //IINC, // visitIincInsn
// 1, //I2L, // visitInsn
// 0, //I2F, //
// 1, //I2D, //
// -1, //L2I, //
// -1, //L2F, //
// 0, //L2D, //
// 0, //F2I, //
// 1, //F2L, //
// 1, //F2D, //
// -1, //D2I, //
// 0, //D2L, //
// -1, //D2F, //
// 0, //I2B, //
// 0, //I2C, //
// 0, //I2S, //
// -3, //LCMP, //
// -1, //FCMPL, //
// -1, //FCMPG, //
// -3, //DCMPL, //
// -3, //DCMPG, //
// -1, //IFEQ, // visitJumpInsn
// -1, //IFNE, //
// -1, //IFLT, //
// -1, //IFGE, //
// -1, //IFGT, //
// -1, //IFLE, //
// -2, //IF_ICMPEQ, //
// -2, //IF_ICMPNE, //
// -2, //IF_ICMPLT, //
// -2, //IF_ICMPGE, //
// -2, //IF_ICMPGT, //
// -2, //IF_ICMPLE, //
// -2, //IF_ACMPEQ, //
// -2, //IF_ACMPNE, //
// 0, //GOTO, //
// 1, //JSR, //
// 0, //RET, // visitVarInsn
// -1, //TABLESWITCH, // visitTableSwitchInsn
// -1, //LOOKUPSWITCH, // visitLookupSwitch
// -1, //IRETURN, // visitInsn
// -2, //LRETURN, //
// -1, //FRETURN, //
// -2, //DRETURN, //
// -1, //ARETURN, //
```

```

// 0, //RETURN, // -
// NA, //GETSTATIC, // visitFieldInsn
// NA, //PUTSTATIC, // -
// NA, //GETFIELD, // -
// NA, //PUTFIELD, // -
// NA, //INVOKEVIRTUAL, // visitMethodInsn
// NA, //INVOKEESPECIAL, // -
// NA, //INVOKESTATIC, // -
// NA, //INVOKEINTERFACE, // -
// NA, //UNUSED, // NOT VISITED
// 1, //NEW, // visitTypeInsn
// 0, //NEWARRAY, // visitIntInsn
// 0, //ANEWARRAY, // visitTypeInsn
// 0, //ARRAYLENGTH, // visitInsn
// NA, //ATHROW, // -
// 0, //CHECKCAST, // visitTypeInsn
// 0, //INSTANCEOF, // -
// -1, //MONITORENTER, // visitInsn
// -1, //MONITOREXIT, // -
// NA, //WIDE, // NOT VISITED
// NA, //MULTIANEWARRAY, // visitMultiANewArrayInsn
// -1, //IFNULL, // visitJumpInsn
// -1, //IFNONNULL, // -
// NA, //GOTO_W, // -
// NA, //JSR_W, // -
// };
// for (i = 0; i < b.length; ++i) {
// System.err.print((char)('E' + b[i]));
// }
// System.err.println();
}

/**
 * The label (i.e. basic block) to which these input and output stack map
 * frames correspond.
 */
Label owner;

/**
 * The input stack map frame locals.
 */
int[] inputLocals;

/**
 * The input stack map frame stack.
 */
int[] inputStack;

/**
 * The output stack map frame locals.

```

```
 */
private int[] outputLocals;

/**
 * The output stack map frame stack.
 */
private int[] outputStack;

/**
 * Relative size of the output stack. The exact semantics of this field
 * depends on the algorithm that is used.
 * <p/>
 * When only the maximum stack size is computed, this field is the size
 * of the output stack relatively to the top of the input stack.
 * <p/>
 * When the stack map frames are completely computed, this field is the
 * actual number of types in {@link #outputStack}.
 */
private int outputStackTop;

/**
 * Number of types that are initialized in the basic block.
 *
 * @see #initializations
 */
private int initializationCount;

/**
 * The types that are initialized in the basic block. A constructor
 * invocation on an UNINITIALIZED or UNINITIALIZED_THIS type must
 * replace <i>every occurrence</i> of this type in the local variables
 * and in the operand stack. This cannot be done during the first
 * phase of the algorithm since, during this phase, the local variables
 * and the operand stack are not completely computed. It is therefore
 * necessary to store the types on which constructors are invoked in
 * the basic block, in order to do this replacement during the second
 * phase of the algorithm, where the frames are fully computed. Note
 * that this array can contain types that are relative to input locals
 * or to the input stack (see below for the description of the
 * algorithm).
 */
private int[] initializations;

/**
 * Returns the output frame local variable type at the given index.
 *
 * @param local the index of the local that must be returned.
 * @return the output frame local variable type at the given index.
 */
private int get(final int local){
```

```

if(outputLocals == null || local >= outputLocals.length)
{
    // this local has never been assigned in this basic block,
    // so it is still equal to its value in the input frame
    return LOCAL | local;
}
else
{
    int type = outputLocals[local];
    if(type == 0)
    {
        // this local has never been assigned in this
        // basic block, so it is still equal to its value
        // in the input frame
        type = outputLocals[local] = LOCAL | local;
    }
    return type;
}

/***
 * Sets the output frame local variable type at the given index.
 *
 * @param local the index of the local that must be set.
 * @param type the value of the local that must be set.
 */
private void set(final int local, final int type){
    // creates and/or resizes the output local variables
    // array if necessary
    if(outputLocals == null)
    {
        outputLocals = new int[10];
    }
    int n = outputLocals.length;
    if(local >= n)
    {
        int[] t = new int[Math.max(local + 1, 2 * n)];
        System.arraycopy(outputLocals, 0, t, 0, n);
        outputLocals = t;
    }
    // sets the local variable
    outputLocals[local] = type;
}

/***
 * Pushes a new type onto the output frame stack.
 *
 * @param type the type that must be pushed.
 */
private void push(final int type){

```

```

// creates and/or resizes the output stack array if necessary
if(outputStack == null)
{
    outputStack = new int[10];
}
int n = outputStack.length;
if(outputStackTop >= n)
{
    int[] t = new int[Math.max(outputStackTop + 1, 2 * n)];
    System.arraycopy(outputStack, 0, t, 0, n);
    outputStack = t;
}
// pushes the type on the output stack
outputStack[outputStackTop++] = type;
// updates the maximum height reached by the output stack, if needed
int top = owner.inputStackTop + outputStackTop;
if(top > owner.outputStackMax)
{
    owner.outputStackMax = top;
}
}

/**
 * Pushes a new type onto the output frame stack.
 *
 * @param cw the ClassWriter to which this label belongs.
 * @param desc the descriptor of the type to be pushed. Can also be a
 *             method descriptor (in this case this method pushes its
 *             return type onto the output frame stack).
 */
private void push(final ClassWriter cw, final String desc){
    int type = type(cw, desc);
    if(type != 0)
    {
        push(type);
        if(type == LONG || type == DOUBLE)
        {
            push(TOP);
        }
    }
}

/**
 * Returns the int encoding of the given type.
 *
 * @param cw the ClassWriter to which this label belongs.
 * @param desc a type descriptor.
 * @return the int encoding of the given type.
 */
private int type(final ClassWriter cw, final String desc){

```

```

String t;
int index = desc.charAt(0) == '(' ? desc.indexOf(')') + 1 : 0;
switch(desc.charAt(index))
{
    case'V':
        return 0;
    case'Z':
    case'C':
    case'B':
    case'S':
    case'I':
        return INTEGER;
    case'F':
        return FLOAT;
    case'J':
        return LONG;
    case'D':
        return DOUBLE;
    case'L':
        // stores the internal name, not the descriptor!
        t = desc.substring(index + 1, desc.length() - 1);
        return OBJECT | cw.addType(t);
        // case '[':
    default:
        // extracts the dimensions and the element type
        int data;
        int dims = index + 1;
        while(desc.charAt(dims) == '[')
        {
            ++dims;
        }
        switch(desc.charAt(dims))
        {
            case'Z':
                data = BOOLEAN;
                break;
            case'C':
                data = CHAR;
                break;
            case'B':
                data = BYTE;
                break;
            case'S':
                data = SHORT;
                break;
            case'I':
                data = INTEGER;
                break;
            case'F':
                data = FLOAT;

```

```

        break;
    case 'J':
        data = LONG;
        break;
    case 'D':
        data = DOUBLE;
        break;
        // case 'L':
    default:
        // stores the internal name, not the descriptor
        t = desc.substring(dims + 1, desc.length() - 1);
        data = OBJECT | cw.addType(t);
    }
    return (dims - index) << 28 | data;
}

/***
 * Pops a type from the output frame stack and returns its value.
 *
 * @return the type that has been popped from the output frame stack.
 */
private int pop(){
    if(outputStackTop > 0)
    {
        return outputStack[--outputStackTop];
    }
    else
    {
        // if the output frame stack is empty, pops from the input stack
        return STACK | -(--owner.inputStackTop);
    }
}

/***
 * Pops the given number of types from the output frame stack.
 *
 * @param elements the number of types that must be popped.
 */
private void pop(final int elements){
    if(outputStackTop >= elements)
    {
        outputStackTop -= elements;
    }
    else
    {
        // if the number of elements to be popped is greater than the
        // number of elements in the output stack, clear it, and pops
        // the remaining elements from the input stack.
        owner.inputStackTop -= elements - outputStackTop;
    }
}

```

```

        outputStackTop = 0;
    }
}

/***
 * Pops a type from the output frame stack.
 *
 * @param desc the descriptor of the type to be popped. Can also be a
 *             method descriptor (in this case this method pops the
 *             types corresponding to the method arguments).
 */
private void pop(final String desc){
    char c = desc.charAt(0);
    if(c == '(')
    {
        pop((MethodWriter.getArgumentsAndReturnSizes(desc) >> 2) - 1);
    }
    else if(c == 'J' || c == 'D')
    {
        pop(2);
    }
    else
    {
        pop(1);
    }
}

/***
 * Adds a new type to the list of types on which a constructor is
 * invoked in the basic block.
 *
 * @param var a type on which a constructor is invoked.
 */
private void init(final int var){
    // creates and/or resizes the initializations array if necessary
    if(initializations == null)
    {
        initializations = new int[2];
    }
    int n = initializations.length;
    if(initializationCount >= n)
    {
        int[] t = new int[Math.max(initializationCount + 1, 2 * n)];
        System.arraycopy(initializations, 0, t, 0, n);
        initializations = t;
    }
    // stores the type to be initialized
    initializations[initializationCount++] = var;
}

```

```

/**
 * Replaces the given type with the appropriate type if it is one of
 * the types on which a constructor is invoked in the basic block.
 *
 * @param cw the ClassWriter to which this label belongs.
 * @param t a type
 * @return t or, if t is one of the types on which a constructor is
 *         invoked in the basic block, the type corresponding to this
 *         constructor.
 */
private int init(final ClassWriter cw, final int t){
    int s;
    if(t == UNINITIALIZED_THIS)
    {
        s = OBJECT | cw.addType(cw.thisName);
    }
    else if((t & (DIM | BASE_KIND)) == UNINITIALIZED)
    {
        String type = cw.typeTable[t & BASE_VALUE].strVal1;
        s = OBJECT | cw.addType(type);
    }
    else
    {
        return t;
    }
    for(int j = 0; j < initializationCount; ++j)
    {
        int u = initializations[j];
        int dim = u & DIM;
        int kind = u & KIND;
        if(kind == LOCAL)
        {
            u = dim + inputLocals[u & VALUE];
        }
        else if(kind == STACK)
        {
            u = dim + inputStack[inputStack.length - (u & VALUE)];
        }
        if(t == u)
        {
            return s;
        }
    }
    return t;
}

/**
 * Initializes the input frame of the first basic block from the method
 * descriptor.
 *

```

```

* @param cw      the ClassWriter to which this label belongs.
* @param access  the access flags of the method to which this
*                 label belongs.
* @param args    the formal parameter types of this method.
* @param maxLocals the maximum number of local variables of this method.
*/
void initInputFrame(
    final ClassWriter cw,
    final int access,
    final Type[] args,
    final int maxLocals){
    inputLocals = new int[maxLocals];
    inputStack = new int[0];
    int i = 0;
    if((access & Opcodes.ACC_STATIC) == 0)
    {
        if((access & MethodWriter.ACC_CONSTRUCTOR) == 0)
        {
            inputLocals[i++] = OBJECT | cw.addType(cw.thisName);
        }
        else
        {
            inputLocals[i++] = UNINITIALIZED_THIS;
        }
    }
    for(int j = 0; j < args.length; ++j)
    {
        int t = type(cw, args[j].getDescriptor());
        inputLocals[i++] = t;
        if(t == LONG || t == DOUBLE)
        {
            inputLocals[i++] = TOP;
        }
    }
    while(i < maxLocals)
    {
        inputLocals[i++] = TOP;
    }
}

/**
 * Simulates the action of the given instruction on the output stack
 * frame.
 *
 * @param opcode the opcode of the instruction.
 * @param arg    the operand of the instruction, if any.
 * @param cw     the class writer to which this label belongs.
 * @param item   the operand of the instructions, if any.
 */
void execute(

```

```
    final int opcode,
    final int arg,
    final ClassWriter cw,
    final Item item){
    int t1, t2, t3, t4;
    switch(opcode)
    {
        case Opcodes.NOP:
        case Opcodes.INEG:
        case Opcodes.LNEG:
        case Opcodes.FNEG:
        case Opcodes.DNEG:
        case Opcodes.I2B:
        case Opcodes.I2C:
        case Opcodes.I2S:
        case Opcodes.GOTO:
        case Opcodes.RETURN:
            break;
        case Opcodes.ACONST_NULL:
            push(NULL);
            break;
        case OpcodesICONST_M1:
        case OpcodesICONST_0:
        case OpcodesICONST_1:
        case OpcodesICONST_2:
        case OpcodesICONST_3:
        case OpcodesICONST_4:
        case OpcodesICONST_5:
        case Opcodes.BIPUSH:
        case Opcodes.SIPUSH:
        case Opcodes.ILOAD:
            push(INTEGER);
            break;
        case Opcodes.LCONST_0:
        case Opcodes.LCONST_1:
        case Opcodes.LLOAD:
            push(LONG);
            push(TOP);
            break;
        case Opcodes.FCONST_0:
        case Opcodes.FCONST_1:
        case Opcodes.FCONST_2:
        case Opcodes.FLOAD:
            push(FLOAT);
            break;
        case Opcodes.DCONST_0:
        case Opcodes.DCONST_1:
        case Opcodes.DLOAD:
            push(DOUBLE);
            push(TOP);
```

```
        break;
    case Opcodes.LDC:
        switch(item.type)
        {
            case ClassWriter.INT:
                push(INTEGER);
                break;
            case ClassWriter.LONG:
                push(LONG);
                push(TOP);
                break;
            case ClassWriter.FLOAT:
                push(FLOAT);
                break;
            case ClassWriter.DOUBLE:
                push(DOUBLE);
                push(TOP);
                break;
            case ClassWriter.CLASS:
                push(OBJECT | cw.addType("java/lang/Class"));
                break;
            // case ClassWriter.STR:
            default:
                push(OBJECT | cw.addType("java/lang/String"));
        }
        break;
    case Opcodes.ALOAD:
        push(get(arg));
        break;
    case Opcodes.IALOAD:
    case Opcodes.BALOAD:
    case Opcodes.CALOAD:
    case Opcodes.SALOAD:
        pop(2);
        push(INTEGER);
        break;
    case Opcodes.LALOAD:
    case Opcodes.D2L:
        pop(2);
        push(LONG);
        push(TOP);
        break;
    case Opcodes.FALOAD:
        pop(2);
        push(FLOAT);
        break;
    case Opcodes.DALOAD:
    case Opcodes.L2D:
        pop(2);
        push(DOUBLE);
```

```
    push(TOP);
    break;
case Opcodes.AALOAD:
    pop(1);
    t1 = pop();
    push(ELEMENT_OF + t1);
    break;
case Opcodes.ISTORE:
case Opcodes.FSTORE:
case Opcodes.ASTORE:
    t1 = pop();
    set(arg, t1);
    if(arg > 0)
    {
        t2 = get(arg - 1);
        // if t2 is of kind STACK or LOCAL we cannot know
        // its size!
        if(t2 == LONG || t2 == DOUBLE)
        {
            set(arg - 1, TOP);
        }
    }
    break;
case Opcodes.LSTORE:
case Opcodes.DSTORE:
    pop(1);
    t1 = pop();
    set(arg, t1);
    set(arg + 1, TOP);
    if(arg > 0)
    {
        t2 = get(arg - 1);
        // if t2 is of kind STACK or LOCAL we cannot know
        // its size!
        if(t2 == LONG || t2 == DOUBLE)
        {
            set(arg - 1, TOP);
        }
    }
    break;
case Opcodes.IASTORE:
case Opcodes.BASTORE:
case Opcodes.CASTORE:
case Opcodes.SASTORE:
case Opcodes.FASTORE:
case Opcodes.AASTORE:
    pop(3);
    break;
case Opcodes.LASTORE:
case Opcodes.DASTORE:
```

```
    pop(4);
    break;
case Opcodes.POP:
case Opcodes.IFEQ:
case Opcodes.IFNE:
case Opcodes.IFLT:
case Opcodes.IFGE:
case Opcodes.IFGT:
case Opcodes.IFLE:
case Opcodes.IRETURN:
case Opcodes.FRETURN:
case Opcodes.ARETURN:
case Opcodes.TABLESWITCH:
case Opcodes.LOOKUPSWITCH:
case Opcodes.ATHROW:
case Opcodes.MONITORENTER:
case Opcodes.MONITOREXIT:
case Opcodes.IFNULL:
case Opcodes.IFNONNULL:
    pop(1);
    break;
case Opcodes.POP2:
case Opcodes.IF_ICMPEQ:
case Opcodes.IF_ICMPNE:
case Opcodes.IF_ICMPLT:
case Opcodes.IF_ICMPGE:
case Opcodes.IF_ICMPGT:
case Opcodes.IF_ICMPLE:
case Opcodes.IF_ACMPEQ:
case Opcodes.IF_ACMPNE:
case Opcodes.LRETURN:
case Opcodes.DRETURN:
    pop(2);
    break;
case Opcodes.DUP:
    t1 = pop();
    push(t1);
    push(t1);
    break;
case Opcodes.DUP_X1:
    t1 = pop();
    t2 = pop();
    push(t1);
    push(t2);
    push(t1);
    break;
case Opcodes.DUP_X2:
    t1 = pop();
    t2 = pop();
    t3 = pop();
```

```
    push(t1);
    push(t3);
    push(t2);
    push(t1);
    break;
  case Opcodes.DUP2:
    t1 = pop();
    t2 = pop();
    push(t2);
    push(t1);
    push(t2);
    push(t1);
    break;
  case Opcodes.DUP2_X1:
    t1 = pop();
    t2 = pop();
    t3 = pop();
    push(t2);
    push(t1);
    push(t3);
    push(t2);
    push(t1);
    break;
  case Opcodes.DUP2_X2:
    t1 = pop();
    t2 = pop();
    t3 = pop();
    t4 = pop();
    push(t2);
    push(t1);
    push(t4);
    push(t3);
    push(t2);
    push(t1);
    break;
  case Opcodes.SWAP:
    t1 = pop();
    t2 = pop();
    push(t1);
    push(t2);
    break;
  case Opcodes.IADD:
  case Opcodes.ISUB:
  case Opcodes.IMUL:
  case Opcodes.IDIV:
  case Opcodes.IREM:
  case Opcodes.IAND:
  case Opcodes.IOR:
  case Opcodes.IXOR:
  case Opcodes.ISHL:
```

```
case Opcodes.ISHR:
case Opcodes.IUSHR:
case Opcodes.L2I:
case Opcodes.D2I:
case Opcodes.FCMPL:
case Opcodes.FCMPG:
    pop(2);
    push(INTEGER);
    break;
case Opcodes.LADD:
case Opcodes.LSUB:
case Opcodes.LMUL:
case Opcodes.LDIV:
case Opcodes.LREM:
case Opcodes.LAND:
case Opcodes.LOR:
case Opcodes.LXOR:
    pop(4);
    push(LONG);
    push(TOP);
    break;
case Opcodes.FADD:
case Opcodes.FSUB:
case Opcodes.FMUL:
case Opcodes.FDIV:
case Opcodes.FREM:
case Opcodes.L2F:
case Opcodes.D2F:
    pop(2);
    push(FLOAT);
    break;
case Opcodes.DADD:
case Opcodes.DSUB:
case Opcodes.DMUL:
case Opcodes.DDIV:
case Opcodes.DREM:
    pop(4);
    push(DOUBLE);
    push(TOP);
    break;
case Opcodes.LSHL:
case Opcodes.LSHR:
case Opcodes.LUSHR:
    pop(3);
    push(LONG);
    push(TOP);
    break;
case Opcodes.IINC:
    set(arg, INTEGER);
    break;
```

```
case Opcodes.I2L:
case Opcodes.F2L:
    pop(1);
    push(LONG);
    push(TOP);
    break;
case Opcodes.I2F:
    pop(1);
    push(FLOAT);
    break;
case Opcodes.I2D:
case Opcodes.F2D:
    pop(1);
    push(DOUBLE);
    push(TOP);
    break;
case Opcodes.F2I:
case Opcodes.ARRAYLENGTH:
case Opcodes.INSTANCEOF:
    pop(1);
    push(INTEGER);
    break;
case Opcodes.LCMP:
case Opcodes.DCMPL:
case Opcodes.DCMPG:
    pop(4);
    push(INTEGER);
    break;
case Opcodes.JSR:
case Opcodes.RET:
    throw new RuntimeException(
        "JSR/RET are not supported with computeFrames option");
case Opcodes.GETSTATIC:
    push(cw, item.strVal3);
    break;
case Opcodes.PUTSTATIC:
    pop(item.strVal3);
    break;
case Opcodes.GETFIELD:
    pop(1);
    push(cw, item.strVal3);
    break;
case Opcodes.PUTFIELD:
    pop(item.strVal3);
    pop();
    break;
case Opcodes.INVOKEVIRTUAL:
case Opcodes.INVOKESPECIAL:
case Opcodes.INVOKESTATIC:
case Opcodes.INVOKEINTERFACE:
```

```

pop(item.strVal3);
if(opcode != Opcodes.INVOKESTATIC)
{
    t1 = pop();
    if(opcode == Opcodes.INVOKESPECIAL
        && item.strVal2.charAt(0) == '<')
    {
        init(t1);
    }
}
push(cw, item.strVal3);
break;
case Opcodes.NEW:
    push(UNINITIALIZED |
        cw.addUninitializedType(item.strVal1, arg));
    break;
case Opcodes.NEWARRAY:
    pop();
    switch(arg)
    {
        case Opcodes.T_BOOLEAN:
            push(ARRAY_OF | BOOLEAN);
            break;
        case Opcodes.T_CHAR:
            push(ARRAY_OF | CHAR);
            break;
        case Opcodes.T_BYTE:
            push(ARRAY_OF | BYTE);
            break;
        case Opcodes.T_SHORT:
            push(ARRAY_OF | SHORT);
            break;
        case Opcodes.T_INT:
            push(ARRAY_OF | INTEGER);
            break;
        case Opcodes.T_FLOAT:
            push(ARRAY_OF | FLOAT);
            break;
        case Opcodes.T_DOUBLE:
            push(ARRAY_OF | DOUBLE);
            break;
        // case Opcodes.T_LONG:
        default:
            push(ARRAY_OF | LONG);
            break;
    }
    break;
case Opcodes.ANEWARRAY:
    String s = item.strVal1;
    pop();

```

```

        if(s.charAt(0) == '[')
        {
            push(cw, "[" + s);
        }
        else
        {
            push(ARRAY_OF | OBJECT | cw.addType(s));
        }
        break;
    case Opcodes.CHECKCAST:
        s = item.strVal1;
        pop();
        if(s.charAt(0) == '[')
        {
            push(cw, s);
        }
        else
        {
            push(OBJECT | cw.addType(s));
        }
        break;
    // case Opcodes.MULTIANEWARRAY:
    default:
        pop(arg);
        push(cw, item.strVal1);
        break;
    }
}

/**
 * Merges the input frame of the given basic block with the input and
 * output frames of this basic block. Returns <tt>true</tt> if the
 * input frame of the given label has been changed by this operation.
 *
 * @param cw    the ClassWriter to which this label belongs.
 * @param frame the basic block whose input frame must be updated.
 * @param edge  the kind of the {@link Edge} between this label and
 *             'label'. See {@link Edge#info}.
 * @return <tt>true</tt> if the input frame of the given label has been
 *         changed by this operation.
 */
boolean merge(final ClassWriter cw, final Frame frame, final int edge){
    boolean changed = false;
    int i, s, dim, kind, t;

    int nLocal = inputLocals.length;
    int nStack = inputStack.length;
    if(frame.inputLocals == null)
    {
        frame.inputLocals = new int[nLocal];
    }
}

```

```

changed = true;
}

for(i = 0; i < nLocal; ++i)
{
    if(outputLocals != null && i < outputLocals.length)
    {
        s = outputLocals[i];
        if(s == 0)
        {
            t = inputLocals[i];
        }
        else
        {
            dim = s & DIM;
            kind = s & KIND;
            if(kind == LOCAL)
            {
                t = dim + inputLocals[s & VALUE];
            }
            else if(kind == STACK)
            {
                t = dim + inputStack[nStack - (s & VALUE)];
            }
            else
            {
                t = s;
            }
        }
    }
    else
    {
        t = inputLocals[i];
    }
    if(initializations != null)
    {
        t = init(cw, t);
    }
    changed |= merge(cw, t, frame.inputLocals, i);
}

if(edge > 0)
{
    for(i = 0; i < nLocal; ++i)
    {
        t = inputLocals[i];
        changed |= merge(cw, t, frame.inputLocals, i);
    }
    if(frame.inputStack == null)
    {
}
}

```

```

        frame.inputStack = new int[1];
        changed = true;
    }
    changed |= merge(cw, edge, frame.inputStack, 0);
    return changed;
}

int nInputStack = inputStack.length + owner.inputStackTop;
if(frame.inputStack == null)
{
    frame.inputStack = new int[nInputStack + outputStackTop];
    changed = true;
}

for(i = 0; i < nInputStack; ++i)
{
    t = inputStack[i];
    if(initializations != null)
    {
        t = init(cw, t);
    }
    changed |= merge(cw, t, frame.inputStack, i);
}
for(i = 0; i < outputStackTop; ++i)
{
    s = outputStack[i];
    dim = s & DIM;
    kind = s & KIND;
    if(kind == LOCAL)
    {
        t = dim + inputLocals[s & VALUE];
    }
    else if(kind == STACK)
    {
        t = dim + inputStack[nStack - (s & VALUE)];
    }
    else
    {
        t = s;
    }
    if(initializations != null)
    {
        t = init(cw, t);
    }
    changed |= merge(cw, t, frame.inputStack, nInputStack + i);
}
return changed;
}

/**

```

```

* Merges the type at the given index in the given type array with
* the given type. Returns <tt>true</tt> if the type array has been
* modified by this operation.
*
* @param cw    the ClassWriter to which this label belongs.
* @param t     the type with which the type array element must be
*              merged.
* @param types an array of types.
* @param index the index of the type that must be merged in 'types'.
* @return <tt>true</tt> if the type array has been modified by this
*         operation.
*/
private boolean merge(
    final ClassWriter cw,
    int t,
    final int[] types,
    final int index){
    int u = types[index];
    if(u == t)
    {
        //
        // if the types are equal, merge(u,t)=u, so there is no change
        return false;
    }
    if((t & ~DIM) == NULL)
    {
        if(u == NULL)
        {
            return false;
        }
        t = NULL;
    }
    if(u == 0)
    {
        //
        // if types[index] has never been assigned, merge(u,t)=t
        types[index] = t;
        return true;
    }
    int v;
    if((u & BASE_KIND) == OBJECT || (u & DIM) != 0)
    {
        //
        // if u is a reference type of any dimension
        if(t == NULL)
        {
            //
            // if t is the NULL type, merge(u,t)=u, so there is no change
            return false;
        }
        else if((t & (DIM | BASE_KIND)) == (u & (DIM | BASE_KIND)))
        {
            if((u & BASE_KIND) == OBJECT)
            {

```

```

// if t is also a reference type, and if u and t have
// the same dimension merge(u,t) = dim(t) | common parent
// of the element types of u and t
v = (t & DIM) | OBJECT
    | cw.getMergedType(t & BASE_VALUE, u & BASE_VALUE);
}
else
{
    //
    // if u and t are array types, but not with the same
    // element type, merge(u,t)=java/lang/Object
    v = OBJECT | cw.addType("java/lang/Object");
}
}
else if((t & BASE_KIND) == OBJECT || (t & DIM) != 0)
{
    //
    // if t is any other reference or array type,
    // merge(u,t)=java/lang/Object
    v = OBJECT | cw.addType("java/lang/Object");
}
else
{
    //
    // if t is any other type, merge(u,t)=TOP
    v = TOP;
}
}
else if(u == NULL)
{
    //
    // if u is the NULL type, merge(u,t)=t,
    // or TOP if t is not a reference type
    v = (t & BASE_KIND) == OBJECT || (t & DIM) != 0 ? t : TOP;
}
else
{
    //
    // if u is any other type, merge(u,t)=TOP whatever t
    v = TOP;
}
if(u != v)
{
    types[index] = v;
    return true;
}
return false;
}
}

```

7.13 Handler.java

— Handler.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * Information about an exception handler block.
 *
 * @author Eric Bruneton
 */
class Handler{

    /**
     * Beginning of the exception handler's scope (inclusive).
     */
    Label start;

    /**
     * End of the exception handler's scope (exclusive).
     */
    Label end;

    /**
     * Beginning of the exception handler's code.
     */
    Label handler;

    /**
     * Internal name of the type of exceptions handled by this handler, or
     * <tt>null</tt> to catch any exceptions.
     */
    String desc;

    /**
     * Constant pool index of the internal name of the type of exceptions
     * handled by this handler, or 0 to catch any exceptions.
     */
    int type;

    /**
     * Next exception handler block info.
     */
    Handler next;
}
```

7.14 Item.java

— Item.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * A constant pool item. Constant pool items can be created with the
 * 'newXXX' methods in the {@link ClassWriter} class.
 *
 * @author Eric Bruneton
 */
final class Item{

    /**
     * Index of this item in the constant pool.
     */
    int index;

    /**
     * Type of this constant pool item. A single class is used to represent
     * all constant pool item types, in order to minimize the bytecode size
     * of this package. The value of this field is one of
     * {@link ClassWriter#INT},
     * {@link ClassWriter#LONG}, {@link ClassWriter#FLOAT},
     * {@link ClassWriter#DOUBLE}, {@link ClassWriter#UTF8},
     * {@link ClassWriter#STR}, {@link ClassWriter#CLASS},
     * {@link ClassWriter#NAME_TYPE}, {@link ClassWriter#FIELD},
     * {@link ClassWriter#METH}, {@link ClassWriter#IMETH}.
     * <p/>
     * Special Item types are used for Items that are stored in the
     * ClassWriter {@link ClassWriter#typeTable}, instead of the constant
     * pool, in order to avoid clashes with normal constant pool items in
     * the ClassWriter constant pool's hash table. These special item types
     * are {@link ClassWriter#TYPE_NORMAL}, {@link ClassWriter#TYPE_UNINIT}
     * and {@link ClassWriter#TYPE_MERGED}.
     */
    int type;

    /**
     * Value of this item, for an integer item.
     */
    int intValue;

    /**
     * Value of this item, for a long item.
     */
}
```

```
long longVal;

/**
 * First part of the value of this item, for items that do not hold a
 * primitive value.
 */
String strVal1;

/**
 * Second part of the value of this item, for items that do not hold a
 * primitive value.
 */
String strVal2;

/**
 * Third part of the value of this item, for items that do not hold a
 * primitive value.
 */
String strVal3;

/**
 * The hash code value of this constant pool item.
 */
int hashCode;

/**
 * Link to another constant pool item, used for collision lists in the
 * constant pool's hash table.
 */
Item next;

/**
 * Constructs an uninitialized {@link Item}.
 */
Item(){
}

/**
 * Constructs an uninitialized {@link Item} for constant pool element at
 * given position.
 *
 * @param index index of the item to be constructed.
 */
Item(final int index){
    this.index = index;
}

/**
 * Constructs a copy of the given item.
 *
```

```
* @param index index of the item to be constructed.  
* @param i      the item that must be copied into the item to be  
*               constructed.  
*/  
Item(final int index, final Item i){  
    this.index = index;  
    type = i.type;  
    intValue = i.intValue;  
    longVal = i.longVal;  
    strVal1 = i.strVal1;  
    strVal2 = i.strVal2;  
    strVal3 = i.strVal3;  
    hashCode = i.hashCode;  
}  
  
/**  
 * Sets this item to an integer item.  
 *  
 * @param intValue the value of this item.  
 */  
void set(final int intValue){  
    this.type = ClassWriter.INT;  
    this.intValue = intValue;  
    this.hashCode = 0x7FFFFFFF & (type + intValue);  
}  
  
/**  
 * Sets this item to a long item.  
 *  
 * @param longVal the value of this item.  
 */  
void set(final long longVal){  
    this.type = ClassWriter.LONG;  
    this.longVal = longVal;  
    this.hashCode = 0x7FFFFFFF & (type + (int) longVal);  
}  
  
/**  
 * Sets this item to a float item.  
 *  
 * @param floatVal the value of this item.  
 */  
void set(final float floatVal){  
    this.type = ClassWriter.FLOAT;  
    this.intValue = Float.floatToIntBits(floatVal);  
    this.hashCode = 0x7FFFFFFF & (type + (int) floatVal);  
}  
  
/**  
 * Sets this item to a double item.  
 */
```

```

*
* @param doubleVal the value of this item.
*/
void set(final double doubleVal){
    this.type = ClassWriter.DOUBLE;
    this.longVal = Double.doubleToRawLongBits(doubleVal);
    this.hashCode = 0x7FFFFFFF & (type + (int) doubleVal);
}

/**
 * Sets this item to an item that do not hold a primitive value.
 *
 * @param type      the type of this item.
 * @param strVal1  first part of the value of this item.
 * @param strVal2  second part of the value of this item.
 * @param strVal3  third part of the value of this item.
 */
void set(
    final int type,
    final String strVal1,
    final String strVal2,
    final String strVal3){
    this.type = type;
    this.strVal1 = strVal1;
    this.strVal2 = strVal2;
    this.strVal3 = strVal3;
    switch(type)
    {
        case ClassWriter.UTF8:
        case ClassWriter.STR:
        case ClassWriter.CLASS:
        case ClassWriter.TYPE_NORMAL:
            hashCode = 0x7FFFFFFF & (type + strVal1.hashCode());
            return;
        case ClassWriter.NAME_TYPE:
            hashCode = 0x7FFFFFFF & (type + strVal1.hashCode()
                                      * strVal2.hashCode());
            return;
        // ClassWriter.FIELD:
        // ClassWriter.METH:
        // ClassWriter.IMETH:
        default:
            hashCode = 0x7FFFFFFF &
                       (type + strVal1.hashCode()
                         * strVal2.hashCode() * strVal3.hashCode());
    }
}

/**
 * Indicates if the given item is equal to this one.

```

```

*
* @param i the item to be compared to this one.
* @return <tt>true</tt> if the given item is equal to this one,
*         <tt>false</tt> otherwise.
*/
boolean isEqualTo(final Item i){
    if(i.type == type)
    {
        switch(type)
        {
            case ClassWriter.INT:
            case ClassWriter.FLOAT:
                return i.intValue == intValue;
            case ClassWriter.TYPE_MERGED:
            case ClassWriter.LONG:
            case ClassWriter.DOUBLE:
                return i.longValue == longValue;
            case ClassWriter.UTF8:
            case ClassWriter.STR:
            case ClassWriter.CLASS:
            case ClassWriter.TYPE_NORMAL:
                return i.stringValue.equals(strVal1);
            case ClassWriter.TYPE_UNINIT:
                return i.intValue == intValue && i.stringValue.equals(strVal1);
            case ClassWriter.NAME_TYPE:
                return i.stringValue.equals(strVal1)
                    && i.stringValue.equals(strVal2);
            // ClassWriter.FIELD:
            // ClassWriter.METH:
            // ClassWriter.IMETH:
        default:
            return i.stringValue.equals(strVal1)
                && i.stringValue.equals(strVal2)
                && i.stringValue.equals(strVal3);
        }
    }
    return false;
}
}

```



7.15 Label.java

— Label.java —

\getchunk{France Telecom Copyright}

```
package clojure.asm;

/**
 * A label represents a position in the bytecode of a method. Labels
 * are used for jump, goto, and switch instructions, and for try
 * catch blocks.
 *
 * @author Eric Bruneton
 */
public class Label{

    /**
     * Indicates if this label is only used for debug attributes. Such a
     * label is not the start of a basic block, the target of a jump
     * instruction, or an exception handler. It can be safely ignored
     * in control flow graph analysis algorithms (for optimization
     * purposes).
     */
    final static int DEBUG = 1;

    /**
     * Indicates if the position of this label is known.
     */
    final static int RESOLVED = 2;

    /**
     * Indicates if this label has been updated, after instruction resizing.
     */
    final static int RESIZED = 4;

    /**
     * Indicates if this basic block has been pushed in the basic block
     * stack. See {@link MethodWriter#visitMaxs visitMaxs}.
     */
    final static int PUSHED = 8;

    /**
     * Indicates if this label is the target of a jump instruction, or
     * the start of an exception handler.
     */
    final static int TARGET = 16;

    /**
     * Indicates if a stack map frame must be stored for this label.
     */
    final static int STORE = 32;

    /**
     * Indicates if this label corresponds to a reachable basic block.
     */
}
```

```
final static int REACHABLE = 64;

/**
 * Indicates if this basic block ends with a JSR instruction.
 */
final static int JSR = 128;

/**
 * Indicates if this basic block ends with a RET instruction.
 */
final static int RET = 256;

/**
 * Field used to associate user information to a label.
 */
public Object info;

/**
 * Flags that indicate the status of this label.
 *
 * @see #DEBUG
 * @see #RESOLVED
 * @see #RESIZED
 * @see #PUSHED
 * @see #TARGET
 * @see #STORE
 * @see #REACHABLE
 * @see #JSR
 * @see #RET
 */
int status;

/**
 * The line number corresponding to this label, if known.
 */
int line;

/**
 * The position of this label in the code, if known.
 */
int position;

/**
 * Number of forward references to this label, times two.
 */
private int referenceCount;

/**
 * Informations about forward references. Each forward reference is
 * described by two consecutive integers in this array: the first one
```

```

* is the position of the first byte of the bytecode instruction that
* contains the forward reference, while the second is the position
* of the first byte of the forward reference itself. In fact the
* sign of the first integer indicates if this reference uses 2 or 4
* bytes, and its absolute value gives the position of the bytecode
* instruction.
*/
private int[] srcAndRefPositions;

// ----

/*
* Fields for the control flow and data flow graph analysis algorithms
* (used to compute the maximum stack size or the stack map frames).
* A control flow graph contains one node per "basic block", and one
* edge per "jump" from one basic block to another. Each node (i.e.,
* each basic block) is represented by the Label object that
* corresponds to the first instruction of this basic block. Each node
* also stores the list of its successors in the graph, as a linked
* list of Edge objects.
*
* The control flow analysis algorithms used to compute the maximum
* stack size or the stack map frames are similar and use two steps.
* The first step, during the visit of each instruction, builds
* information about the state of the local variables and the operand
* stack at the end of each basic block, called the "output frame",
* <i>relatively</i> to the frame state at the beginning of the basic
* block, which is called the "input frame", and which is <i>unknown</i>
* during this step. The second step, in
* {@link MethodWriter#visitMaxs}, is a fix point algorithm that
* computes information about the input frame of each basic block,
* from the input state of the first basic block (known from the
* method signature), and by the using the previously computed
* relative output frames.
*
* The algorithm used to compute the maximum stack size only computes
* the relative output and absolute input stack heights, while the
* algorithm used to compute stack map frames computes relative
* output frames and absolute input frames.
*/

```

/**

- * Start of the output stack relatively to the input stack. The exact
- * semantics of this field depends on the algorithm that is used.
- * <p/>
- * When only the maximum stack size is computed, this field is the
- * number of elements in the input stack.
- * <p/>
- * When the stack map frames are completely computed, this field is
- * the offset of the first output stack element relatively to the top

```
* of the input stack. This offset is always negative or null. A
* null offset means that the output stack must be appended to the
* input stack. A -n offset means that the first n output stack
* elements must replace the top n input stack elements, and that
* the other elements must be appended to the input stack.
*/
int inputStackTop;

/**
 * Maximum height reached by the output stack, relatively to the top
 * of the input stack. This maximum is always positive or null.
 */
int outputStackMax;

/**
 * Information about the input and output stack map frames of this
 * basic block. This field is only used when
 * {@link ClassWriter#COMPUTE_FRAMES} option is used.
 */
Frame frame;

/**
 * The successor of this label, in the order they are visited. This
 * linked list does not include labels used for debug info only. If
 * {@link ClassWriter#COMPUTE_FRAMES} option is used then, in addition,
 * it does not contain successive labels that denote the same
 * bytecode position (in this case only the first label appears in
 * this list).
 */
Label successor;

/**
 * The successors of this node in the control flow graph. These
 * successors are stored in a linked list of {@link Edge Edge}
 * objects, linked to each other by their {@link Edge#next} field.
 */
Edge successors;

/**
 * The next basic block in the basic block stack. This stack is used
 * in the main loop of the fix point algorithm used in the second step
 * of the control flow analysis algorithms.
 *
 * @see MethodWriter#visitMaxs
 */
Label next;

// -----
// Constructor
// -----
```

```

/**
 * Constructs a new label.
 */
public Label(){
}

/**
 * Constructs a new label.
 *
 * @param debug if this label is only used for debug attributes.
 */
Label(final boolean debug){
    this.status = debug ? DEBUG : 0;
}

// -----
// Methods to compute offsets and to manage forward references
// -----

/**
 * Returns the offset corresponding to this label. This offset is
 * computed from the start of the method's bytecode. <i>This method
 * is intended for {@link Attribute} sub classes, and is normally
 * not needed by class generators or adapters.</i>
 *
 * @return the offset corresponding to this label.
 * @throws IllegalStateException if this label is not resolved yet.
 */
public int getOffset(){
    if((status & RESOLVED) == 0)
    {
        throw new IllegalStateException(
            "Label offset position has not been resolved yet");
    }
    return position;
}

/**
 * Puts a reference to this label in the bytecode of a method. If the
 * position of the label is known, the offset is computed and written
 * directly. Otherwise, a null offset is written and a new forward
 * reference is declared for this label.
 *
 * @param owner      the code writer that calls this method.
 * @param out        the bytecode of the method.
 * @param source     the position of first byte of the bytecode
 *                   instruction that contains this label.
 * @param wideOffset <tt>true</tt> if the reference must be stored in 4
 *                   bytes, or <tt>false</tt> if it must be stored with 2

```

```

*
*           bytes.
* @throws IllegalArgumentException if this label has not been created by
*                               the given code writer.
*/
void put(
    final MethodWriter owner,
    final ByteVector out,
    final int source,
    final boolean wideOffset){
if((status & RESOLVED) != 0)
{
    if(wideOffset)
    {
        out.putInt(position - source);
    }
    else
    {
        out.putShort(position - source);
    }
}
else
{
    if(wideOffset)
    {
        addReference(-1 - source, out.length);
        out.putInt(-1);
    }
    else
    {
        addReference(source, out.length);
        out.putShort(-1);
    }
}
}

/**
 * Adds a forward reference to this label. This method must be called
 * only for a true forward reference, i.e. only if this label is not
 * resolved yet. For backward references, the offset of the reference
 * can be, and must be, computed and stored directly.
 *
 * @param sourcePosition      the position of the referencing instruction.
 *                           This position will be used to compute the
 *                           offset of this forward reference.
 * @param referencePosition   the position where the offset for this
 *                           forward reference must be stored.
 */
private void addReference(
    final int sourcePosition,
    final int referencePosition){

```

```

if(srcAndRefPositions == null)
{
    srcAndRefPositions = new int[6];
}
if(referenceCount >= srcAndRefPositions.length)
{
    int[] a = new int[srcAndRefPositions.length + 6];
    System.arraycopy(srcAndRefPositions,
                    0,
                    a,
                    0,
                    srcAndRefPositions.length);
    srcAndRefPositions = a;
}
srcAndRefPositions[referenceCount++] = sourcePosition;
srcAndRefPositions[referenceCount++] = referencePosition;
}

/**
 * Resolves all forward references to this label. This method must be
 * called when this label is added to the bytecode of the method,
 * i.e. when its position becomes known. This method fills in the
 * blanks that were left in the bytecode by each forward reference
 * previously added to this label.
 *
 * @param owner    the code writer that calls this method.
 * @param position the position of this label in the bytecode.
 * @param data     the bytecode of the method.
 * @return <tt>true</tt> if a blank that was left for this label was
 *         too small to store the offset. In such a case the
 *         corresponding jump instruction is replaced with a pseudo
 *         instruction (using unused opcodes) using an unsigned two
 *         bytes offset. These pseudo instructions will need to be
 *         replaced with true instructions with wider offsets (4 bytes
 *         instead of 2). This is done in
 *         {@link MethodWriter#resizeInstructions}.
 * @throws IllegalArgumentException if this label has already been
 *                                 resolved, or if it has not been
 *                                 created by the given code writer.
 */
boolean resolve(
    final MethodWriter owner,
    final int position,
    final byte[] data){
boolean needUpdate = false;
this.status != RESOLVED;
this.position = position;
int i = 0;
while(i < referenceCount)
{

```

```

int source = srcAndRefPositions[i++];
int reference = srcAndRefPositions[i++];
int offset;
if(source >= 0)
{
    offset = position - source;
    if(offset < Short.MIN_VALUE || offset > Short.MAX_VALUE)
    {
        /*
         * changes the opcode of the jump instruction, in order
         * to be able to find it later (see resizeInstructions
         * in MethodWriter). These temporary opcodes are similar
         * to jump instruction opcodes, except that the 2 bytes
         * offset is unsigned (and can therefore represent
         * values from 0 to 65535, which is sufficient since
         * the size of a method is limited to 65535 bytes).
        */
        int opcode = data[reference - 1] & 0xFF;
        if(opcode <= Opcodes.JSR)
        {
            // changes IFEQ ... JSR to opcodes 202 to 217
            data[reference - 1] = (byte) (opcode + 49);
        }
        else
        {
            // changes IFNULL and IFNONNULL to opcodes 218
            // and 219
            data[reference - 1] = (byte) (opcode + 20);
        }
        needUpdate = true;
    }
    data[reference++] = (byte) (offset >>> 8);
    data[reference] = (byte) offset;
}
else
{
    offset = position + source + 1;
    data[reference++] = (byte) (offset >>> 24);
    data[reference++] = (byte) (offset >>> 16);
    data[reference++] = (byte) (offset >>> 8);
    data[reference] = (byte) offset;
}
}
return needUpdate;
}

/**
 * Returns the first label of the series to which this label belongs.
 * For an isolated label or for the first label in a series of
 * successive labels, this method returns the label itself. For other

```

```

    * labels it returns the first label of the series.
    *
    * @return the first label of the series to which this label belongs.
    */
Label getFirst(){
    return frame == null ? this : frame.owner;
}

// -----
// Overriden Object methods
// -----


/***
 * Returns a string representation of this label.
 *
 * @return a string representation of this label.
 */
public String toString(){
    return "L" + System.identityHashCode(this);
}
}

```

—————

7.16 MethodAdapter.java

```

(MethodVisitor [317])
— MethodAdapter.java —

\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * An empty {@link MethodVisitor} that delegates to another
 * {@link MethodVisitor}. This class can be used as a super class to
 * quickly implement useful method adapter classes, just by overriding
 * the necessary methods.
 *
 * @author Eric Bruneton
 */
public class MethodAdapter implements MethodVisitor{

/**
 * The {@link MethodVisitor} to which this adapter delegates calls.
 */
protected MethodVisitor mv;

/**

```

```
* Constructs a new {@link MethodAdapter} object.
*
* @param mv the code visitor to which this adapter must delegate calls.
*/
public MethodAdapter(final MethodVisitor mv){
    this.mv = mv;
}

public AnnotationVisitor visitAnnotationDefault(){
    return mv.visitAnnotationDefault();
}

public AnnotationVisitor visitAnnotation(
    final String desc,
    final boolean visible){
    return mv.visitAnnotation(desc, visible);
}

public AnnotationVisitor visitParameterAnnotation(
    final int parameter,
    final String desc,
    final boolean visible){
    return mv.visitParameterAnnotation(parameter, desc, visible);
}

public void visitAttribute(final Attribute attr){
    mv.visitAttribute(attr);
}

public void visitCode(){
    mv.visitCode();
}

public void visitFrame(
    final int type,
    final int nLocal,
    final Object[] local,
    final int nStack,
    final Object[] stack){
    mv.visitFrame(type, nLocal, local, nStack, stack);
}

public void visitInsn(final int opcode){
    mv.visitInsn(opcode);
}

public void visitIntInsn(final int opcode, final int operand){
    mv.visitIntInsn(opcode, operand);
}
```

```
public void visitVarInsn(final int opcode, final int var){
    mv.visitVarInsn(opcode, var);
}

public void visitTypeInsn(final int opcode, final String desc){
    mv.visitTypeInsn(opcode, desc);
}

public void visitFieldInsn(
    final int opcode,
    final String owner,
    final String name,
    final String desc){
    mv.visitFieldInsn(opcode, owner, name, desc);
}

public void visitMethodInsn(
    final int opcode,
    final String owner,
    final String name,
    final String desc){
    mv.visitMethodInsn(opcode, owner, name, desc);
}

public void visitJumpInsn(final int opcode, final Label label){
    mv.visitJumpInsn(opcode, label);
}

public void visitLabel(final Label label){
    mv.visitLabel(label);
}

public void visitLdcInsn(final Object cst){
    mv.visitLdcInsn(cst);
}

public void visitIincInsn(final int var, final int increment){
    mv.visitIincInsn(var, increment);
}

public void visitTableSwitchInsn(
    final int min,
    final int max,
    final Label dflt,
    final Label labels[]){
    mv.visitTableSwitchInsn(min, max, dflt, labels);
}

public void visitLookupSwitchInsn(
    final Label dflt,
```

```
    final int keys[],
    final Label labels[]){
    mv.visitLookupSwitchInsn(dfilt, keys, labels);
}

public void visitMultiANewArrayInsn(final String desc, final int dims){
    mv.visitMultiANewArrayInsn(desc, dims);
}

public void visitTryCatchBlock(
    final Label start,
    final Label end,
    final Label handler,
    final String type){
    mv.visitTryCatchBlock(start, end, handler, type);
}

public void visitLocalVariable(
    final String name,
    final String desc,
    final String signature,
    final Label start,
    final Label end,
    final int index){
    mv.visitLocalVariable(name, desc, signature, start, end, index);
}

public void visitLineNumber(final int line, final Label start){
    mv.visitLineNumber(line, start);
}

public void visitMaxs(final int maxStack, final int maxLocals){
    mv.visitMaxs(maxStack, maxLocals);
}

public void visitEnd(){
    mv.visitEnd();
}
```



7.17 MethodVisitor.java

— MethodVisitor.java —

\getchunk{France Telecom Copyright}

```

package clojure.asm;

/**
 * A visitor to visit a Java method. The methods of this interface must
 * be called in the following order:
 * [ <tt>visitAnnotationDefault</tt> ] ( * <tt>visitAnnotation</tt> |
 * <tt>visitParameterAnnotation</tt> | * <tt>visitAttribute</tt> )*
 * [ <tt>visitCode</tt> ( <tt>visitFrame</tt> |
 * <tt>visit</tt><i>X</i>Insn</tt> | <tt>visitLabel</tt> |
 * <tt>visitTryCatchBlock</tt> | * <tt>visitLocalVariable</tt> |
 * <tt>visitLineNumber</tt>)* <tt>visitMaxs</tt> ]
 * <tt>visitEnd</tt>. In addition, the <tt>visit</tt><i>X</i>Insn</tt>
 * and <tt>visitLabel</tt> methods must be called in the sequential
 * order of the bytecode instructions of the visited code,
 * <tt>visitTryCatchBlock</tt> must be called <i>before</i> the
 * labels passed as arguments have been visited, and the
 * <tt>visitLocalVariable</tt> and <tt>visitLineNumber</tt> methods
 * must be called <i>after</i> the labels passed as arguments have
 * been visited.
 *
 * @author Eric Bruneton
 */
public interface MethodVisitor{

// -----
// Annotations and non standard attributes
// -----


/**
 * Visits the default value of this annotation interface method.
 *
 * @return a visitor to the visit the actual default value of this
 *         annotation interface method, or <tt>null</tt> if this
 *         visitor is not interested in visiting this default value.
 *         The 'name' parameters passed to the methods of this
 *         annotation visitor are ignored. Moreover, exactly one visit
 *         method must be called on this annotation visitor, followed
 *         by visitEnd.
 */
AnnotationVisitor visitAnnotationDefault();

/**
 * Visits an annotation of this method.
 *
 * @param desc    the class descriptor of the annotation class.
 * @param visible <tt>true</tt> if the annotation is visible at runtime.
 * @return a visitor to visit the annotation values, or <tt>null</tt> if
 *         this visitor is not interested in visiting this annotation.
 */
AnnotationVisitor visitAnnotation(String desc, boolean visible);
}

```

```

/**
 * Visits an annotation of a parameter this method.
 *
 * @param parameter the parameter index.
 * @param desc      the class descriptor of the annotation class.
 * @param visible   <tt>true</tt> if the annotation is visible at
 *                  runtime.
 * @return a visitor to visit the annotation values, or <tt>null</tt>
 *         if this visitor is not interested in visiting this annotation.
 */
AnnotationVisitor visitParameterAnnotation(
    int parameter,
    String desc,
    boolean visible);

/**
 * Visits a non standard attribute of this method.
 *
 * @param attr an attribute.
 */
void visitAttribute(Attribute attr);

/**
 * Starts the visit of the method's code, if any (i.e. non abstract
 * method).
 */
void visitCode();

/**
 * Visits the current state of the local variables and operand stack
 * elements. This method must(*) be called <i>just before</i> any
 * instruction <b>i</b> that follows an unconditionnal branch
 * instruction such as GOTO or THROW, that is the target of a jump
 * instruction, or that starts an exception handler block. The visited
 * types must describe the values of the local variables and of the
 * operand stack elements <i>just before</i> <b>i</b> is executed.
 * <br> <br> (*) this is mandatory only for classes whose version
 * is greater than or equal to {@link Opcodes#V1_6 V1_6}. <br> <br>
 * Packed frames are basically "deltas" from the state of the
 * previous frame (very first frame is implicitly defined by the
 * method's parameters and access flags): <ul> <li>
 * {@link Opcodes#F_SAME} representing frame with exactly the same
 * locals as the previous frame and with the empty stack.</li>
 * <li>{@link Opcodes#F_SAME1}
 * representing frame with exactly the same locals as the previous
 * frame and with single value on the stack (<code>nStack</code> is 1
 * and <code>stack[0]</code> contains value for the type of the
 * stack item).</li> <li>{@link Opcodes#F_APPEND} representing frame
 * with current locals are the same as the locals in the previous

```

```

* frame, except that additional locals are defined
* (<code>nLocal</code> is 1, 2 or 3 and * <code>local</code> elements
* contains values representing added types).</li>
* <li>{@link Opcodes#F_CHOP} representing frame with current locals
* are the same as the locals in the previous frame, except that the
* last 1-3 locals are absent and with the empty stack
* (<code>nLocals</code> is 1, 2 or 3). </li> <li>
* {@link Opcodes#F_FULL} representing complete frame
* data.</li> </li> </ul>
*
* @param type    the type of this stack map frame. Must be
*                 {@link Opcodes#F_NEW} for expanded frames, or
*                 {@link Opcodes#F_FULL}, {@link Opcodes#F_APPEND},
*                 {@link Opcodes#F_CHOP}, {@link Opcodes#F_SAME} or
*                 {@link Opcodes#F_APPEND}, {@link Opcodes#F_SAME1}
*                 for compressed frames.
* @param nLocal  the number of local variables in the visited frame.
* @param local   the local variable types in this frame. This array must
*                 not be modified. Primitive types are represented by
*                 {@link Opcodes#TOP}, {@link Opcodes#INTEGER},
*                 {@link Opcodes#FLOAT}, {@link Opcodes#LONG},
*                 {@link Opcodes#DOUBLE}, {@link Opcodes#NULL} or
*                 {@link Opcodes#UNINITIALIZED_THIS} (long and double are
*                 represented by a single element). Reference types are
*                 represented by String objects (representing internal
*                 names, or type descriptors for array types), and
*                 uninitialized types by Label objects (this label
*                 designates the NEW instruction that created this
*                 uninitialized value).
* @param nStack   the number of operand stack elements in the visited
*                 frame.
* @param stack   the operand stack types in this frame. This array must
*                 not be modified. Its content has the same format as
*                 the "local" array.
*/
void visitFrame(
    int type,
    int nLocal,
    Object[] local,
    int nStack,
    Object[] stack);

// -----
// Normal instructions
// -----


/***
* Visits a zero operand instruction.
*
* @param opcode the opcode of the instruction to be visited. This

```

```

*           opcode is either NOP, ACONST_NULL, ICONST_M1, ICONST_0,
*           ICONST_1, ICONST_2, ICONST_3, ICONST_4, ICONST_5,
*           LCONST_0, LCONST_1, FCONST_0, FCONST_1, FCONST_2,
*           DCONST_0, DCONST_1, IALOAD, LALOAD, FALOAD,
*           DALOAD, AALOAD, BALOAD, CALOAD, SALOAD, IASTORE,
*           LASTORE, FASTORE, DASTORE, AASTORE, BASTORE, CASTORE,
*           SASTORE, POP, POP2, DUP, DUP_X1, DUP_X2, DUP2, DUP2_X1,
*           DUP2_X2, SWAP, IADD, LADD, FADD, DADD, ISUB, LSUB, FSUB,
*           DSUB, IMUL, LMUL, FMUL, DMUL, IDIV, LDIV, FDIV, DDIV,
*           IREM, LREM, FREM, DREM, INEG, LNEG, FNEG, DNEG, ISHL,
*           LSHL, ISHR, LSHR, IUSHR, LUSHR, IAND, LAND, IOR, LOR,
*           IXOR, LXOR, I2L, I2F, I2D, L2I, L2F, L2D, F2I, F2L,
*           F2D, D2I, D2L, D2F, I2B, I2C, I2S, LCMP, FCMPL, FCMPG,
*           DCMPL, DCMPG, IRETURN, LRETURN, FRETURN, DRETURN,
*           ARETURN, RETURN, ARRAYLENGTH, ATHROW, MONITORENTER,
*           or MONITOREXIT.
*/
void visitInsn(int opcode);

/**
 * Visits an instruction with a single int operand.
 *
 * @param opcode the opcode of the instruction to be visited. This
 *               opcode is either BIPUSH, SIPUSH or NEWARRAY.
 * @param operand the operand of the instruction to be visited.<br> When
 *               opcode is BIPUSH, operand value should be between
 *               Byte.MIN_VALUE and Byte.MAX_VALUE.<br> When opcode
 *               is SIPUSH, operand value should be between
 *               Short.MIN_VALUE and Short.MAX_VALUE.<br> When opcode
 *               is NEWARRAY, operand value should be one of
 *               {@link Opcodes#T_BOOLEAN}, {@link Opcodes#T_CHAR},
 *               {@link Opcodes#T_FLOAT}, {@link Opcodes#T_DOUBLE},
 *               {@link Opcodes#T_BYTE}, {@link Opcodes#T_SHORT},
 *               {@link Opcodes#T_INT} or {@link Opcodes#T_LONG}.
 */
void visitIntInsn(int opcode, int operand);

/**
 * Visits a local variable instruction. A local variable instruction is
 * an instruction that loads or stores the value of a local variable.
 *
 * @param opcode the opcode of the local variable instruction to be
 *               visited. This opcode is either ILOAD, LLOAD, FLOAD,
 *               DLOAD, ALOAD, ISTORE, LSTORE, FSTORE, DSTORE, ASTORE
 *               or RET.
 * @param var    the operand of the instruction to be visited. This
 *               operand is the index of a local variable.
 */
void visitVarInsn(int opcode, int var);

```

```

/**
 * Visits a type instruction. A type instruction is an instruction that
 * takes a type descriptor as parameter.
 *
 * @param opcode the opcode of the type instruction to be visited.
 *               This opcode is either NEW, ANEWARRAY, CHECKCAST or
 *               INSTANCEOF.
 * @param desc   the operand of the instruction to be visited. This
 *               operand is must be a fully qualified class name in
 *               internal form, or the type descriptor of an array
 *               type (see {@link Type Type}).
 */
void visitTypeInsn(int opcode, String desc);

/**
 * Visits a field instruction. A field instruction is an instruction
 * that loads or stores the value of a field of an object.
 *
 * @param opcode the opcode of the type instruction to be visited.
 *               This opcode is either GETSTATIC, PUTSTATIC, GETFIELD
 *               or PUTFIELD.
 * @param owner  the internal name of the field's owner class (see {@link
 *               Type#getInternalName() getInternalName}).
 * @param name   the field's name.
 * @param desc   the field's descriptor (see {@link Type Type}).
 */
void visitFieldInsn(int opcode, String owner, String name, String desc);

/**
 * Visits a method instruction. A method instruction is an instruction
 * that invokes a method.
 *
 * @param opcode the opcode of the type instruction to be visited.
 *               This opcode is either INVOKEVIRTUAL, INVOKESPECIAL,
 *               INVOKESTATIC or INVOKEINTERFACE.
 * @param owner  the internal name of the method's owner class (see
 *               {@link Type#getInternalName() getInternalName}).
 * @param name   the method's name.
 * @param desc   the method's descriptor (see {@link Type Type}).
 */
void visitMethodInsn(int opcode, String owner, String name, String desc);

/**
 * Visits a jump instruction. A jump instruction is an instruction that
 * may jump to another instruction.
 *
 * @param opcode the opcode of the type instruction to be visited. This
 *               opcode is either IFEQ, IFNE, IFLT, IFGE, IFGT, IFLE,
 *               IF_ICMPEQ, IF_ICMPNE, IF_ICMPLT, IF_ICMPGE, IF_ICMPGT,
 *               IF_ICMPLE, IF_ACMPEQ, IF_ACMPNE, GOTO, JSR, IFNULL or

```

```

*
*           IFNONNULL.
* @param label  the operand of the instruction to be visited. This
*               operand is a label that designates the instruction
*               to which the jump instruction may jump.
*/
void visitJumpInsn(int opcode, Label label);

/**
* Visits a label. A label designates the instruction that will be
* visited just after it.
*
* @param label a {@link Label} object.
*/
void visitLabel(Label label);

// -----
// Special instructions
// -----


/** 
* Visits a LDC instruction.
*
* @param cst the constant to be loaded on the stack. This parameter
*             must be a non null {@link Integer}, a {@link Float},
*             a {@link Long}, a {@link Double} a {@link String} (or
*             a {@link Type} for <tt>.class</tt> constants, for classes
*             whose version is 49.0 or more).
*/
void visitLdcInsn(Object cst);

/** 
* Visits an IINC instruction.
*
* @param var      index of the local variable to be incremented.
* @param increment amount to increment the local variable by.
*/
void visitIincInsn(int var, int increment);

/** 
* Visits a TABLESWITCH instruction.
*
* @param min      the minimum key value.
* @param max      the maximum key value.
* @param dflt     beginning of the default handler block.
* @param labels   beginnings of the handler blocks. <tt>labels[i]</tt>
*                 is the beginning of the handler block for the
*                 <tt>min + i</tt> key.
*/
void visitTableSwitchInsn(int min, int max, Label dflt, Label labels[]);

```

```

/**
 * Visits a LOOKUPSWITCH instruction.
 *
 * @param dflt    beginning of the default handler block.
 * @param keys   the values of the keys.
 * @param labels beginnings of the handler blocks. <tt>labels[i]</tt>
 *                  is the beginning of the handler block for the
 *                  <tt>keys[i]</tt> key.
 */
void visitLookupSwitchInsn(Label dflt, int keys[], Label labels[]);

/**
 * Visits a MULTIANEWARRAY instruction.
 *
 * @param desc an array type descriptor (see {@link Type Type}).
 * @param dims number of dimensions of the array to allocate.
 */
void visitMultiANewArrayInsn(String desc, int dims);

// -----
// Exceptions table entries, debug information, max stack and max
// locals
// -----

/**
 * Visits a try catch block.
 *
 * @param start    beginning of the exception handler's scope (inclusive).
 * @param end      end of the exception handler's scope (exclusive).
 * @param handler beginning of the exception handler's code.
 * @param type     internal name of the type of exceptions handled by the
 *                 handler, or <tt>null</tt> to catch any exceptions
 *                 (for "finally" blocks).
 * @throws IllegalArgumentException if one of the labels has already
 *                                 been visited by this visitor (by the
 *                                 {@link #visitLabel visitLabel} method).
 */
void visitTryCatchBlock(Label start,
                       Label end,
                       Label handler,
                       String type);

/**
 * Visits a local variable declaration.
 *
 * @param name      the name of a local variable.
 * @param desc      the type descriptor of this local variable.
 * @param signature the type signature of this local variable. May be
 *                  <tt>null</tt> if the local variable type does not
 *                  use generic types.

```

```
* @param start      the first instruction corresponding to the scope of
*                   this local variable (inclusive).
* @param end        the last instruction corresponding to the scope of
*                   this local variable (exclusive).
* @param index      the local variable's index.
* @throws IllegalArgumentException if one of the labels has not already
*                                   been visited by this visitor (by the
*                                   {@link #visitLabel visitLabel} method).
*/
void visitLocalVariable(
    String name,
    String desc,
    String signature,
    Label start,
    Label end,
    int index);

/**
 * Visits a line number declaration.
 *
 * @param line  a line number. This number refers to the source file
 *              from which the class was compiled.
 * @param start the first instruction corresponding to this line number.
 * @throws IllegalArgumentException if <tt>start</tt> has not already
 *                                   been visited by this visitor (by the
 *                                   {@link #visitLabel visitLabel} method).
 */
void visitLineNumber(int line, Label start);

/**
 * Visits the maximum stack size and the maximum number of local
 * variables of the method.
 *
 * @param maxStack  maximum stack size of the method.
 * @param maxLocals maximum number of local variables for the method.
 */
void visitMaxs(int maxStack, int maxLocals);

/**
 * Visits the end of the method. This method, which is the last one to
 * be called, is used to inform the visitor that all the annotations and
 * attributes of the method have been visited.
 */
void visitEnd();
}
```

7.18 MethodWriter.java

```
(MethodVisitor [317])
— MethodWriter.java —

\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * A {@link MethodVisitor} that generates methods in bytecode form.
 * Each visit method of this class appends the bytecode corresponding
 * to the visited instruction to a byte vector, in the order these
 * methods are called.
 *
 * @author Eric Bruneton
 * @author Eugene Kuleshov
 */
class MethodWriter implements MethodVisitor {

    /**
     * Pseudo access flag used to denote constructors.
     */
    final static int ACC_CONSTRUCTOR = 262144;

    /**
     * Frame has exactly the same locals as the previous stack map frame
     * and number of stack items is zero.
     */
    final static int SAME_FRAME = 0; // to 63 (0-3f)

    /**
     * Frame has exactly the same locals as the previous stack map frame
     * and number of stack items is 1
     */
    final static int SAME_LOCALS_1_STACK_ITEM_FRAME = 64; // to 127 (40-7f)

    /**
     * Reserved for future use
     */
    final static int RESERVED = 128;

    /**
     * Frame has exactly the same locals as the previous stack map frame
     * and number of stack items is 1. Offset is bigger then 63;
     */
    final static int SAME_LOCALS_1_STACK_ITEM_FRAME_EXTENDED = 247; // f7

    /**
     * Frame where current locals are the same as the locals in the previous

```

```
* frame, except that the k last locals are absent. The value of k is
* given by the formula 251-frame_type.
*/
final static int CHOP_FRAME = 248; // to 250 (f8-fA)

/**
 * Frame has exactly the same locals as the previous stack map frame
 * and number of stack items is zero. Offset is bigger then 63;
 */
final static int SAME_FRAME_EXTENDED = 251; // fb

/**
 * Frame where current locals are the same as the locals in the
 * previous frame, except that k additional locals are defined.
 * The value of k is given by the formula frame_type-251.
 */
final static int APPEND_FRAME = 252; // to 254 // fc-fe

/**
 * Full frame
 */
final static int FULL_FRAME = 255; // ff

/**
 * Indicates that the stack map frames must be recomputed from scratch.
 * In this case the maximum stack size and number of local variables
 * is also recomputed from scratch.
 *
 * @see #compute
 */
private final static int FRAMES = 0;

/**
 * Indicates that the maximum stack size and number of local variables
 * must be automatically computed.
 *
 * @see #compute
 */
private final static int MAXS = 1;

/**
 * Indicates that nothing must be automatically computed.
 *
 * @see #compute
 */
private final static int NOTHING = 2;

/**
 * Next method writer (see {@link ClassWriter#firstMethod}).
 */
```

```
MethodWriter next;

/**
 * The class writer to which this method must be added.
 */
ClassWriter cw;

/**
 * Access flags of this method.
 */
private int access;

/**
 * The index of the constant pool item that contains the name of this
 * method.
 */
private int name;

/**
 * The index of the constant pool item that contains the descriptor
 * of this method.
 */
private int desc;

/**
 * The descriptor of this method.
 */
private String descriptor;

/**
 * The signature of this method.
 */
String signature;

/**
 * If not zero, indicates that the code of this method must be copied
 * from the ClassReader associated to this writer in
 * <code>cw.cr</code>. More precisely, this field gives the index of
 * the first byte to copied from <code>cw.cr.b</code>.
 */
int classReaderOffset;

/**
 * If not zero, indicates that the code of this method must be
 * copied from the ClassReader associated to this writer in
 * <code>cw.cr</code>. More precisely, this field gives the number
 * of bytes to copied from <code>cw.cr.b</code>.
 */
int classReaderLength;
```

```
/**  
 * Number of exceptions that can be thrown by this method.  
 */  
int exceptionCount;  
  
/**  
 * The exceptions that can be thrown by this method. More precisely,  
 * this array contains the indexes of the constant pool items that  
 * contain the internal names of these exception classes.  
 */  
int[] exceptions;  
  
/**  
 * The annotation default attribute of this method. May be  
 * <tt>null</tt>.  
 */  
private ByteVector annd;  
  
/**  
 * The runtime visible annotations of this method. May be  
 * <tt>null</tt>.  
 */  
private AnnotationWriter anns;  
  
/**  
 * The runtime invisible annotations of this method. May be  
 * <tt>null</tt>.  
 */  
private AnnotationWriter ianns;  
  
/**  
 * The runtime visible parameter annotations of this method. May be  
 * <tt>null</tt>.  
 */  
private AnnotationWriter[] panns;  
  
/**  
 * The runtime invisible parameter annotations of this method. May be  
 * <tt>null</tt>.  
 */  
private AnnotationWriter[] ipanns;  
  
/**  
 * The non standard attributes of the method.  
 */  
private Attribute attrs;  
  
/**  
 * The bytecode of this method.  
 */
```

```
private ByteVector code = new ByteVector();

/**
 * Maximum stack size of this method.
 */
private int maxStack;

/**
 * Maximum number of local variables for this method.
 */
private int maxLocals;

/**
 * Number of stack map frames in the StackMapTable attribute.
 */
private int frameCount;

/**
 * The StackMapTable attribute.
 */
private ByteVector stackMap;

/**
 * The offset of the last frame that was written in the StackMapTable
 * attribute.
 */
private int previousFrameOffset;

/**
 * The last frame that was written in the StackMapTable attribute.
 *
 * @see #frame
 */
private int[] previousFrame;

/**
 * Index of the next element to be added in {@link #frame}.
 */
private int frameIndex;

/**
 * The current stack map frame. The first element contains the offset
 * of the instruction to which the frame corresponds, the second
 * element is the number of locals and the third one is the number
 * of stack elements. The local variables start at index 3 and are
 * followed by the operand stack values. In summary frame[0] = offset,
 * frame[1] = nLocal, frame[2] = nStack, frame[3] = nLocal. All types
 * are encoded as integers, with the same format as the one used in
 * {@link Label}, but limited to BASE types.
 */
```

```
private int[] frame;

/**
 * Number of elements in the exception handler list.
 */
private int handlerCount;

/**
 * The first element in the exception handler list.
 */
private Handler firstHandler;

/**
 * The last element in the exception handler list.
 */
private Handler lastHandler;

/**
 * Number of entries in the LocalVariableTable attribute.
 */
private int localVarCount;

/**
 * The LocalVariableTable attribute.
 */
private ByteVector localVar;

/**
 * Number of entries in the LocalVariableTypeTable attribute.
 */
private int localVarTypeCount;

/**
 * The LocalVariableTypeTable attribute.
 */
private ByteVector localVarType;

/**
 * Number of entries in the LineNumberTable attribute.
 */
private int lineNumberCount;

/**
 * The LineNumberTable attribute.
 */
private ByteVector lineNumber;

/**
 * The non standard attributes of the method's code.
 */
```

```
private Attribute cattrs;

< /**
 * Indicates if some jump instructions are too small and need to be
 * resized.
 */
private boolean resize;

< /**
 * Indicates if the instructions contain at least one JSR instruction.
 */
private boolean jsr;

// ----

/*
 * Fields for the control flow graph analysis algorithm (used to
 * compute the maximum stack size). A control flow graph contains
 * one node per "basic block", and one edge per "jump" from one basic
 * block to another. Each node (i.e., each basic block) is represented
 * by the Label object that corresponds to the first instruction of
 * this basic block. Each node also stores the list of its successors
 * in the graph, as a linked list of Edge objects.
 */

/** 
 * Indicates what must be automatically computed.
 *
 * @see FRAMES
 * @see MAXS
 * @see NOTHING
 */
private int compute;

< /**
 * A list of labels. This list is the list of basic blocks in the
 * method, i.e. a list of Label objects linked to each other by their
 * {@link Label#successor} field, in the order they are visited by
 * {@link visitLabel}, and starting with the first basic block.
 */
private Label labels;

< /**
 * The previous basic block.
 */
private Label previousBlock;

< /**
 * The current basic block.
 */

```

```

private Label currentBlock;

<�新
 * The (relative) stack size after the last visited instruction. This
 * size is relative to the beginning of the current basic block, i.e.,
 * the true stack size after the last visited instruction is equal to
 * the {@link Label#inputStackTop beginStackSize} of the current basic
 * block plus <tt>stackSize</tt>.
 */
private int stackSize;

<�新
 * The (relative) maximum stack size after the last visited instruction.
 * This size is relative to the beginning of the current basic block,
 * i.e., the true maximum stack size after the last visited instruction
 * is equal to the {@link Label#inputStackTop beginStackSize} of the
 * current basic block plus <tt>stackSize</tt>.
 */
private int maxStackSize;

// -----
// Constructor
// -----


<�新
 * Constructs a new {@link MethodWriter}.
 *
 * @param cw           the class writer in which the method must be
 *                     added.
 * @param access       the method's access flags (see {@link Opcodes}).
 * @param name         the method's name.
 * @param desc         the method's descriptor (see {@link Type}).
 * @param signature    the method's signature. May be <tt>null</tt>.
 * @param exceptions  the internal names of the method's exceptions.
 *                     May be <tt>null</tt>.
 * @param computeMaxs <tt>true</tt> if the maximum stack size and
 *                     number of local variables must be automatically
 *                     computed.
 * @param computeFrames <tt>true</tt> if the stack map tables must be
 *                     recomputed from scratch.
 */
MethodWriter(
    final ClassWriter cw,
    final int access,
    final String name,
    final String desc,
    final String signature,
    final String[] exceptions,
    final boolean computeMaxs,
    final boolean computeFrames){
```

```

if(cw.firstMethod == null)
{
    cw.firstMethod = this;
}
else
{
    cw.lastMethod.next = this;
}
cw.lastMethod = this;
this.cw = cw;
this.access = access;
this.name = cw.newUTF8(name);
this.desc = cw.newUTF8(desc);
this.descriptor = desc;
this.signature = signature;
if(exceptions != null && exceptions.length > 0)
{
    exceptionCount = exceptions.length;
    this.exceptions = new int[exceptionCount];
    for(int i = 0; i < exceptionCount; ++i)
    {
        this.exceptions[i] = cw.newClass(exceptions[i]);
    }
}
this.compute =
    computeFrames ? FRAMES : (computeMaxs ? MAXS : NOTHING);
if(computeMaxs || computeFrames)
{
    if(computeFrames && name.equals("<init>"))
    {
        this.access |= ACC_CONSTRUCTOR;
    }
    // updates maxLocals
    int size = getArgumentsAndReturnSizes(descriptor) >> 2;
    if((access & Opcodes.ACC_STATIC) != 0)
    {
        --size;
    }
    maxLocals = size;
    // creates and visits the label for the first basic block
    labels = new Label();
    labels.status |= Label.PUSHED;
    visitLabel(labels);
}
}

// -----
// Implementation of the MethodVisitor interface
// -----

```

```
public AnnotationVisitor visitAnnotationDefault(){
    annd = new ByteVector();
    return new AnnotationWriter(cw, false, annd, null, 0);
}

public AnnotationVisitor visitAnnotation(
    final String desc,
    final boolean visible){
    ByteVector bv = new ByteVector();
    // write type, and reserve space for values count
    bv.putShort(cw.newUTF8(desc)).putShort(0);
    AnnotationWriter aw = new AnnotationWriter(cw, true, bv, bv, 2);
    if(visible)
    {
        aw.next = anns;
        anns = aw;
    }
    else
    {
        aw.next = ianns;
        ianns = aw;
    }
    return aw;
}

public AnnotationVisitor visitParameterAnnotation(
    final int parameter,
    final String desc,
    final boolean visible){
    ByteVector bv = new ByteVector();
    // write type, and reserve space for values count
    bv.putShort(cw.newUTF8(desc)).putShort(0);
    AnnotationWriter aw = new AnnotationWriter(cw, true, bv, bv, 2);
    if(visible)
    {
        if(panns == null)
        {
            panns =
                new AnnotationWriter[
                    Type.getArgumentTypes(descriptor).length];
        }
        aw.next = panns[parameter];
        panns[parameter] = aw;
    }
    else
    {
        if(ipanns == null)
        {
            ipanns =
                new AnnotationWriter[
```

```

        Type.getArgumentTypes(descriptor).length];
    }
    aw.next = ipanns[parameter];
    ipanns[parameter] = aw;
}
return aw;
}

public void visitAttribute(final Attribute attr){
    if(attr.isCodeAttribute())
    {
        attr.next = cattrs;
        cattrs = attr;
    }
    else
    {
        attr.next = attrs;
        attrs = attr;
    }
}

public void visitCode(){
}

public void visitFrame(
    final int type,
    final int nLocal,
    final Object[] local,
    final int nStack,
    final Object[] stack){
if(compute == FRAMES)
{
    return;
}

if(type == Opcodes.F_NEW)
{
    startFrame(code.length, nLocal, nStack);
    for(int i = 0; i < nLocal; ++i)
    {
        if(local[i] instanceof String)
        {
            frame[frameIndex++] = Frame.OBJECT
                | cw.addType((String) local[i]);
        }
        else if(local[i] instanceof Integer)
        {
            frame[frameIndex++] = ((Integer) local[i]).intValue();
        }
    }
}
}

```

```

        {
            frame[frameIndex++] =
                Frame.UNINITIALIZED
                | cw.addUninitializedType("",

                                            ((Label) local[i]).position);
        }
    }

    for(int i = 0; i < nStack; ++i)
    {
        if(stack[i] instanceof String)
        {
            frame[frameIndex++] = Frame.OBJECT
                | cw.addType((String) stack[i]);
        }
        else if(stack[i] instanceof Integer)
        {
            frame[frameIndex++] = ((Integer) stack[i]).intValue();
        }
        else
        {
            frame[frameIndex++] =
                Frame.UNINITIALIZED
                | cw.addUninitializedType("",

                                            ((Label) stack[i]).position);
        }
    }

    endFrame();
}

else
{
    int delta;
    if(stackMap == null)
    {
        stackMap = new ByteVector();
        delta = code.length;
    }
    else
    {
        delta = code.length - previousFrameOffset - 1;
    }

    switch(type)
    {
        case Opcodes.F_FULL:
            stackMap.putByte(FULL_FRAME)
                .putShort(delta)
                .putShort(nLocal);
            for(int i = 0; i < nLocal; ++i)
            {
                writeFrameType(local[i]);
            }
    }
}

```

```

        }
        stackMap.putShort(nStack);
        for(int i = 0; i < nStack; ++i)
        {
            writeFrameType(stack[i]);
        }
        break;
    case Opcodes.F_APPEND:
        stackMap.putByte(SAME_FRAME_EXTENDED + nLocal)
            .putShort(delta);
        for(int i = 0; i < nLocal; ++i)
        {
            writeFrameType(local[i]);
        }
        break;
    case Opcodes.F_CHOP:
        stackMap.putByte(SAME_FRAME_EXTENDED - nLocal)
            .putShort(delta);
        break;
    case Opcodes.F_SAME:
        if(delta < 64)
        {
            stackMap.putByte(delta);
        }
        else
        {
            stackMap.putByte(SAME_FRAME_EXTENDED).putShort(delta);
        }
        break;
    case Opcodes.F_SAME1:
        if(delta < 64)
        {
            stackMap.putByte(
                SAME_LOCALS_1_STACK_ITEM_FRAME + delta);
        }
        else
        {
            stackMap.putByte(
                SAME_LOCALS_1_STACK_ITEM_FRAME_EXTENDED)
                .putShort(delta);
        }
        writeFrameType(stack[0]);
        break;
    }

    previousFrameOffset = code.length;
    ++frameCount;
}
}

```

```

public void visitInsn(final int opcode){
    // adds the instruction to the bytecode of the method
    code.putByte(opcode);
    // update currentBlock
    // Label currentBlock = this.currentBlock;
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame.execute(opcode, 0, null, null);
        }
        else
        {
            // updates current and max stack sizes
            int size = stackSize + Frame.SIZE[opcode];
            if(size > maxStackSize)
            {
                maxStackSize = size;
            }
            stackSize = size;
        }
        // if opcode == ATHROW or xRETURN, ends current
        // block (no successor)
        if((opcode >= Opcodes.IRETURN && opcode <= Opcodes.RETURN)
           || opcode == Opcodes.ATHROW)
        {
            noSuccessor();
        }
    }
}

public void visitIntInsn(final int opcode, final int operand){
    // Label currentBlock = this.currentBlock;
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame.execute(opcode, operand, null, null);
        }
        else if(opcode != Opcodes.NEWARRAY)
        {
            // updates current and max stack sizes only for NEWARRAY
            // (stack size variation = 0 for BIPUSH or SIPUSH)
            int size = stackSize + 1;
            if(size > maxStackSize)
            {
                maxStackSize = size;
            }
            stackSize = size;
        }
    }
}

```

```

        }
        // adds the instruction to the bytecode of the method
        if(opcode == Opcodes.SIPUSH)
        {
            code.putInt2(opcode, operand);
        }
        else
        { // BIPUSH or NEWARRAY
            code.putInt1(opcode, operand);
        }
    }

    public void visitVarInsn(final int opcode, final int var){
        // Label currentBlock = this.currentBlock;
        if(currentBlock != null)
        {
            if(compute == FRAMES)
            {
                currentBlock.frame.execute(opcode, var, null, null);
            }
            else
            {
                // updates current and max stack sizes
                if(opcode == Opcodes.RET)
                {
                    // no stack change, but end of current block
                    // (no successor)
                    currentBlock.status |= Label.RET;
                    // save 'stackSize' here for future use
                    // (see {@link #findSubroutineSuccessors})
                    currentBlock.inputStackTop = stackSize;
                    noSuccessor();
                }
                else
                { // xLOAD or xSTORE
                    int size = stackSize + Frame.SIZE[opcode];
                    if(size > maxStackSize)
                    {
                        maxStackSize = size;
                    }
                    stackSize = size;
                }
            }
        }
        if(compute != NOTHING)
        {
            // updates max locals
            int n;
            if(opcode == Opcodes.LLOAD || opcode == Opcodes.DLOAD
                || opcode == Opcodes.LSTORE || opcode == Opcodes.DSTORE)

```

```

    {
        n = var + 2;
    }
    else
    {
        n = var + 1;
    }
    if(n > maxLocals)
    {
        maxLocals = n;
    }
}
// adds the instruction to the bytecode of the method
if(var < 4 && opcode != Opcodes.RET)
{
    int opt;
    if(opcode < Opcodes.ISTORE)
    {
        /* ILOAD_0 */
        opt = 26 + ((opcode - Opcodes.ILOAD) << 2) + var;
    }
    else
    {
        /* ISTORE_0 */
        opt = 59 + ((opcode - Opcodes.ISTORE) << 2) + var;
    }
    code.putByte(opt);
}
else if(var >= 256)
{
    code.putByte(196 /* WIDE */).put12(opcode, var);
}
else
{
    code.put11(opcode, var);
}
if(opcode >= Opcodes.ISTORE && compute == FRAMES && handlerCount > 0)
{
    visitLabel(new Label());
}
}

public void visitTypeInsn(final int opcode, final String desc){
    Item i = cw.newClassItem(desc);
    // Label currentBlock = this.currentBlock;
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame.execute(opcode, code.length, cw, i);
        }
    }
}

```

```

        }
    else if(opcode == Opcodes.NEW)
    {
        // updates current and max stack sizes only if opcode == NEW
        // (no stack change for ANEWARRAY, CHECKCAST, INSTANCEOF)
        int size = stackSize + 1;
        if(size > maxStackSize)
        {
            maxStackSize = size;
        }
        stackSize = size;
    }
}

// adds the instruction to the bytecode of the method
code.put12(opcode, i.index);
}

public void visitFieldInsn(
    final int opcode,
    final String owner,
    final String name,
    final String desc){
    Item i = cw.newFieldItem(owner, name, desc);
    // Label currentBlock = this.currentBlock;
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame.execute(opcode, 0, cw, i);
        }
        else
        {
            int size;
            // computes the stack size variation
            char c = desc.charAt(0);
            switch(opcode)
            {
                case Opcodes.GETSTATIC:
                    size = stackSize + (c == 'D' || c == 'J' ? 2 : 1);
                    break;
                case Opcodes.PUTSTATIC:
                    size = stackSize + (c == 'D' || c == 'J' ? -2 : -1);
                    break;
                case Opcodes.GETFIELD:
                    size = stackSize + (c == 'D' || c == 'J' ? 1 : 0);
                    break;
                    // case Constants.PUTFIELD:
                default:
                    size = stackSize + (c == 'D' || c == 'J' ? -3 : -2);
                    break;
            }
        }
    }
}

```

```

        }
        // updates current and max stack sizes
        if(size > maxStackSize)
        {
            maxStackSize = size;
        }
        stackSize = size;
    }
}

// adds the instruction to the bytecode of the method
code.put12(opcode, i.index);
}

public void visitMethodInsn(
    final int opcode,
    final String owner,
    final String name,
    final String desc){
    boolean itf = opcode == Opcodes.INVOKEINTERFACE;
    Item i = cw.visitMethod(owner, name, desc, itf);
    int argSize = i.intValue;
    // Label currentBlock = this.currentBlock;
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame.execute(opcode, 0, cw, i);
        }
    }
    else
    {
        /*
         * computes the stack size variation. In order not to
         * recompute several times this variation for the same
         * Item, we use the intValue field of this item to store
         * this variation, once it has been computed. More
         * precisely this intValue field stores the sizes of the
         * arguments and of the return value corresponding to desc.
         */
        if(argSize == 0)
        {
            // the above sizes have not been computed yet,
            // so we compute them...
            argSize = getArgumentsAndReturnSizes(desc);
            // ... and we save them in order
            // not to recompute them in the future
            i.intValue = argSize;
        }
        int size;
        if(opcode == Opcodes.INVOKESTATIC)
        {
    }
}

```

```

        size = stackSize - (argSize >> 2) + (argSize & 0x03) + 1;
    }
else
{
    size = stackSize - (argSize >> 2) + (argSize & 0x03);
}
// updates current and max stack sizes
if(size > maxStackSize)
{
    maxStackSize = size;
}
stackSize = size;
}
}

// adds the instruction to the bytecode of the method
if(itf)
{
    if(argSize == 0)
    {
        argSize = getArgumentsAndReturnSizes(desc);
        i.intValue = argSize;
    }
    code.put12(Opcodes.INVOKEINTERFACE, i.index)
        .put11(argSize >> 2, 0);
}
else
{
    code.put12(opcode, i.index);
}
}

public void visitJumpInsn(final int opcode, final Label label){
    Label nextInsn = null;
    // Label currentBlock = this.currentBlock;
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame.execute(opcode, 0, null, null);
            // 'label' is the target of a jump instruction
            label.getFirst().status |= Label.TARGET;
            // adds 'label' as a successor of this basic block
            addSuccessor(Edge.NORMAL, label);
            if(opcode != Opcodes.GOTO)
            {
                // creates a Label for the next basic block
                nextInsn = new Label();
            }
        }
    }
}

```

```

{
    if(opcode == Opcodes.JSR)
    {
        jsr = true;
        currentBlock.status |= Label.JSR;
        addSuccessor(stackSize + 1, label);
        // creates a Label for the next basic block
        nextInsn = new Label();
        /*
         * note that, by construction in this method, a JSR block
         * has at least two successors in the control flow graph:
         * the first one leads the next instruction after the
         * JSR, while the second one leads to the JSR target.
         */
    }
    else
    {
        // updates current stack size (max stack size unchanged
        // because stack size variation always negative in this
        // case)
        stackSize += Frame.SIZE[opcode];
        addSuccessor(stackSize, label);
    }
}
// adds the instruction to the bytecode of the method
if((label.status & Label.RESOLVED) != 0
    && label.position - code.length < Short.MIN_VALUE)
{
/*
 * case of a backward jump with an offset < -32768. In this case
 * we automatically replace GOTO with GOTO_W, JSR with JSR_W
 * and IFxxx <l> with IFNOTxxx <l'> GOTO_W <l>, where IFNOTxxx
 * is the "opposite" opcode of IFxxx (i.e., IFNE for IFEQ) and
 * where <l'> designates the instruction just after the GOTO_W.
*/
if(opcode == Opcodes.GOTO)
{
    code.putByte(200); // GOTO_W
}
else if(opcode == Opcodes.JSR)
{
    code.putByte(201); // JSR_W
}
else
{
    // if the IF instruction is transformed into IFNOT GOTO_W the
    // next instruction becomes the target of the IFNOT
    // instruction
    if(nextInsn != null)

```

```

        {
            nextInsn.status |= Label.TARGET;
        }
        code.putByte(opcode <= 166
                    ? ((opcode + 1) ^ 1) - 1
                    : opcode ^ 1);
        code.putShort(8); // jump offset
        code.putByte(200); // GOTO_W
    }
    label.put(this, code, code.length - 1, true);
}
else
{
/*
 * case of a backward jump with an offset >= -32768, or of a
 * forward jump with, of course, an unknown offset. In these
 * cases we store the offset in 2 bytes (which will be
 * increased in resizeInstructions, if needed).
 */
code.putByte(opcode);
label.put(this, code, code.length - 1, false);
}
if(currentBlock != null)
{
    if(nextInsn != null)
    {
        // if the jump instruction is not a GOTO, the next
        // instruction is also a successor of this instruction.
        // Calling visitLabel adds the label of this next
        // instruction as a successor of the current block,
        // and starts a new basic block
        visitLabel(nextInsn);
    }
    if(opcode == Opcodes.GOTO)
    {
        noSuccessor();
    }
}
}

public void visitLabel(final Label label){
    // resolves previous forward references to label, if any
    resize |= label.resolve(this, code.length, code.data);
    // updates currentBlock
    if((label.status & Label.DEBUG) != 0)
    {
        return;
    }
    if(compute == FRAMES)
    {

```

```
if(currentBlock != null)
{
    if(label.position == currentBlock.position)
    {
        // successive labels, do not start a new basic block
        currentBlock.status |= (label.status & Label.TARGET);
        label.frame = currentBlock.frame;
        return;
    }
    // ends current block (with one new successor)
    addSuccessor(Edge.NORMAL, label);
}
// begins a new current block
currentBlock = label;
if(label.frame == null)
{
    label.frame = new Frame();
    label.frame.owner = label;
}
// updates the basic block list
if(previousBlock != null)
{
    if(label.position == previousBlock.position)
    {
        previousBlock.status |= (label.status & Label.TARGET);
        label.frame = previousBlock.frame;
        currentBlock = previousBlock;
        return;
    }
    previousBlock.successor = label;
}
previousBlock = label;
}
else if(compute == MAXS)
{
    if(currentBlock != null)
    {
        // ends current block (with one new successor)
        currentBlock.outputStackMax = maxStackSize;
        addSuccessor(stackSize, label);
    }
    // begins a new current block
    currentBlock = label;
    // resets the relative current and max stack sizes
    stackSize = 0;
    maxStackSize = 0;
    // updates the basic block list
    if(previousBlock != null)
    {
        previousBlock.successor = label;
```

```

        }
        previousBlock = label;
    }
}

public void visitLdcInsn(final Object cst){
    Item i = cw.newConstItem(cst);
    // Label currentBlock = this.currentBlock;
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame.execute(OpCodes.LDC, 0, cw, i);
        }
        else
        {
            int size;
            // computes the stack size variation
            if(i.type == ClassWriter.LONG ||
               i.type == ClassWriter.DOUBLE)
            {
                size = stackSize + 2;
            }
            else
            {
                size = stackSize + 1;
            }
            // updates current and max stack sizes
            if(size > maxStackSize)
            {
                maxStackSize = size;
            }
            stackSize = size;
        }
    }
    // adds the instruction to the bytecode of the method
    int index = i.index;
    if(i.type == ClassWriter.LONG || i.type == ClassWriter.DOUBLE)
    {
        code.put12(20 /* LDC2_W */, index);
    }
    else if(index >= 256)
    {
        code.put12(19 /* LDC_W */, index);
    }
    else
    {
        code.put11(OpCodes.LDC, index);
    }
}

```

```

public void visitIincInsn(final int var, final int increment){
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame.execute(OpCodes.IINC, var, null, null);
        }
    }
    if(compute != NOTHING)
    {
        // updates max locals
        int n = var + 1;
        if(n > maxLocals)
        {
            maxLocals = n;
        }
    }
    // adds the instruction to the bytecode of the method
    if((var > 255) || (increment > 127) || (increment < -128))
    {
        code.putByte(196 /* WIDE */)
            .put12(OpCodes.IINC, var)
            .putShort(increment);
    }
    else
    {
        code.putByte(OpCodes.IINC).put11(var, increment);
    }
}

public void visitTableSwitchInsn(
    final int min,
    final int max,
    final Label dflt,
    final Label labels[]){
    // adds the instruction to the bytecode of the method
    int source = code.length;
    code.putByte(OpCodes.TABLESWITCH);
    code.length += (4 - code.length % 4) % 4;
    dflt.put(this, code, source, true);
    code.putInt(min).putInt(max);
    for(int i = 0; i < labels.length; ++i)
    {
        labels[i].put(this, code, source, true);
    }
    // updates currentBlock
    visitSwitchInsn(dflt, labels);
}

```

```

public void visitLookupSwitchInsn(
    final Label dflt,
    final int keys[],
    final Label labels[]){
    // adds the instruction to the bytecode of the method
    int source = code.length;
    code.putByte(OpCodes.LOOKUPSWITCH);
    code.length += (4 - code.length % 4) % 4;
    dflt.put(this, code, source, true);
    code.putInt(labels.length);
    for(int i = 0; i < labels.length; ++i)
    {
        code.putInt(keys[i]);
        labels[i].put(this, code, source, true);
    }
    // updates currentBlock
    visitSwitchInsn(dflt, labels);
}

private void visitSwitchInsn(final Label dflt, final Label[] labels){
    // Label currentBlock = this.currentBlock;
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame
                .execute(OpCodes.LOOKUPSWITCH, 0, null, null);
            // adds current block successors
            addSuccessor(Edge.NORMAL, dflt);
            dflt.getFirst().status |= Label.TARGET;
            for(int i = 0; i < labels.length; ++i)
            {
                addSuccessor(Edge.NORMAL, labels[i]);
                labels[i].getFirst().status |= Label.TARGET;
            }
        }
        else
        {
            // updates current stack size (max stack size unchanged)
            --stackSize;
            // adds current block successors
            addSuccessor(stackSize, dflt);
            for(int i = 0; i < labels.length; ++i)
            {
                addSuccessor(stackSize, labels[i]);
            }
        }
        // ends current block
        noSuccessor();
    }
}

```

```

}

public void visitMultiANewArrayInsn(final String desc, final int dims){
    Item i = cw.newClassItem(desc);
    // Label currentBlock = this.currentBlock;
    if(currentBlock != null)
    {
        if(compute == FRAMES)
        {
            currentBlock.frame
                .execute(0pcodes.MULTIANEWARRAY, dims, cw, i);
        }
        else
        {
            // updates current stack size (max stack size unchanged
            // because stack size variation always negative or null)
            stackSize += 1 - dims;
        }
    }
    // adds the instruction to the bytecode of the method
    code.put12(0pcodes.MULTIANEWARRAY, i.index).putByte(dims);
}

public void visitTryCatchBlock(
    final Label start,
    final Label end,
    final Label handler,
    final String type){
    ++handlerCount;
    Handler h = new Handler();
    h.start = start;
    h.end = end;
    h.handler = handler;
    h.desc = type;
    h.type = type != null ? cw.newClass(type) : 0;
    if(lastHandler == null)
    {
        firstHandler = h;
    }
    else
    {
        lastHandler.next = h;
    }
    lastHandler = h;
}

public void visitLocalVariable(
    final String name,
    final String desc,
    final String signature,

```

```

        final Label start,
        final Label end,
        final int index){
    if(signature != null)
    {
        if(localVarType == null)
        {
            localVarType = new ByteVector();
        }
        ++localVarTypeCount;
        localVarType.putShort(start.position)
            .putShort(end.position - start.position)
            .putShort(cw.newUTF8(name))
            .putShort(cw.newUTF8(signature))
            .putShort(index);
    }
    if(localVar == null)
    {
        localVar = new ByteVector();
    }
    ++localVarCount;
    localVar.putShort(start.position)
        .putShort(end.position - start.position)
        .putShort(cw.newUTF8(name))
        .putShort(cw.newUTF8(desc))
        .putShort(index);
    if(compute != NOTHING)
    {
        // updates max locals
        char c = desc.charAt(0);
        int n = index + (c == 'J' || c == 'D' ? 2 : 1);
        if(n > maxLocals)
        {
            maxLocals = n;
        }
    }
}

public void visitLineNumber(final int line, final Label start){
    if(lineNumber == null)
    {
        lineNumber = new ByteVector();
    }
    ++lineNumberCount;
    lineNumber.putShort(start.position);
    lineNumber.putShort(line);
}

public void visitMaxs(final int maxStack, final int maxLocals){
    if(compute == FRAMES)

```

```

{
// completes the control flow graph with exception handler blocks
Handler handler = firstHandler;
while(handler != null)
{
    Label l = handler.start.getFirst();
    Label h = handler.handler.getFirst();
    Label e = handler.end.getFirst();
    // computes the kind of the edges to 'h'
    String t = handler.desc == null
        ? "java/lang/Throwable"
        : handler.desc;
    int kind = Frame.OBJECT | cw.addType(t);
    // h is an exception handler
    h.status |= Label.TARGET;
    // adds 'h' as a successor of labels between 'start'
    // and 'end'
    while(l != e)
    {
        // creates an edge to 'h'
        Edge b = new Edge();
        b.info = kind;
        b.successor = h;
        // adds it to the successors of 'l'
        b.next = l.successors;
        l.successors = b;
        // goes to the next label
        l = l.successor;
    }
    handler = handler.next;
}

// creates and visits the first (implicit) frame
Frame f = labels.frame;
Type[] args = Type.getArgumentTypes(descriptor);
f.initInputFrame(cw, access, args, this.maxLocals);
visitFrame(f);

/*
 * fix point algorithm: mark the first basic block as 'changed'
 * (i.e. put it in the 'changed' list) and, while there are
 * changed basic blocks, choose one, mark it as unchanged,
 * and update its successors (which can be changed in the
 * process).
 */
int max = 0;
Label changed = labels;
while(changed != null)
{
    // removes a basic block from the list of changed basic
}

```

```

// blocks
Label l = changed;
changed = changed.next;
l.next = null;
f = l.frame;
// a reacheable jump target must be stored in the stack map
if((l.status & Label.TARGET) != 0)
{
    l.status |= Label.STORE;
}
// all visited labels are reacheable, by definition
l.status |= Label.REACHABLE;
// updates the (absolute) maximum stack size
int blockMax = f.inputStack.length + l.outputStackMax;
if(blockMax > max)
{
    max = blockMax;
}
// updates the successors of the current basic block
Edge e = l.successors;
while(e != null)
{
    Label n = e.successor.getFirst();
    boolean change = f.merge(cw, n.frame, e.info);
    if(change && n.next == null)
    {
        // if n has changed and is not already in the
        // 'changed' list, adds it to this list
        n.next = changed;
        changed = n;
    }
    e = e.next;
}
this.maxStack = max;

// visits all the frames that must be stored in the stack map
Label l = labels;
while(l != null)
{
    f = l.frame;
    if((l.status & Label.STORE) != 0)
    {
        visitFrame(f);
    }
    if((l.status & Label.REACHABLE) == 0)
    {
        // finds start and end of dead basic block
        Label k = l.successor;
        int start = l.position;
    }
}

```

```

        int end = (k == null ? code.length : k.position) - 1;
        // if non empty basic block
        if(end >= start)
        {
            //
            // replaces instructions with NOP ... NOP ATHROW
            for(int i = start; i < end; ++i)
            {
                code.data[i] = Opcodes.NOP;
            }
            code.data[end] = (byte) Opcodes.ATHROW;
            // emits a frame for this unreachable block
            startFrame(start, 0, 1);
            frame[frameIndex++] =
                Frame.OBJECT
                | cw.addType("java/lang/Throwable");
            endFrame();
        }
    }
    l = l.successor;
}
}
else if(compute == MAXS)
{
    // completes the control flow graph with exception handler blocks
    Handler handler = firstHandler;
    while(handler != null)
    {
        Label l = handler.start;
        Label h = handler.handler;
        Label e = handler.end;
        // adds 'h' as a successor of labels between 'start'
        // and 'end'
        while(l != e)
        {
            //
            // creates an edge to 'h'
            Edge b = new Edge();
            b.info = Edge.EXCEPTION;
            b.successor = h;
            // adds it to the successors of 'l'
            if((l.status & Label.JSR) != 0)
            {
                //
                // if l is a JSR block, adds b after the first two
                // edges to preserve the hypothesis about JSR block
                // successors order (see {@link #visitJumpInsn})
                b.next = l.successors.next.next;
                l.successors.next.next = b;
            }
            else
            {
                b.next = l.successors;
            }
        }
    }
}

```

```

        l.successors = b;
    }
    // goes to the next label
    l = l.successor;
}
handler = handler.next;
}

if(jsr)
{
// completes the control flow graph with the RET successors
/*
 * first step: finds the subroutines. This step determines,
 * for each basic block, to which subroutine(s) it belongs,
 * and stores this set as a bit set in the
 * {@link Label#status} field. Subroutines are numbered
 * with powers of two, from 0x1000 to 0x80000000 (so there
 * must be at most 20 subroutines in a method).
 */
// finds the basic blocks that belong to the "main"
// subroutine
int id = 0x1000;
findSubroutine(labels, id);
// finds the basic blocks that belong to the real subroutines
Label l = labels;
while(l != null)
{
    if((l.status & Label.JSR) != 0)
    {
        // the subroutine is defined by l's TARGET, not by l
        Label subroutine = l.successors.next.successor;
        // if this subroutine does not have an id yet...
        if((subroutine.status & ~0xFFFF) == 0)
        {
            // ...assigns it a new id and finds its
            // basic blocks
            id = id << 1;
            findSubroutine(subroutine, id);
        }
    }
    l = l.successor;
}
// second step: finds the successors of RET blocks
findSubroutineSuccessors(0x1000, new Label[10], 0);
}

/*
 * control flow analysis algorithm: while the block stack is not
 * empty, pop a block from this stack, update the max stack size,
 * compute the true (non relative) begin stack size of the

```

```

* successors of this block, and push these successors onto the
* stack (unless they have already been pushed onto the stack).
* Note: by hypothesis, the {@link Label#inputStackTop} of the
* blocks in the block stack are the true (non relative)
* beginning stack sizes of these blocks.
*/
int max = 0;
Label stack = labels;
while(stack != null)
{
    //
    // pops a block from the stack
    Label l = stack;
    stack = stack.next;
    //
    // computes the true (non relative) max stack size of this
    // block
    int start = l.inputStackTop;
    int blockMax = start + l.outputStackMax;
    //
    // updates the global max stack size
    if(blockMax > max)
    {
        max = blockMax;
    }
    //
    // analyses the successors of the block
    Edge b = l.successors;
    if((l.status & Label.JSR) != 0)
    {
        //
        // ignores the first edge of JSR blocks
        // (virtual successor)
        b = b.next;
    }
    while(b != null)
    {
        l = b.successor;
        //
        // if this successor has not already been pushed...
        if((l.status & Label.PUSHED) == 0)
        {
            //
            // computes its true beginning stack size...
            l.inputStackTop = b.info == Edge.EXCEPTION
                ? 1 : start + b.info;
            //
            // ...and pushes it onto the stack
            l.status |= Label.PUSHED;
            l.next = stack;
            stack = l;
        }
        b = b.next;
    }
}
this.maxStack = max;
}
else

```

```

    {
    this.maxStack = maxStack;
    this.maxLocals = maxLocals;
    }
}

public void visitEnd(){

// -----
// Utility methods: control flow analysis algorithm
// -----


/***
 * Computes the size of the arguments and of the return value of a
 * method.
 *
 * @param desc the descriptor of a method.
 * @return the size of the arguments of the method (plus one for the
 *         implicit this argument), argSize, and the size of its return
 *         value, retSize, packed into a single int i =
 *         <tt>(argSize << 2) | retSize</tt> (argSize is therefore equal
 *         to <tt>i >> 2</tt>, and retSize to <tt>i & 0x03</tt>).
 */
static int getArgumentsAndReturnSizes(final String desc){
    int n = 1;
    int c = 1;
    while(true)
    {
        char car = desc.charAt(c++);
        if(car == ')')
        {
            car = desc.charAt(c);
            return n << 2
                | (car == 'V' ? 0 : (car == 'D' || car == 'J' ? 2 : 1));
        }
        else if(car == 'L')
        {
            while(desc.charAt(c++) != ';')
            {
            }
            n += 1;
        }
        else if(car == '[')
        {
            while((car = desc.charAt(c)) == '[')
            {
                ++c;
            }
            if(car == 'D' || car == 'J')

```

```

        {
            n -= 1;
        }
    }
    else if(car == 'D' || car == 'J')
    {
        n += 2;
    }
    else
    {
        n += 1;
    }
}
}

/**
 * Adds a successor to the {@link #currentBlock currentBlock} block.
 *
 * @param info      information about the control flow edge to be added.
 * @param successor the successor block to be added to the current block.
 */
private void addSuccessor(final int info, final Label successor){
    // creates and initializes an Edge object...
    Edge b = new Edge();
    b.info = info;
    b.successor = successor;
    // ...and adds it to the successor list of the currentBlock block
    b.next = currentBlock.successors;
    currentBlock.successors = b;
}

/**
 * Ends the current basic block. This method must be used in the case
 * where the current basic block does not have any successor.
 */
private void noSuccessor(){
    if(compute == FRAMES)
    {
        Label l = new Label();
        l.frame = new Frame();
        l.frame.owner = l;
        l.resolve(this, code.length, code.data);
        previousBlock.successor = l;
        previousBlock = l;
    }
    else
    {
        currentBlock.outputStackMax = maxStackSize;
    }
    currentBlock = null;
}

```

```

}

/***
 * Finds the basic blocks that belong to a given subroutine, and marks
 * these blocks as belonging to this subroutine (by using
 * {@link Label#status} as a bit set (see {@link #visitMaxs}). This
 * recursive method follows the control flow graph to find all the
 * blocks that are reachable from the given block WITHOUT following
 * any JSR target.
 *
 * @param block a block that belongs to the subroutine
 * @param id    the id of this subroutine
 */
private void findSubroutine(final Label block, final int id){
    // if 'block' is already marked as belonging to subroutine 'id',
    // returns
    if((block.status & id) != 0)
    {
        return;
    }
    // marks 'block' as belonging to subroutine 'id'
    block.status |= id;
    // calls this method recursively on each successor, except
    // JSR targets
    Edge e = block.successors;
    while(e != null)
    {
        // if 'block' is a JSR block, then 'block.successors.next'
        // leads to the JSR target (see {@link #visitJumpInsn}) and
        // must therefore not be followed
        if((block.status & Label.JSR) == 0 || e != block.successors.next)
        {
            findSubroutine(e.successor, id);
        }
        e = e.next;
    }
}

/***
 * Finds the successors of the RET blocks of the specified subroutine,
 * and of any nested subroutine it calls.
 *
 * @param id    id of the subroutine whose RET block successors must
 *              be found.
 * @param JSRs  the JSR blocks that were followed to reach this
 *              subroutine.
 * @param nJSRs number of JSR blocks in the JSRs array.
 */
private void findSubroutineSuccessors(
    final int id,

```



```

        l.successors = e;
        break;
    }
}
l = l.successor;
}
}

// -----
// Utility methods: stack map frames
// -----


/**
 * Visits a frame that has been computed from scratch.
 *
 * @param f the frame that must be visited.
 */
private void visitFrame(final Frame f){
    int i, t;
    int nTop = 0;
    int nLocal = 0;
    int nStack = 0;
    int[] locals = f.inputLocals;
    int[] stacks = f.inputStack;
    // computes the number of locals (ignores TOP types that are just
    // after a LONG or a DOUBLE, and all trailing TOP types)
    for(i = 0; i < locals.length; ++i)
    {
        t = locals[i];
        if(t == Frame.TOP)
        {
            ++
            nTop;
        }
        else
        {
            nLocal += nTop + 1;
            nTop = 0;
        }
        if(t == Frame.LONG || t == Frame.DOUBLE)
        {
            ++
            i;
        }
    }
    // computes the stack size (ignores TOP types that are just after
    // a LONG or a DOUBLE)
    for(i = 0; i < stacks.length; ++i)
    {
        t = stacks[i];
    }
}

```

```

        ++nStack;
        if(t == Frame.LONG || t == Frame.DOUBLE)
        {
            ++i;
        }
    }

    // visits the frame and its content
    startFrame(f.owner.position, nLocal, nStack);
    for(i = 0; nLocal > 0; ++i, --nLocal)
    {
        t = locals[i];
        frame[frameIndex++] = t;
        if(t == Frame.LONG || t == Frame.DOUBLE)
        {
            ++i;
        }
    }

    for(i = 0; i < stacks.length; ++i)
    {
        t = stacks[i];
        frame[frameIndex++] = t;
        if(t == Frame.LONG || t == Frame.DOUBLE)
        {
            ++i;
        }
    }

    endFrame();
}

/**
 * Starts the visit of a stack map frame.
 *
 * @param offset the offset of the instruction to which the frame
 *               corresponds.
 * @param nLocal the number of local variables in the frame.
 * @param nStack the number of stack elements in the frame.
 */
private void startFrame(final int offset,
                       final int nLocal,
                       final int nStack){
    int n = 3 + nLocal + nStack;
    if(frame == null || frame.length < n)
    {
        frame = new int[n];
    }
    frame[0] = offset;
    frame[1] = nLocal;
    frame[2] = nStack;
    frameIndex = 3;
}

```

```

/**
 * Checks if the visit of the current frame {@link #frame} is finished,
 * and if yes, write it in the StackMapTable attribute.
 */
private void endFrame(){
    if(previousFrame != null)
        { // do not write the first frame
        if(stackMap == null)
            {
            stackMap = new ByteVector();
            }
        writeFrame();
        ++frameCount;
        }
    previousFrame = frame;
    frame = null;
}

/**
 * Compress and writes the current frame {@link #frame} in the
 * StackMapTable attribute.
 */
private void writeFrame(){
    int clocalsSize = frame[1];
    int cstackSize = frame[2];
    if((cw.version & 0xFFFF) < Opcodes.V1_6)
    {
        stackMap.putShort(frame[0]).putShort(clocalsSize);
        writeFrameTypes(3, 3 + clocalsSize);
        stackMap.putShort(cstackSize);
        writeFrameTypes(3 + clocalsSize, 3 + clocalsSize + cstackSize);
        return;
    }
    int localsSize = previousFrame[1];
    int type = FULL_FRAME;
    int k = 0;
    int delta;
    if(frameCount == 0)
    {
        delta = frame[0];
    }
    else
    {
        delta = frame[0] - previousFrame[0] - 1;
    }
    if(cstackSize == 0)
    {
        k = clocalsSize - localsSize;
        switch(k)

```

```

{
case-3:
case-2:
case-1:
    type = CHOP_FRAME;
    localsSize = clocalsSize;
    break;
case 0:
    type = delta < 64 ? SAME_FRAME : SAME_FRAME_EXTENDED;
    break;
case 1:
case 2:
case 3:
    type = APPEND_FRAME;
    break;
}
}

else if(clocalsSize == localsSize && cstackSize == 1)
{
    type = delta < 63
        ? SAME_LOCALS_1_STACK_ITEM_FRAME
        : SAME_LOCALS_1_STACK_ITEM_FRAME_EXTENDED;
}

if(type != FULL_FRAME)
{
    // verify if locals are the same
    int l = 3;
    for(int j = 0; j < localsSize; j++)
    {
        if(frame[1] != previousFrame[1])
        {
            type = FULL_FRAME;
            break;
        }
        l++;
    }
}
switch(type)
{
case SAME_FRAME:
    stackMap.putByte(delta);
    break;
case SAME_LOCALS_1_STACK_ITEM_FRAME:
    stackMap.putByte(SAME_LOCALS_1_STACK_ITEM_FRAME + delta);
    writeFrameTypes(3 + clocalsSize, 4 + clocalsSize);
    break;
case SAME_LOCALS_1_STACK_ITEM_FRAME_EXTENDED:
    stackMap.putByte(SAME_LOCALS_1_STACK_ITEM_FRAME_EXTENDED)
        .putShort(delta);
    writeFrameTypes(3 + clocalsSize, 4 + clocalsSize);
}

```

```

        break;
    case SAME_FRAME_EXTENDED:
        stackMap.putByte(SAME_FRAME_EXTENDED).putShort(delta);
        break;
    case CHOP_FRAME:
        stackMap.putByte(SAME_FRAME_EXTENDED + k).putShort(delta);
        break;
    case APPEND_FRAME:
        stackMap.putByte(SAME_FRAME_EXTENDED + k).putShort(delta);
        writeFrameTypes(3 + localsSize, 3 + clocalsSize);
        break;
    // case FULL_FRAME:
    default:
        stackMap.putByte(FULL_FRAME)
            .putShort(delta)
            .putShort(clocalsSize);
        writeFrameTypes(3, 3 + clocalsSize);
        stackMap.putShort(cstackSize);
        writeFrameTypes(3+clocalsSize, 3+clocalsSize+cstackSize);
    }
}

/**
 * Writes some types of the current frame {@link #frame} into the
 * StackMapTableAttribute. This method converts types from the format
 * used in {@link Label} to the format used in StackMapTable attributes.
 * In particular, it converts type table indexes to constant pool
 * indexes.
 *
 * @param start index of the first type in {@link #frame} to write.
 * @param end   index of last type in {@link #frame} to write
 *             (exclusive).
 */
private void writeFrameTypes(final int start, final int end){
    for(int i = start; i < end; ++i)
    {
        int t = frame[i];
        int d = t & Frame.DIM;
        if(d == 0)
        {
            int v = t & Frame.BASE_VALUE;
            switch(t & Frame.BASE_KIND)
            {
                case Frame.OBJECT:
                    stackMap.putByte(7)
                        .putShort(cw.newClass(cw.typeTable[v].strVal1));
                    break;
                case Frame.UNINITIALIZED:
                    stackMap.putByte(8).putShort(cw.typeTable[v].intVal);
                    break;
            }
        }
    }
}

```

```
        default:
            stackMap.putByte(v);
        }
    }
else
{
    StringBuffer buf = new StringBuffer();
    d >= 28;
    while(d-- > 0)
    {
        buf.append('[');
    }
    if((t & Frame.BASE_KIND) == Frame.OBJECT)
    {
        buf.append('L');
        buf.append(cw.typeTable[t & Frame.BASE_VALUE].strVal1);
        buf.append(';');
    }
    else
    {
        switch(t & 0xF)
        {
            case 1:
                buf.append('I');
                break;
            case 2:
                buf.append('F');
                break;
            case 3:
                buf.append('D');
                break;
            case 9:
                buf.append('Z');
                break;
            case 10:
                buf.append('B');
                break;
            case 11:
                buf.append('C');
                break;
            case 12:
                buf.append('S');
                break;
            default:
                buf.append('J');
        }
    }
    stackMap.putByte(7).putShort(cw.newClass(buf.toString()));
}
}
```

```

}

private void writeFrameType(final Object type){
    if(type instanceof String)
    {
        stackMap.putByte(7).putShort(cw.newClass((String) type));
    }
    else if(type instanceof Integer)
    {
        stackMap.putByte(((Integer) type).intValue());
    }
    else
    {
        stackMap.putByte(8).putShort(((Label) type).position);
    }
}

// -----
// Utility methods: dump bytecode array
// -----


/***
 * Returns the size of the bytecode of this method.
 *
 * @return the size of the bytecode of this method.
 */
final int getSize(){
    if(classReaderOffset != 0)
    {
        return 6 + classReaderLength;
    }
    if(resize)
    {
        // replaces the temporary jump opcodes introduced by
        // Label.resolve.
        resizeInstructions();
    }
    int size = 8;
    if(code.length > 0)
    {
        cw.newUTF8("Code");
        size += 18 + code.length + 8 * handlerCount;
        if(localVar != null)
        {
            cw.newUTF8("LocalVariableTable");
            size += 8 + localVar.length;
        }
        if(localVarType != null)
        {
            cw.newUTF8("LocalVariableTypeTable");
        }
    }
}

```

```
        size += 8 + localVarType.length;
    }
    if(lineNumber != null)
    {
        cw.newUTF8("LineNumberTable");
        size += 8 + lineNumber.length;
    }
    if(stackMap != null)
    {
        boolean zip = (cw.version & 0xFFFF) >= Opcodes.V1_6;
        cw.newUTF8(zip ? "StackMapTable" : "StackMap");
        size += 8 + stackMap.length;
    }
    if(cattrs != null)
    {
        size += cattrs.getSize(cw,
                               code.data,
                               code.length,
                               maxStack,
                               maxLocals);
    }
}
if(exceptionCount > 0)
{
    cw.newUTF8("Exceptions");
    size += 8 + 2 * exceptionCount;
}
if((access & Opcodes.ACC_SYNTHETIC) != 0
    && (cw.version & 0xffff) < Opcodes.V1_5)
{
    cw.newUTF8("Synthetic");
    size += 6;
}
if((access & Opcodes.ACC_DEPRECATED) != 0)
{
    cw.newUTF8("Deprecated");
    size += 6;
}
if(signature != null)
{
    cw.newUTF8("Signature");
    cw.newUTF8(signature);
    size += 8;
}
if(annd != null)
{
    cw.newUTF8("AnnotationDefault");
    size += 6 + annd.length;
}
if(anns != null)
```

```

    {
      cw.newUTF8("RuntimeVisibleAnnotations");
      size += 8 + anns.getSize();
    }
    if(ianns != null)
    {
      cw.newUTF8("RuntimeInvisibleAnnotations");
      size += 8 + ianns.getSize();
    }
    if(panns != null)
    {
      cw.newUTF8("RuntimeVisibleParameterAnnotations");
      size += 7 + 2 * panns.length;
      for(int i = panns.length - 1; i >= 0; --i)
      {
        size += panns[i] == null ? 0 : panns[i].getSize();
      }
    }
    if(ipannts != null)
    {
      cw.newUTF8("RuntimeInvisibleParameterAnnotations");
      size += 7 + 2 * ipannts.length;
      for(int i = ipannts.length - 1; i >= 0; --i)
      {
        size += ipannts[i] == null ? 0 : ipannts[i].getSize();
      }
    }
    if(attrs != null)
    {
      size += attrs.getSize(cw, null, 0, -1, -1);
    }
    return size;
}

/**
 * Puts the bytecode of this method in the given byte vector.
 *
 * @param out the byte vector into which the bytecode of this method must
 *            be copied.
 */
final void put(final ByteVector out){
  out.putShort(access).putShort(name).putShort(desc);
  if(classReaderOffset != 0)
  {
    out.putByteArray(cw.cr.b, classReaderOffset, classReaderLength);
    return;
  }
  int attributeCount = 0;
  if(code.length > 0)
  {

```

```
    ++attributeCount;
}
if(exceptionCount > 0)
{
    ++attributeCount;
}
if((access & Opcodes.ACC_SYNTHETIC) != 0
    && (cw.version & 0xffff) < Opcodes.V1_5)
{
    ++attributeCount;
}
if((access & Opcodes.ACC_DEPRECATED) != 0)
{
    ++attributeCount;
}
if(signature != null)
{
    ++attributeCount;
}
if(annd != null)
{
    ++attributeCount;
}
if(anns != null)
{
    ++attributeCount;
}
if(ianns != null)
{
    ++attributeCount;
}
if(panns != null)
{
    ++attributeCount;
}
if(ipannts != null)
{
    ++attributeCount;
}
if(attrs != null)
{
    attributeCount += attrs.getCount();
}
out.putShort(attributeCount);
if(code.length > 0)
{
    int size = 12 + code.length + 8 * handlerCount;
    if(localVar != null)
    {
        size += 8 + localVar.length;
```

```
        }
        if(localVarType != null)
        {
            size += 8 + localVarType.length;
        }
        if(lineNumber != null)
        {
            size += 8 + lineNumber.length;
        }
        if(stackMap != null)
        {
            size += 8 + stackMap.length;
        }
        if(cattrs != null)
        {
            size += cattrs.getSize(cw,
                code.data,
                code.length,
                maxStack,
                maxLocals);
        }
        out.putShort(cw.newUTF8("Code")).putInt(size);
        out.putShort(maxStack).putShort(maxLocals);
        out.putInt(code.length).putByteArray(code.data, 0, code.length);
        out.putShort(handlerCount);
        if(handlerCount > 0)
        {
            Handler h = firstHandler;
            while(h != null)
            {
                out.putShort(h.start.position)
                    .putShort(h.end.position)
                    .putShort(h.handler.position)
                    .putShort(h.type);
                h = h.next;
            }
        }
        attributeCount = 0;
        if(localVar != null)
        {
            ++attributeCount;
        }
        if(localVarType != null)
        {
            ++attributeCount;
        }
        if(lineNumber != null)
        {
            ++attributeCount;
        }
```

```

    if(stackMap != null)
    {
        ++attributeCount;
    }
    if(cattrs != null)
    {
        attributeCount += cattrs.getCount();
    }
    out.putShort(attributeCount);
    if(localVar != null)
    {
        out.putShort(cw.newUTF8("LocalVariableTable"));
        out.putInt(localVar.length + 2).putShort(localVarCount);
        out.putByteArray(localVar.data, 0, localVar.length);
    }
    if(localVarType != null)
    {
        out.putShort(cw.newUTF8("LocalVariableTypeTable"));
        out.putInt(localVarType.length + 2)
            .putShort(localVarTypeCount);
        out.putByteArray(localVarType.data, 0, localVarType.length);
    }
    if(lineNumber != null)
    {
        out.putShort(cw.newUTF8("LineNumberTable"));
        out.putInt(lineNumber.length + 2).putShort(lineNumberCount);
        out.putByteArray(lineNumber.data, 0, lineNumber.length);
    }
    if(stackMap != null)
    {
        boolean zip = (cw.version & 0xFFFF) >= Opcodes.V1_6;
        out.putShort(cw.newUTF8(zip ? "StackMapTable" : "StackMap"));
        out.putInt(stackMap.length + 2).putShort(frameCount);
        out.putByteArray(stackMap.data, 0, stackMap.length);
    }
    if(cattrs != null)
    {
        cattrs.put(cw, code.data, code.length,
                   maxLocals, maxStack, out);
    }
}
if(exceptionCount > 0)
{
    out.putShort(cw.newUTF8("Exceptions"))
        .putInt(2 * exceptionCount + 2);
    out.putShort(exceptionCount);
    for(int i = 0; i < exceptionCount; ++i)
    {
        out.putShort(exceptions[i]);
    }
}

```

```

        }
        if((access & Opcodes.ACC_SYNTHETIC) != 0
          && (cw.version & 0xffff) < Opcodes.V1_5)
        {
          out.putShort(cw.newUTF8("Synthetic")).putInt(0);
        }
        if((access & Opcodes.ACC_DEPRECATED) != 0)
        {
          out.putShort(cw.newUTF8("Deprecated")).putInt(0);
        }
        if(signature != null)
        {
          out.putShort(cw.newUTF8("Signature"))
            .putInt(2)
            .putShort(cw.newUTF8(signature));
        }
        if(annd != null)
        {
          out.putShort(cw.newUTF8("AnnotationDefault"));
          out.putInt(annd.length);
          out.putByteArray(annd.data, 0, annd.length);
        }
        if(anns != null)
        {
          out.putShort(cw.newUTF8("RuntimeVisibleAnnotations"));
          anns.put(out);
        }
        if(ianns != null)
        {
          out.putShort(cw.newUTF8("RuntimeInvisibleAnnotations"));
          ianns.put(out);
        }
        if(panns != null)
        {
          out.putShort(cw.newUTF8("RuntimeVisibleParameterAnnotations"));
          AnnotationWriter.put(panns, out);
        }
        if(ipanns != null)
        {
          out.putShort(cw.newUTF8("RuntimeInvisibleParameterAnnotations"));
          AnnotationWriter.put(ipanns, out);
        }
        if(attrs != null)
        {
          attrs.put(cw, null, 0, -1, -1, out);
        }
      }

      // -----
      // Utility methods: instruction resizing (used to handle GOTO_W and
    
```

```
// JSR_W)
// -----
/** 
 * Resizes and replaces the temporary instructions inserted by
 * {@link Label#resolve} for wide forward jumps, while keeping jump
 * offsets and instruction addresses consistent. This may require to
 * resize other existing instructions, or even to introduce new
 * instructions: for example, increasing the size of an instruction
 * by 2 at the middle of a method can increases the offset of an
 * IFEQ instruction from 32766 to 32768, in which case IFEQ 32766
 * must be replaced with IFNEQ 8 GOTO_W 32765. This, in turn, may
 * require to increase the size of another jump instruction, and so
 * on... All these operations are handled automatically by this
 * method. <p> <i>This method must be called after all the method
 * that is being built has been visited</i>. In particular, the
 * {@link Label Label} objects used to construct the method are no
 * longer valid after this method has been called.
*/
private void resizeInstructions(){
    byte[] b = code.data; // bytecode of the method
    int u, v, label; // indexes in b
    int i, j; // loop indexes
    /*
     * 1st step: As explained above, resizing an instruction may require
     * to resize another one, which may require to resize yet another
     * one, and so on. The first step of the algorithm consists in
     * finding all the instructions that need to be resized, without
     * modifying the code. This is done by the following "fix point"
     * algorithm:
     *
     * Parse the code to find the jump instructions whose offset will
     * need more than 2 bytes to be stored (the future offset is
     * computed from the current offset and from the number of bytes
     * that will be inserted or removed between the source and target
     * instructions). For each such instruction, adds an entry in (a
     * copy of) the indexes and sizes arrays (if this has not already
     * been done in a previous iteration!).
     *
     * If at least one entry has been added during the previous step, go
     * back to the beginning, otherwise stop.
     *
     * In fact the real algorithm is complicated by the fact that the
     * size of TABLESWITCH and LOOKUPSWITCH instructions depends on their
     * position in the bytecode (because of padding). In order to ensure
     * the convergence of the algorithm, the number of bytes to be added
     * or removed from these instructions is over estimated during the
     * previous loop, and computed exactly only after the loop is
     * finished (this requires another pass to parse the bytecode of
     * the method).
    
```

```

*/
int[] allIndexes = new int[0]; // copy of indexes
int[] allSizes = new int[0]; // copy of sizes
boolean[] resize; // instructions to be resized
int newOffset; // future offset of a jump instruction

resize = new boolean[code.length];

// 3 = loop again, 2 = loop ended, 1 = last pass, 0 = done
int state = 3;
do
{
    if(state == 3)
    {
        state = 2;
    }
    u = 0;
    while(u < b.length)
    {
        int opcode = b[u] & 0xFF; // opcode of current instruction
        int insert = 0; // bytes to be added after this instruction

        switch(ClassWriter.TYPE[opcode])
        {
            case ClassWriter.NOARG_INSN:
            case ClassWriter.IMPLVAR_INSN:
                u += 1;
                break;
            case ClassWriter.LABEL_INSN:
                if(opcode > 201)
                {
                    // converts temporary opcodes 202 to 217, 218 and
                    // 219 to IFEQ ... JSR (inclusive), IFNULL and
                    // IFNONNULL
                    opcode =
                        opcode < 218 ? opcode - 49 : opcode - 20;
                    label = u + readUnsignedShort(b, u + 1);
                }
                else
                {
                    label = u + readShort(b, u + 1);
                }
                newOffset =
                    getNewOffset(allIndexes, allSizes, u, label);
                if(newOffset < Short.MIN_VALUE
                   || newOffset > Short.MAX_VALUE)
                {
                    if(!resize[u])
                    {
                        if(opcode == Opcodes.GOTO

```

```

        || opcode == Opcodes.JSR)
    {
        // two additional bytes will be required to
        // replace this GOTO or JSR instruction with
        // a GOTO_W or a JSR_W
        insert = 2;
    }
    else
    {
        // five additional bytes will be required to
        // replace this IFxxx <l> instruction with
        // IFNOTxxx <l'> GOTO_W <l>, where IFNOTxxx
        // is the "opposite" opcode of IFxxx (i.e.,
        // IFNE for IFEQ) and where <l'> designates
        // the instruction just after the GOTO_W.
        insert = 5;
    }
    resize[u] = true;
}
}
u += 3;
break;
case ClassWriter.LABELW_INSN:
u += 5;
break;
case ClassWriter.TABL_INSN:
if(state == 1)
{
    // true number of bytes to be added (or removed)
    // from this instruction = (future number of
    // padding bytes - current number of padding
    // byte) - previously over estimated variation =
    // = ((3 - newOffset%4) - (3 - u%4)) - u%4
    // = (-newOffset%4 + u%4) - u%4
    // = -(newOffset & 3)
    newOffset =
        getNewOffset(allIndexes, allSizes, 0, u);
    insert = -(newOffset & 3);
}
else if(!resize[u])
{
    // over estimation of the number of bytes to be
    // added to this instruction = 3 - current number
    // of padding bytes = 3 - (3 - u%4) = u%4 = u & 3
    insert = u & 3;
    resize[u] = true;
}
// skips instruction
u = u + 4 - (u & 3);
u += 4*(readInt(b,u+8) - readInt(b,u+4)+1)+12;

```

```

        break;
case ClassWriter.LOOK_INSN:
    if(state == 1)
    {
        // like TABL_INSN
        newOffset =
            getNewOffset(allIndexes, allSizes, 0, u);
        insert = -(newOffset & 3);
    }
    else if(!resize[u])
    {
        // like TABL_INSN
        insert = u & 3;
        resize[u] = true;
    }
    // skips instruction
    u = u + 4 - (u & 3);
    u += 8 * readInt(b, u + 4) + 8;
    break;
case ClassWriter.WIDE_INSN:
    opcode = b[u + 1] & 0xFF;
    if(opcode == Opcodes.IINC)
    {
        u += 6;
    }
    else
    {
        u += 4;
    }
    break;
case ClassWriter.VAR_INSN:
case ClassWriter.SBYTE_INSN:
case ClassWriter.LDC_INSN:
    u += 2;
    break;
case ClassWriter.SHORT_INSN:
case ClassWriter.LDCW_INSN:
case ClassWriter.FIELDORMETH_INSN:
case ClassWriter.TYPE_INSN:
case ClassWriter.IINC_INSN:
    u += 3;
    break;
case ClassWriter.ITFMETH_INSN:
    u += 5;
    break;
    // case ClassWriter.MANA_INSN:
default:
    u += 4;
    break;
}

```

```

        if(insert != 0)
        {
            // adds a new (u, insert) entry in the allIndexes and
            // allSizes arrays
            int[] newIndexes = new int[allIndexes.length + 1];
            int[] newSizes = new int[allSizes.length + 1];
            System.arraycopy(allIndexes,
                0,
                newIndexes,
                0,
                allIndexes.length);
            System.arraycopy(allSizes,0,newSizes,0,allSizes.length);
            newIndexes[allIndexes.length] = u;
            newSizes[allSizes.length] = insert;
            allIndexes = newIndexes;
            allSizes = newSizes;
            if(insert > 0)
            {
                state = 3;
            }
        }
        if(state < 3)
        {
            --state;
        }
    } while(state != 0);

    // 2nd step:
    // copies the bytecode of the method into a new bytevector, updates
    // the offsets, and inserts (or removes) bytes as requested.

    ByteVector newCode = new ByteVector(code.length);

    u = 0;
    while(u < code.length)
    {
        int opcode = b[u] & 0xFF;
        switch(ClassWriter.TYPE[opcode])
        {
            case ClassWriter.NOARG_INSN:
            case ClassWriter.IMPLVAR_INSN:
                newCode.putByte(opcode);
                u += 1;
                break;
            case ClassWriter.LABEL_INSN:
                if(opcode > 201)
                {
                    // changes temporary opcodes 202 to 217 (inclusive),
                    // 218 and 219 to IFEQ ... JSR (inclusive), IFNULL

```

```

// and IFNONNULL
opcode = opcode < 218 ? opcode - 49 : opcode - 20;
label = u + readUnsignedShort(b, u + 1);
}
else
{
label = u + readShort(b, u + 1);
}
newOffset = getNewOffset(allIndexes, allSizes, u, label);
if(resize[u])
{
// replaces GOTO with GOTO_W, JSR with JSR_W and
// IFxxx <l> with IFNOTxxx <l'> GOTO_W <l>, where
// IFNOTxxx is the "opposite" opcode of IFxxx
// (i.e., IFNE for IFEQ) and where <l'> designates
// the instruction just after the GOTO_W.
if(opcode == Opcodes.GOTO)
{
newCode.putByte(200); // GOTO_W
}
else if(opcode == Opcodes.JSR)
{
newCode.putByte(201); // JSR_W
}
else
{
newCode.putByte(opcode <= 166
? ((opcode + 1) ^ 1) - 1
: opcode ^ 1);
newCode.putShort(8); // jump offset
newCode.putByte(200); // GOTO_W
// newOffset now computed from start of GOTO_W
newOffset -= 3;
}
newCode.putInt(newOffset);
}
else
{
newCode.putByte(opcode);
newCode.putShort(newOffset);
}
u += 3;
break;
case ClassWriter.LABELW_INSN:
label = u + readInt(b, u + 1);
newOffset = getNewOffset(allIndexes, allSizes, u, label);
newCode.putByte(opcode);
newCode.putInt(newOffset);
u += 5;
break;
}

```

```
case ClassWriter.TABL_INSN:
    // skips 0 to 3 padding bytes
    v = u;
    u = u + 4 - (v & 3);
    // reads and copies instruction
    newCode.putByte(OpCodes.TABLESWITCH);
    newCode.length += (4 - newCode.length % 4) % 4;
    label = v + readInt(b, u);
    u += 4;
    newOffset = getNewOffset(allIndexes, allSizes, v, label);
    newCode.putInt(newOffset);
    j = readInt(b, u);
    u += 4;
    newCode.putInt(j);
    j = readInt(b, u) - j + 1;
    u += 4;
    newCode.putInt(readInt(b, u - 4));
    for(; j > 0; --j)
    {
        label = v + readInt(b, u);
        u += 4;
        newOffset =
            getNewOffset(allIndexes, allSizes, v, label);
        newCode.putInt(newOffset);
    }
    break;
case ClassWriter.LOOK_INSN:
    // skips 0 to 3 padding bytes
    v = u;
    u = u + 4 - (v & 3);
    // reads and copies instruction
    newCode.putByte(OpCodes.LOOKUPSWITCH);
    newCode.length += (4 - newCode.length % 4) % 4;
    label = v + readInt(b, u);
    u += 4;
    newOffset = getNewOffset(allIndexes, allSizes, v, label);
    newCode.putInt(newOffset);
    j = readInt(b, u);
    u += 4;
    newCode.putInt(j);
    for(; j > 0; --j)
    {
        newCode.putInt(readInt(b, u));
        u += 4;
        label = v + readInt(b, u);
        u += 4;
        newOffset =
            getNewOffset(allIndexes, allSizes, v, label);
        newCode.putInt(newOffset);
    }
```

```

        break;
    case ClassWriter.WIDE_INSN:
        opcode = b[u + 1] & 0xFF;
        if(opcode == Opcodes.IINC)
        {
            newCode.putByteArray(b, u, 6);
            u += 6;
        }
        else
        {
            newCode.putByteArray(b, u, 4);
            u += 4;
        }
        break;
    case ClassWriter.VAR_INSN:
    case ClassWriter.SBYTE_INSN:
    case ClassWriter.LDC_INSN:
        newCode.putByteArray(b, u, 2);
        u += 2;
        break;
    case ClassWriter.SHORT_INSN:
    case ClassWriter.LDCW_INSN:
    case ClassWriter.FIELDORMETH_INSN:
    case ClassWriter.TYPE_INSN:
    case ClassWriter.IINC_INSN:
        newCode.putByteArray(b, u, 3);
        u += 3;
        break;
    case ClassWriter.ITFMETH_INSN:
        newCode.putByteArray(b, u, 5);
        u += 5;
        break;
        // case MANA_INSN:
    default:
        newCode.putByteArray(b, u, 4);
        u += 4;
        break;
    }
}

// recomputes the stack map frames
if(frameCount > 0)
{
    if(compute == FRAMES)
    {
        frameCount = 0;
        stackMap = null;
        previousFrame = null;
        frame = null;
        Frame f = new Frame();

```

```

f.owner = labels;
Type[] args = Type.getArgumentTypes(descriptor);
f.initInputFrame(cw, access, args, maxLocals);
visitFrame(f);
Label l = labels;
while(l != null)
{
/*
 * here we need the original label position. getNewOffset
 * must therefore never have been called for this label.
 */
u = l.position - 3;
if((l.status & Label.STORE) != 0 ||
(u >= 0 && resize[u]))
{
getNewOffset(allIndexes, allSizes, l);
// TODO update offsets in UNINITIALIZED values
visitFrame(l.frame);
}
l = l.successor;
}
}
else
{
/*
 * Resizing an existing stack map frame table is really
 * hard. Not only the table must be parsed to update the
 * offsets, but new frames may be needed for jump
 * instructions that were inserted by this method. And
 * updating the offsets or inserting frames can change
 * the format of the following frames, in case of packed
 * frames. In practice the whole table must be recomputed.
 * For this the frames are marked as potentially invalid.
 * This will cause the whole class to be reread and
 * rewritten with the COMPUTE_FRAMES option (see the
 * ClassWriter.toByteArray method). This is not very
 * efficient but is much easier and requires much less
 * code than any other method I can think of.
*/
cw.invalidFrames = true;
}
}
// updates the exception handler block labels
Handler h = firstHandler;
while(h != null)
{
getNewOffset(allIndexes, allSizes, h.start);
getNewOffset(allIndexes, allSizes, h.end);
getNewOffset(allIndexes, allSizes, h.handler);
h = h.next;
}

```

```

        }
        // updates the instructions addresses in the
        // local var and line number tables
        for(i = 0; i < 2; ++i)
        {
            ByteVector bv = i == 0 ? localVar : localVarType;
            if(bv != null)
            {
                b = bv.data;
                u = 0;
                while(u < bv.length)
                {
                    label = readUnsignedShort(b, u);
                    newOffset = getNewOffset(allIndexes, allSizes, 0, label);
                    writeShort(b, u, newOffset);
                    label += readUnsignedShort(b, u + 2);
                    newOffset = getNewOffset(allIndexes, allSizes, 0, label)
                        - newOffset;
                    writeShort(b, u + 2, newOffset);
                    u += 10;
                }
            }
            if(lineNumber != null)
            {
                b = lineNumber.data;
                u = 0;
                while(u < lineNumber.length)
                {
                    writeShort(b, u, getNewOffset(allIndexes,
                        allSizes,
                        0,
                        readUnsignedShort(b, u)));
                    u += 4;
                }
            }
        }
        // updates the labels of the other attributes
        Attribute attr = cattrs;
        while(attr != null)
        {
            Label[] labels = attr.getLabels();
            if(labels != null)
            {
                for(i = labels.length - 1; i >= 0; --i)
                {
                    getNewOffset(allIndexes, allSizes, labels[i]);
                }
            }
            attr = attr.next;
        }
    }
}

```

```
// replaces old bytecodes with new ones
    code = newCode;
}

/**
 * Reads an unsigned short value in the given byte array.
 *
 * @param b      a byte array.
 * @param index the start index of the value to be read.
 * @return the read value.
 */
static int readUnsignedShort(final byte[] b, final int index){
    return ((b[index] & 0xFF) << 8) | (b[index + 1] & 0xFF);
}

/**
 * Reads a signed short value in the given byte array.
 *
 * @param b      a byte array.
 * @param index the start index of the value to be read.
 * @return the read value.
 */
static short readShort(final byte[] b, final int index){
    return (short) (((b[index] & 0xFF) << 8) | (b[index + 1] & 0xFF));
}

/**
 * Reads a signed int value in the given byte array.
 *
 * @param b      a byte array.
 * @param index the start index of the value to be read.
 * @return the read value.
 */
static int readInt(final byte[] b, final int index){
    return ((b[index] & 0xFF) << 24) | ((b[index + 1] & 0xFF) << 16)
        | ((b[index + 2] & 0xFF) << 8) | (b[index + 3] & 0xFF);
}

/**
 * Writes a short value in the given byte array.
 *
 * @param b      a byte array.
 * @param index where the first byte of the short value must be written.
 * @param s      the value to be written in the given byte array.
 */
static void writeShort(final byte[] b, final int index, final int s){
    b[index] = (byte) (s >>> 8);
    b[index + 1] = (byte) s;
}
```

```
/*
 * Computes the future value of a bytecode offset. <p> Note: it is
 * possible to have several entries for the same instruction in the
 * <tt>indexes</tt> and <tt>sizes</tt>: two entries (index=a,size=b)
 * and (index=a,size=b') are equivalent to a single entry
 * (index=a,size=b+b').
 *
 * @param indexes current positions of the instructions to be resized.
 *                 Each instruction must be designated by the index of
 *                 its <i>last</i> byte, plus one (or, in other words, by
 *                 the index of the <i>first</i> byte of the <i>next</i>
 *                 instruction).
 * @param sizes   the number of bytes to be <i>added</i> to the above
 *                 instructions. More precisely, for each
 *                 i < <tt>len</tt>, <tt>sizes</tt>[i] bytes will be
 *                 added at the end of the instruction designated by
 *                 <tt>indexes</tt>[i] or, if <tt>sizes</tt>[i] is
 *                 negative, the <i>last</i> |<tt>sizes[i]</tt>|
 *                 bytes of the instruction will be removed (the
 *                 instruction size <i>must not</i> become negative or
 *                 null).
 * @param begin   index of the first byte of the source instruction.
 * @param end     index of the first byte of the target instruction.
 * @return the future value of the given bytecode offset.
 */
static int getNewOffset(
    final int[] indexes,
    final int[] sizes,
    final int begin,
    final int end){
    int offset = end - begin;
    for(int i = 0; i < indexes.length; ++i)
    {
        if(begin < indexes[i] && indexes[i] <= end)
        {
            // forward jump
            offset += sizes[i];
        }
        else if(end < indexes[i] && indexes[i] <= begin)
        {
            // backward jump
            offset -= sizes[i];
        }
    }
    return offset;
}

/**
 * Updates the offset of the given label.

```

```

*
* @param indexes current positions of the instructions to be resized.
*                 Each instruction must be designated by the index of
*                 its <i>last</i> byte, plus one (or, in other words,
*                 by the index of the <i>first</i> byte of the
*                 <i>next</i> instruction).
* @param sizes   the number of bytes to be <i>added</i> to the above
*                 instructions. More precisely, for each
*                 i < tt>len</tt>, <tt>sizes</tt>[i] bytes will be
*                 added at the end of the instruction designated by
*                 <tt>indexes</tt>[i] or, if <tt>sizes</tt>[i] is
*                 negative, the <i>last</i> | <tt>sizes[i]</tt>|
*                 bytes of the instruction will be removed (the
*                 instruction size <i>must not</i> become negative
*                 or null).
* @param label   the label whose offset must be updated.
*/
static void getNewOffset(
    final int[] indexes,
    final int[] sizes,
    final Label label){
    if((label.status & Label.RESIZED) == 0)
    {
        label.position = getNewOffset(indexes, sizes, 0, label.position);
        label.status |= Label.RESIZED;
    }
}
}

```

7.19 Opcodes.java

— Opcodes.java —

```

\getchunk{France Telecom Copyright}
package clojure.asm;

/**
 * Defines the JVM opcodes, access flags and array type codes. This
 * interface does not define all the JVM opcodes because some opcodes
 * are automatically handled. For example, the xLOAD and xSTORE opcodes
 * are automatically replaced by xLOAD_n and xSTORE_n opcodes when
 * possible. The xLOAD_n and xSTORE_n opcodes are therefore not
 * defined in this interface. Likewise for LDC, automatically replaced
 * by LDC_W or LDC2_W when necessary, WIDE, GOTO_W and JSR_W.
 */

```

```

* @author Eric Bruneton
* @author Eugene Kuleshov
*/
public interface Opcodes{

    // versions

    int V1_1 = 3 << 16 | 45;
    int V1_2 = 0 << 16 | 46;
    int V1_3 = 0 << 16 | 47;
    int V1_4 = 0 << 16 | 48;
    int V1_5 = 0 << 16 | 49;
    int V1_6 = 0 << 16 | 50;

    // access flags

    int ACC_PUBLIC = 0x0001; // class, field, method
    int ACC_PRIVATE = 0x0002; // class, field, method
    int ACC_PROTECTED = 0x0004; // class, field, method
    int ACC_STATIC = 0x0008; // field, method
    int ACC_FINAL = 0x0010; // class, field, method
    int ACC_SUPER = 0x0020; // class
    int ACC_SYNCHRONIZED = 0x0020; // method
    int ACC_VOLATILE = 0x0040; // field
    int ACC_BRIDGE = 0x0040; // method
    int ACC_VARARGS = 0x0080; // method
    int ACC_TRANSIENT = 0x0080; // field
    int ACC_NATIVE = 0x0100; // method
    int ACC_INTERFACE = 0x0200; // class
    int ACC_ABSTRACT = 0x0400; // class, method
    int ACC_STRICT = 0x0800; // method
    int ACC_SYNTHETIC = 0x1000; // class, field, method
    int ACC_ANNOTATION = 0x2000; // class
    int ACC_ENUM = 0x4000; // class(?) field inner

    // ASM specific pseudo access flags

    int ACC_DEPRECATED = 131072; // class, field, method

    // types for NEWARRAY

    int T_BOOLEAN = 4;
    int T_CHAR = 5;
    int T_FLOAT = 6;
    int T_DOUBLE = 7;
    int T_BYTE = 8;
    int T_SHORT = 9;
    int T_INT = 10;
    int T_LONG = 11;
}

```

```
// stack map frame types

/**
 * Represents an expanded frame. See {@link ClassReader#EXPAND_FRAMES}.
 */
int F_NEW = -1;

/**
 * Represents a compressed frame with complete frame data.
 */
int F_FULL = 0;

/**
 * Represents a compressed frame where locals are the same as the
 * locals in the previous frame, except that additional 1-3 locals
 * are defined, and with an empty stack.
 */
int F_APPEND = 1;

/**
 * Represents a compressed frame where locals are the same as the
 * locals in the previous frame, except that the last 1-3 locals are
 * absent and with an empty stack.
 */
int F_CHOP = 2;

/**
 * Represents a compressed frame with exactly the same locals as the
 * previous frame and with an empty stack.
 */
int F_SAME = 3;

/**
 * Represents a compressed frame with exactly the same locals as the
 * previous frame and with a single value on the stack.
 */
int F_SAME1 = 4;

Integer TOP = new Integer(0);
Integer INTEGER = new Integer(1);
Integer FLOAT = new Integer(2);
Integer DOUBLE = new Integer(3);
Integer LONG = new Integer(4);
Integer NULL = new Integer(5);
Integer UNINITIALIZED_THIS = new Integer(6);

// opcodes // visit method (- = idem)

int NOP = 0; // visitInsn
int ACONST_NULL = 1; // -
```

```
int ICONST_M1 = 2; // -
int ICONST_0 = 3; // -
int ICONST_1 = 4; // -
int ICONST_2 = 5; // -
int ICONST_3 = 6; // -
int ICONST_4 = 7; // -
int ICONST_5 = 8; // -
int LCONST_0 = 9; // -
int LCONST_1 = 10; // -
int FCONST_0 = 11; // -
int FCONST_1 = 12; // -
int FCONST_2 = 13; // -
int DCONST_0 = 14; // -
int DCONST_1 = 15; // -
int BIPUSH = 16; // visitIntInsn
int SIPUSH = 17; // -
int LDC = 18; // visitLdcInsn
// int LDC_W = 19; // -
// int LDC2_W = 20; // -
int ILOAD = 21; // visitVarInsn
int LLOAD = 22; // -
int FLOAD = 23; // -
int DLOAD = 24; // -
int ALOAD = 25; // -
// int ILOAD_0 = 26; // -
// int ILOAD_1 = 27; // -
// int ILOAD_2 = 28; // -
// int ILOAD_3 = 29; // -
// int LLOAD_0 = 30; // -
// int LLOAD_1 = 31; // -
// int LLOAD_2 = 32; // -
// int LLOAD_3 = 33; // -
// int FLOAD_0 = 34; // -
// int FLOAD_1 = 35; // -
// int FLOAD_2 = 36; // -
// int FLOAD_3 = 37; // -
// int DLOAD_0 = 38; // -
// int DLOAD_1 = 39; // -
// int DLOAD_2 = 40; // -
// int DLOAD_3 = 41; // -
// int ALOAD_0 = 42; // -
// int ALOAD_1 = 43; // -
// int ALOAD_2 = 44; // -
// int ALOAD_3 = 45; // -
int IALOAD = 46; // visitInsn
int LALOAD = 47; // -
int FALOAD = 48; // -
int DALOAD = 49; // -
int AALOAD = 50; // -
int BALOAD = 51; // -
```

```
int CALOAD = 52; // -
int SALOAD = 53; // -
int ISTORE = 54; // visitVarInsn
int LSTORE = 55; // -
int FSTORE = 56; // -
int DSTORE = 57; // -
int ASTORE = 58; // -
// int ISTORE_0 = 59; // -
// int ISTORE_1 = 60; // -
// int ISTORE_2 = 61; // -
// int ISTORE_3 = 62; // -
// int LSTORE_0 = 63; // -
// int LSTORE_1 = 64; // -
// int LSTORE_2 = 65; // -
// int LSTORE_3 = 66; // -
// int FSTORE_0 = 67; // -
// int FSTORE_1 = 68; // -
// int FSTORE_2 = 69; // -
// int FSTORE_3 = 70; // -
// int DSTORE_0 = 71; // -
// int DSTORE_1 = 72; // -
// int DSTORE_2 = 73; // -
// int DSTORE_3 = 74; // -
// int ASTORE_0 = 75; // -
// int ASTORE_1 = 76; // -
// int ASTORE_2 = 77; // -
// int ASTORE_3 = 78; // -
int IASTORE = 79; // visitInsn
int LASTORE = 80; // -
int FASTORE = 81; // -
int DASTORE = 82; // -
int AASTORE = 83; // -
int BASTORE = 84; // -
int CASTORE = 85; // -
int SASTORE = 86; // -
int POP = 87; // -
int POP2 = 88; // -
int DUP = 89; // -
int DUP_X1 = 90; // -
int DUP_X2 = 91; // -
int DUP2 = 92; // -
int DUP2_X1 = 93; // -
int DUP2_X2 = 94; // -
int SWAP = 95; // -
int IADD = 96; // -
int LADD = 97; // -
int FADD = 98; // -
int DADD = 99; // -
int ISUB = 100; // -
int LSUB = 101; // -
```

```
int FSUB = 102; // -
int DSUB = 103; // -
int IMUL = 104; // -
int LMUL = 105; // -
int FMUL = 106; // -
int DMUL = 107; // -
int IDIV = 108; // -
int LDIV = 109; // -
int FDIV = 110; // -
int DDIV = 111; // -
int IREM = 112; // -
int LREM = 113; // -
int FREM = 114; // -
int DREM = 115; // -
int INEG = 116; // -
int LNEG = 117; // -
int FNEG = 118; // -
int DNEG = 119; // -
int ISHL = 120; // -
int LSHL = 121; // -
int ISHR = 122; // -
int LSHR = 123; // -
int IUSHR = 124; // -
int LUSHR = 125; // -
int IAND = 126; // -
int LAND = 127; // -
int IOR = 128; // -
int LOR = 129; // -
int IXOR = 130; // -
int LXOR = 131; // -
int IINC = 132; // visitIincInsn
int I2L = 133; // visitInsn
int I2F = 134; // -
int I2D = 135; // -
int L2I = 136; // -
int L2F = 137; // -
int L2D = 138; // -
int F2I = 139; // -
int F2L = 140; // -
int F2D = 141; // -
int D2I = 142; // -
int D2L = 143; // -
int D2F = 144; // -
int I2B = 145; // -
int I2C = 146; // -
int I2S = 147; // -
int LCMP = 148; // -
int FCMPL = 149; // -
int FCMPG = 150; // -
int DCMPL = 151; // -
```

```
int DCMPG = 152; // -
int IFEQ = 153; // visitJumpInsn
int IFNE = 154; // -
int IFLT = 155; // -
int IFGE = 156; // -
int IFGT = 157; // -
int IFLE = 158; // -
int IF_ICMPEQ = 159; // -
int IF_ICMPNE = 160; // -
int IF_ICMPLT = 161; // -
int IF_ICMPGE = 162; // -
int IF_ICMPGT = 163; // -
int IF_ICMPLE = 164; // -
int IF_ACMPEQ = 165; // -
int IF_ACMPNE = 166; // -
int GOTO = 167; // -
int JSR = 168; // -
int RET = 169; // visitVarInsn
int TABLESWITCH = 170; // visitTableSwitchInsn
int LOOKUPSWITCH = 171; // visitLookupSwitch
intIRETURN = 172; // visitInsn
intLRETURN = 173; // -
intFRETURN = 174; // -
intDRETURN = 175; // -
intARETURN = 176; // -
intRETURN = 177; // -
intGETSTATIC = 178; // visitFieldInsn
intPUTSTATIC = 179; // -
intGETFIELD = 180; // -
intPUTFIELD = 181; // -
intINVOKEVIRTUAL = 182; // visitMethodInsn
intINVOKEPECIAL = 183; // -
intINVOKESTATIC = 184; // -
intINVOKEINTERFACE = 185; // -
// int UNUSED = 186; // NOT VISITED
intNEW = 187; // visitTypeInsn
intNEWARRAY = 188; // visitIntInsn
intANEWARRAY = 189; // visitTypeInsn
intARRAYLENGTH = 190; // visitInsn
intATHROW = 191; // -
intCHECKCAST = 192; // visitTypeInsn
intINSTANCEOF = 193; // -
intMONITORENTER = 194; // visitInsn
intMONITOREXIT = 195; // -
// int WIDE = 196; // NOT VISITED
intMULTIANEWARRAY = 197; // visitMultiANewArrayInsn
intIFNULL = 198; // visitJumpInsn
intIFNONNULL = 199; // -
// int GOTO_W = 200; // -
// int JSR_W = 201; // -
```

}

7.20 Type.java

— Type.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

/**
 * A Java type. This class can be used to make it easier to manipulate
 * type and method descriptors.
 *
 * @author Eric Bruneton
 * @author Chris Nokleberg
 */
public class Type{

    /**
     * The sort of the <tt>void</tt> type. See {@link #getSort getSort}.
     */
    public final static int VOID = 0;

    /**
     * The sort of the <tt>boolean</tt> type. See {@link #getSort getSort}.
     */
    public final static int BOOLEAN = 1;

    /**
     * The sort of the <tt>char</tt> type. See {@link #getSort getSort}.
     */
    public final static int CHAR = 2;

    /**
     * The sort of the <tt>byte</tt> type. See {@link #getSort getSort}.
     */
    public final static int BYTE = 3;

    /**
     * The sort of the <tt>short</tt> type. See {@link #getSort getSort}.
     */
    public final static int SHORT = 4;
```

```
/**  
 * The sort of the <tt>int</tt> type. See {@link #getSort getSort}.  
 */  
public final static int INT = 5;  
  
/**  
 * The sort of the <tt>float</tt> type. See {@link #getSort getSort}.  
 */  
public final static int FLOAT = 6;  
  
/**  
 * The sort of the <tt>long</tt> type. See {@link #getSort getSort}.  
 */  
public final static int LONG = 7;  
  
/**  
 * The sort of the <tt>double</tt> type. See {@link #getSort getSort}.  
 */  
public final static int DOUBLE = 8;  
  
/**  
 * The sort of array reference types. See {@link #getSort getSort}.  
 */  
public final static int ARRAY = 9;  
  
/**  
 * The sort of object reference type. See {@link #getSort getSort}.  
 */  
public final static int OBJECT = 10;  
  
/**  
 * The <tt>void</tt> type.  
 */  
public final static Type VOID_TYPE = new Type(VOID);  
  
/**  
 * The <tt>boolean</tt> type.  
 */  
public final static Type BOOLEAN_TYPE = new Type(BOOLEAN);  
  
/**  
 * The <tt>char</tt> type.  
 */  
public final static Type CHAR_TYPE = new Type(CHAR);  
  
/**  
 * The <tt>byte</tt> type.  
 */  
public final static Type BYTE_TYPE = new Type(BYTE);
```

```
/**  
 * The <tt>short</tt> type.  
 */  
public final static Type SHORT_TYPE = new Type(SHORT);  
  
/**  
 * The <tt>int</tt> type.  
 */  
public final static Type INT_TYPE = new Type(INT);  
  
/**  
 * The <tt>float</tt> type.  
 */  
public final static Type FLOAT_TYPE = new Type(FLOAT);  
  
/**  
 * The <tt>long</tt> type.  
 */  
public final static Type LONG_TYPE = new Type(LONG);  
  
/**  
 * The <tt>double</tt> type.  
 */  
public final static Type DOUBLE_TYPE = new Type(DOUBLE);  
  
// -----  
// Fields  
// -----  
  
/**  
 * The sort of this Java type.  
 */  
private final int sort;  
  
/**  
 * A buffer containing the descriptor of this Java type. This field  
 * is only used for reference types.  
 */  
private char[] buf;  
  
/**  
 * The offset of the descriptor of this Java type in {@link #buf buf}.  
 * This field is only used for reference types.  
 */  
private int off;  
  
/**  
 * The length of the descriptor of this Java type.  
 */
```

```
private int len;

// -----
// Constructors
// -----

/** 
 * Constructs a primitive type.
 *
 * @param sort the sort of the primitive type to be constructed.
 */
private Type(final int sort){
    this.sort = sort;
    this.len = 1;
}

/** 
 * Constructs a reference type.
 *
 * @param sort the sort of the reference type to be constructed.
 * @param buf a buffer containing the descriptor of the previous type.
 * @param off the offset of this descriptor in the previous buffer.
 * @param len the length of this descriptor.
 */
private Type(final int sort,
            final char[] buf,
            final int off,
            final int len){
    this.sort = sort;
    this.buf = buf;
    this.off = off;
    this.len = len;
}

/** 
 * Returns the Java type corresponding to the given type descriptor.
 *
 * @param typeDescriptor a type descriptor.
 * @return the Java type corresponding to the given type descriptor.
 */
public static Type getType(final String typeDescriptor){
    return getType(typeDescriptor.toCharArray(), 0);
}

/** 
 * Returns the Java type corresponding to the given class.
 *
 * @param c a class.
 * @return the Java type corresponding to the given class.
 */

```

```

public static Type getType(final Class c){
    if(c.isPrimitive())
    {
        if(c == Integer.TYPE)
        {
            return INT_TYPE;
        }
        else if(c == Void.TYPE)
        {
            return VOID_TYPE;
        }
        else if(c == Boolean.TYPE)
        {
            return BOOLEAN_TYPE;
        }
        else if(c == Byte.TYPE)
        {
            return BYTE_TYPE;
        }
        else if(c == Character.TYPE)
        {
            return CHAR_TYPE;
        }
        else if(c == Short.TYPE)
        {
            return SHORT_TYPE;
        }
        else if(c == Double.TYPE)
        {
            return DOUBLE_TYPE;
        }
        else if(c == Float.TYPE)
        {
            return FLOAT_TYPE;
        }
        else /* if (c == Long.TYPE) */
        {
            return LONG_TYPE;
        }
    }
    else
    {
        return getType(getDescriptor(c));
    }
}

/**
 * Returns the {@link Type#OBJECT} type for the given internal class
 * name. This is a shortcut method for
 * <code>Type.getType("L"+name+";")</code>.

```

```
* <i>Note that opposed to {@link Type#getType(String)}, this method
* takes internal class names and not class descriptor.</i>
*
* @param name an internal class name.
* @return the the {@link Type#OBJECT} type for the given class name.
*/
public static Type getObjectType(String name){
    int l = name.length();
    char[] buf = new char[l + 2];
    buf[0] = 'L';
    buf[l + 1] = ';';
    name.getChars(0, l, buf, 1);
    return new Type(OBJECT, buf, 0, l + 2);
}

/**
 * Returns the Java types corresponding to the argument types of the
 * given method descriptor.
 *
 * @param methodDescriptor a method descriptor.
 * @return the Java types corresponding to the argument types of the
 *         given method descriptor.
 */
public static Type[] getArgumentsTypes(final String methodDescriptor){
    char[] buf = methodDescriptor.toCharArray();
    int off = 1;
    int size = 0;
    while(true)
    {
        char car = buf[off++];
        if(car == ')')
        {
            break;
        }
        else if(car == 'L')
        {
            while(buf[off++] != ';')
            {
            }
            ++size;
        }
        else if(car != '[')
        {
            ++size;
        }
    }
    Type[] args = new Type[size];
    off = 1;
    size = 0;
    while(buf[off] != ')')
```

```
    {
        args[size] = getType(buf, off);
        off += args[size].len;
        size += 1;
    }
    return args;
}

/**
 * Returns the Java types corresponding to the argument types of the
 * given method.
 *
 * @param method a method.
 * @return the Java types corresponding to the argument types of the
 *         given method.
 */
public static Type[] getArgumentTypes(final Method method){
    Class[] classes = method.getParameterTypes();
    Type[] types = new Type[classes.length];
    for(int i = classes.length - 1; i >= 0; --i)
    {
        types[i] = getType(classes[i]);
    }
    return types;
}

/**
 * Returns the Java type corresponding to the return type of the given
 * method descriptor.
 *
 * @param methodDescriptor a method descriptor.
 * @return the Java type corresponding to the return type of the given
 *         method descriptor.
 */
public static Type getReturnType(final String methodDescriptor){
    char[] buf = methodDescriptor.toCharArray();
    return getType(buf, methodDescriptor.indexOf(')') + 1);
}

/**
 * Returns the Java type corresponding to the return type of the given
 * method.
 *
 * @param method a method.
 * @return the Java type corresponding to the return type of the given
 *         method.
 */
public static Type getReturnType(final Method method){
    return getType(method.getReturnType());
}
```

```
/**
 * Returns the Java type corresponding to the given type descriptor.
 *
 * @param buf a buffer containing a type descriptor.
 * @param off the offset of this descriptor in the previous buffer.
 * @return the Java type corresponding to the given type descriptor.
 */
private static Type getType(final char[] buf, final int off){
    int len;
    switch(buf[off])
    {
        case'V':
            return VOID_TYPE;
        case'Z':
            return BOOLEAN_TYPE;
        case'C':
            return CHAR_TYPE;
        case'B':
            return BYTE_TYPE;
        case'S':
            return SHORT_TYPE;
        case'I':
            return INT_TYPE;
        case'F':
            return FLOAT_TYPE;
        case'J':
            return LONG_TYPE;
        case'D':
            return DOUBLE_TYPE;
        case '[':
            len = 1;
            while(buf[off + len] == '[')
            {
                ++len;
            }
            if(buf[off + len] == 'L')
            {
                ++len;
                while(buf[off + len] != ';')
                {
                    ++len;
                }
            }
            return new Type(ARRAY, buf, off, len + 1);
        // case 'L':
        default:
            len = 1;
            while(buf[off + len] != ';')
            {

```

```

        ++len;
    }
    return new Type(OBJECT, buf, off, len + 1);
}
}

// -----
// Accessors
// -----


/***
 * Returns the sort of this Java type.
 *
 * @return {@link #VOID VOID}, {@link #BOOLEAN BOOLEAN},
 *         {@link #CHAR CHAR}, {@link #BYTE BYTE}, {@link #SHORT SHORT},
 *         {@link #INT INT}, {@link #FLOAT FLOAT}, {@link #LONG LONG},
 *         {@link #DOUBLE DOUBLE}, {@link #ARRAY ARRAY} or
 *         {@link #OBJECT OBJECT}.
 */
public int getSort(){
    return sort;
}

/***
 * Returns the number of dimensions of this array type. This method
 * should only be used for an array type.
 *
 * @return the number of dimensions of this array type.
 */
public int getDimensions(){
    int i = 1;
    while(buf[off + i] == '[')
    {
        ++i;
    }
    return i;
}

/***
 * Returns the type of the elements of this array type. This method
 * should only be used for an array type.
 *
 * @return Returns the type of the elements of this array type.
 */
public Type getElementType(){
    return getType(buf, off + getDimensions());
}

/***
 * Returns the name of the class corresponding to this type.
 */

```

```

*
* @return the fully qualified name of the class corresponding to
* this type.
*/
public String getClassName(){
    switch(sort)
    {
        case VOID:
            return "void";
        case BOOLEAN:
            return "boolean";
        case CHAR:
            return "char";
        case BYTE:
            return "byte";
        case SHORT:
            return "short";
        case INT:
            return "int";
        case FLOAT:
            return "float";
        case LONG:
            return "long";
        case DOUBLE:
            return "double";
        case ARRAY:
            StringBuffer b =
                new StringBuffer(getElementType().getClassName());
            for(int i = getDimensions(); i > 0; --i)
            {
                b.append("[]");
            }
            return b.toString();
            // case OBJECT:
        default:
            return new String(buf, off + 1, len - 2).replace('/', '.');
    }
}

/**
 * Returns the internal name of the class corresponding to this object
 * type. The internal name of a class is its fully qualified name,
 * where '.' are replaced by '/'. This method should only be used for
 * an object type.
 *
 * @return the internal name of the class corresponding to this object
 * type.
*/
public String getInternalName(){
    return new String(buf, off + 1, len - 2);
}

```

```
}

// -----
// Conversion to type descriptors
// -----


/***
 * Returns the descriptor corresponding to this Java type.
 *
 * @return the descriptor corresponding to this Java type.
 */
public String getDescriptor(){
    StringBuffer buf = new StringBuffer();
    getDescriptor(buf);
    return buf.toString();
}

/***
 * Returns the descriptor corresponding to the given argument and return
 * types.
 *
 * @param returnType the return type of the method.
 * @param argumentTypes the argument types of the method.
 * @return the descriptor corresponding to the given argument and return
 *         types.
 */
public static String getMethodDescriptor(
    final Type returnType,
    final Type[] argumentTypes){
    StringBuffer buf = new StringBuffer();
    buf.append('(');
    for(int i = 0; i < argumentTypes.length; ++i)
    {
        argumentTypes[i].getDescriptor(buf);
    }
    buf.append(')');
    returnType.getDescriptor(buf);
    return buf.toString();
}

/***
 * Appends the descriptor corresponding to this Java type to the given
 * string buffer.
 *
 * @param buf the string buffer to which the descriptor must be appended.
 */
private void getDescriptor(final StringBuffer buf){
    switch(sort)
    {
        case VOID:
```

```

        buf.append('V');
        return;
    case BOOLEAN:
        buf.append('Z');
        return;
    case CHAR:
        buf.append('C');
        return;
    case BYTE:
        buf.append('B');
        return;
    case SHORT:
        buf.append('S');
        return;
    case INT:
        buf.append('I');
        return;
    case FLOAT:
        buf.append('F');
        return;
    case LONG:
        buf.append('J');
        return;
    case DOUBLE:
        buf.append('D');
        return;
    // case ARRAY:
    // case OBJECT:
    default:
        buf.append(this.buf, off, len);
    }
}

// -----
// Direct conversion from classes to type descriptors,
// without intermediate Type objects
// -----


/**
 * Returns the internal name of the given class. The internal name of a
 * class is its fully qualified name, where '.' are replaced by '/'.
 *
 * @param c an object class.
 * @return the internal name of the given class.
 */
public static String getInternalName(final Class c){
    return c.getName().replace('.', '/');
}

/**

```

```
* Returns the descriptor corresponding to the given Java type.  
*  
* @param c an object class, a primitive class or an array class.  
* @return the descriptor corresponding to the given class.  
*/  
public static String getDescriptor(final Class c){  
    StringBuffer buf = new StringBuffer();  
    getDescriptor(buf, c);  
    return buf.toString();  
}  
  
/**  
 * Returns the descriptor corresponding to the given constructor.  
 *  
 * @param c a {@link Constructor} object.  
 * @return the descriptor of the given constructor.  
 */  
public static String getConstructorDescriptor(final Constructor c){  
    Class[] parameters = c.getParameterTypes();  
    StringBuffer buf = new StringBuffer();  
    buf.append('(');  
    for(int i = 0; i < parameters.length; ++i)  
    {  
        getDescriptor(buf, parameters[i]);  
    }  
    return buf.append(")V").toString();  
}  
  
/**  
 * Returns the descriptor corresponding to the given method.  
 *  
 * @param m a {@link Method} object.  
 * @return the descriptor of the given method.  
 */  
public static String getMethodDescriptor(final Method m){  
    Class[] parameters = m.getParameterTypes();  
    StringBuffer buf = new StringBuffer();  
    buf.append('(');  
    for(int i = 0; i < parameters.length; ++i)  
    {  
        getDescriptor(buf, parameters[i]);  
    }  
    buf.append(')');  
    getDescriptor(buf, m.getReturnType());  
    return buf.toString();  
}  
  
/**  
 * Appends the descriptor of the given class to the given string buffer.  
 */
```

```
* @param buf the string buffer to which the descriptor must be appended.
* @param c   the class whose descriptor must be computed.
*/
private static void getDescriptor(final StringBuffer buf, final Class c){
    Class d = c;
    while(true)
    {
        if(d.isPrimitive())
        {
            char car;
            if(d == Integer.TYPE)
            {
                car = 'I';
            }
            else if(d == Void.TYPE)
            {
                car = 'V';
            }
            else if(d == Boolean.TYPE)
            {
                car = 'Z';
            }
            else if(d == Byte.TYPE)
            {
                car = 'B';
            }
            else if(d == Character.TYPE)
            {
                car = 'C';
            }
            else if(d == Short.TYPE)
            {
                car = 'S';
            }
            else if(d == Double.TYPE)
            {
                car = 'D';
            }
            else if(d == Float.TYPE)
            {
                car = 'F';
            }
            else /* if (d == Long.TYPE) */
            {
                car = 'J';
            }
            buf.append(car);
            return;
        }
        else if(d.isArray())
```

```

        {
            buf.append('[');
            d = d.getComponentType();
        }
    else
    {
        buf.append('L');
        String name = d.getName();
        int len = name.length();
        for(int i = 0; i < len; ++i)
        {
            char car = name.charAt(i);
            buf.append(car == '.' ? '/' : car);
        }
        buf.append(']');
        return;
    }
}

// -----
// Corresponding size and opcodes
// -----


/***
 * Returns the size of values of this type.
 *
 * @return the size of values of this type, i.e., 2 for <tt>long</tt> and
 *         <tt>double</tt>, and 1 otherwise.
 */
public int getSize(){
    return sort == LONG || sort == DOUBLE ? 2 : 1;
}

/***
 * Returns a JVM instruction opcode adapted to this Java type.
 *
 * @param opcode a JVM instruction opcode. This opcode must be one of
 *               ILOAD, ISTORE, IALOAD, IASTORE, IADD, ISUB, IMUL,
 *               IDIV, IREM, INEG, ISHL, ISHR, IAND, IOR,
 *               IXOR and IRETURN.
 * @return an opcode that is similar to the given opcode, but adapted to
 *         this Java type. For example, if this type is <tt>float</tt>
 *         and <tt>opcode</tt> is IRETURN, this method returns FRETURN.
 */
public int getOpcode(final int opcode){
    if(opcode == Opcodes.IALOAD || opcode == Opcodes.IASTORE)
    {
        switch(sort)
        {

```

```
        case BOOLEAN:
        case BYTE:
            return opcode + 5;
        case CHAR:
            return opcode + 6;
        case SHORT:
            return opcode + 7;
        case INT:
            return opcode;
        case FLOAT:
            return opcode + 2;
        case LONG:
            return opcode + 1;
        case DOUBLE:
            return opcode + 3;
            // case ARRAY:
            // case OBJECT:
        default:
            return opcode + 4;
    }
}
else
{
    switch(sort)
    {
        case VOID:
            return opcode + 5;
        case BOOLEAN:
        case CHAR:
        case BYTE:
        case SHORT:
        case INT:
            return opcode;
        case FLOAT:
            return opcode + 2;
        case LONG:
            return opcode + 1;
        case DOUBLE:
            return opcode + 3;
            // case ARRAY:
            // case OBJECT:
        default:
            return opcode + 4;
    }
}
}

// -----
// Equals, hashCode and toString
// -----
```

```

/**
 * Tests if the given object is equal to this type.
 *
 * @param o the object to be compared to this type.
 * @return <tt>true</tt> if the given object is equal to this type.
 */
public boolean equals(final Object o){
    if(this == o)
        {
            return true;
        }
    if(!(o instanceof Type))
        {
            return false;
        }
    Type t = (Type) o;
    if(sort != t.sort)
        {
            return false;
        }
    if(sort == Type.OBJECT || sort == Type.ARRAY)
        {
            if(len != t.len)
                {
                    return false;
                }
            for(int i = off, j = t.off, end = i + len; i < end; i++, j++)
                {
                    if(buf[i] != t.buf[j])
                        {
                            return false;
                        }
                }
            }
        return true;
}

/**
 * Returns a hash code value for this type.
 *
 * @return a hash code value for this type.
 */
public int hashCode(){
    int hc = 13 * sort;
    if(sort == Type.OBJECT || sort == Type.ARRAY)
        {
            for(int i = off, end = i + len; i < end; i++)
                {
                    hc = 17 * (hc + buf[i]);
                }
        }
}

```

```
        }
    }
    return hc;
}

/**
 * Returns a string representation of this type.
 *
 * @return the descriptor of this type.
 */
public String toString(){
    return getDescriptor();
}
}
```

—————

Chapter 8

jvm/clojure/asm/commons

8.1 AdviceAdapter.java

(Opcodes [387]) (GeneratorAdapter [453])
— AdviceAdapter.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm.commons;

import java.util.ArrayList;
import java.util.HashMap;

import clojure.asm.Label;
import clojure.asm.MethodVisitor;
import clojure.asm.Opcodes;
import clojure.asm.Type;

/**
 * A {@link clojure.asm.MethodAdapter} to insert before, after and around
 * advices in methods and constructors. <p> The behavior for constructors
 * is like this: <ol>
 * <p/>
 * <li>as long as the INVOKESTATIC for the object initialization has
 * not been reached, every bytecode instruction is dispatched in the
 * ctor code visitor</li>
 * <p/>
 * <li>when this one is reached, it is only added in the ctor code
 * visitor and a JP invoke is added</li>
 * <p/>
 * <li>after that, only the other code visitor receives the
 * instructions</li>
 * <p/>
 * </ol>
```

```
*  
* @author Eugene Kuleshov  
* @author Eric Bruneton  
*/  
public abstract class AdviceAdapter  
    extends GeneratorAdapter implements Opcodes{  
private static final Object THIS = new Object();  
private static final Object OTHER = new Object();  
  
protected int methodAccess;  
protected String methodDesc;  
  
private boolean constructor;  
private boolean superInitialized;  
private ArrayList stackFrame;  
private HashMap branches;  
  
/**  
 * Creates a new {@link AdviceAdapter}.  
 *  
 * @param mv      the method visitor to which this adapter delegates  
 *                calls.  
 * @param access  the method's access flags (see {@link Opcodes}).  
 * @param name    the method's name.  
 * @param desc    the method's descriptor (see {@link Type Type}).  
 */  
public AdviceAdapter(  
    final MethodVisitor mv,  
    final int access,  
    final String name,  
    final String desc){  
super(mv, access, name, desc);  
methodAccess = access;  
methodDesc = desc;  
  
constructor = "<init>".equals(name);  
}  
  
public void visitCode(){  
mv.visitCode();  
if(!constructor)  
{  
    superInitialized = true;  
    onMethodEnter();  
}  
else  
{  
    stackFrame = new ArrayList();  
    branches = new HashMap();  
}
```

```
}

public void visitLabel(final Label label){
    mv.visitLabel(label);

    if(constructor && branches != null)
    {
        ArrayList frame = (ArrayList) branches.get(label);
        if(frame != null)
        {
            stackFrame = frame;
            branches.remove(label);
        }
    }
}

public void visitInsn(final int opcode){
    if(constructor)
    {
        switch(opcode)
        {
            case RETURN: // empty stack
                onMethodExit(opcode);
                break;

            case IRETURN: // 1 before n/a after
            case FRETURN: // 1 before n/a after
            case ARETURN: // 1 before n/a after
            case ATHROW: // 1 before n/a after
                popValue();
                popValue();
                onMethodExit(opcode);
                break;

            case LRETURN: // 2 before n/a after
            case DRETURN: // 2 before n/a after
                popValue();
                popValue();
                onMethodExit(opcode);
                break;

            case NOP:
            case LALOAD: // remove 2 add 2
            case DALOAD: // remove 2 add 2
            case LNEG:
            case DNEG:
            case FNEG:
            case INEG:
            case L2D:
            case D2L:
```

```
case F2I:  
case I2B:  
case I2C:  
case I2S:  
case I2F:  
case Opcodes.ARRAYLENGTH:  
    break;  
  
case ACONST_NULL:  
case ICONST_M1:  
case ICONST_0:  
case ICONST_1:  
case ICONST_2:  
case ICONST_3:  
case ICONST_4:  
case ICONST_5:  
case FCONST_0:  
case FCONST_1:  
case FCONST_2:  
case F2L: // 1 before 2 after  
case F2D:  
case I2L:  
case I2D:  
    pushValue(OTHER);  
    break;  
  
case LCONST_0:  
case LCONST_1:  
case DCONST_0:  
case DCONST_1:  
    pushValue(OTHER);  
    pushValue(OTHER);  
    break;  
  
case IALOAD: // remove 2 add 1  
case FALOAD: // remove 2 add 1  
case AALOAD: // remove 2 add 1  
case BALOAD: // remove 2 add 1  
case CALOAD: // remove 2 add 1  
case SALOAD: // remove 2 add 1  
case POP:  
case IADD:  
case FADD:  
case ISUB:  
case LSHL: // 3 before 2 after  
case LSHR: // 3 before 2 after  
case LUSHR: // 3 before 2 after  
case L2I: // 2 before 1 after  
case L2F: // 2 before 1 after  
case D2I: // 2 before 1 after
```

```
case D2F: // 2 before 1 after
case FSUB:
case FMUL:
case FDIV:
case FREM:
case FCMPL: // 2 before 1 after
case FCMPG: // 2 before 1 after
case IMUL:
case IDIV:
case IREM:
case ISHL:
case ISHR:
case IUSHR:
case IAND:
case IOR:
case IXOR:
case MONITORENTER:
case MONITOREXIT:
    popValue();
    break;

case POP2:
case LSUB:
case LMUL:
case LDIV:
case LREM:
case LADD:
case LAND:
case LOR:
case LXOR:
case DADD:
case DMUL:
case DSUB:
case DDIV:
case DREM:
    popValue();
    popValue();
    break;

case IASTORE:
case FASTORE:
case AASTORE:
case BASTORE:
case CASTORE:
case SASTORE:
case LCMP: // 4 before 1 after
case DCMPL:
case DCMPG:
    popValue();
    popValue();
```

```
popValue();
break;

case LASTORE:
case DASTORE:
    popValue();
    popValue();
    popValue();
    popValue();
    break;

case DUP:
    pushValue(peekValue());
    break;

case DUP_X1:
    // TODO optimize this
{
Object o1 = popValue();
Object o2 = popValue();
pushValue(o1);
pushValue(o2);
pushValue(o1);
}
break;

case DUP_X2:
    // TODO optimize this
{
Object o1 = popValue();
Object o2 = popValue();
Object o3 = popValue();
pushValue(o1);
pushValue(o3);
pushValue(o2);
pushValue(o1);
}
break;

case DUP2:
    // TODO optimize this
{
Object o1 = popValue();
Object o2 = popValue();
pushValue(o2);
pushValue(o1);
pushValue(o2);
pushValue(o1);
}
break;
```

```
case DUP2_X1:
    // TODO optimize this
{
Object o1 = popValue();
Object o2 = popValue();
Object o3 = popValue();
pushValue(o2);
pushValue(o1);
pushValue(o3);
pushValue(o2);
pushValue(o1);
}
break;

case DUP2_X2:
    // TODO optimize this
{
Object o1 = popValue();
Object o2 = popValue();
Object o3 = popValue();
Object o4 = popValue();
pushValue(o2);
pushValue(o1);
pushValue(o4);
pushValue(o3);
pushValue(o2);
pushValue(o1);
}
break;

case SWAP:
{
Object o1 = popValue();
Object o2 = popValue();
pushValue(o1);
pushValue(o2);
}
break;
}

else
{
switch(opcode)
{
case RETURN:
case IRETURN:
case FRETURN:
case ARETURN:
case LRETURN:
```

```
        case DRETURN:
        case ATHROW:
            onMethodExit(opcode);
            break;
        }
    }
    mv.visitInsn(opcode);
}

public void visitVarInsn(final int opcode, final int var){
    super.visitVarInsn(opcode, var);

    if(constructor)
    {
        switch(opcode)
        {
        case ILOAD:
        case FLOAD:
            pushValue(OTHER);
            break;
        case LLOAD:
        case DLOAD:
            pushValue(OTHER);
            pushValue(OTHER);
            break;
        case ALOAD:
            pushValue(var == 0 ? THIS : OTHER);
            break;
        case ASTORE:
        case ISTORE:
        case FSTORE:
            popValue();
            break;
        case LSTORE:
        case DSTORE:
            popValue();
            popValue();
            break;
        }
    }
}

public void visitFieldInsn(
    final int opcode,
    final String owner,
    final String name,
    final String desc){
    mv.visitFieldInsn(opcode, owner, name, desc);

    if(constructor)
```

```

{
    char c = desc.charAt(0);
    boolean longOrDouble = c == 'J' || c == 'D';
    switch(opcode)
    {
        case GETSTATIC:
            pushValue(OTHER);
            if(longOrDouble)
            {
                pushValue(OTHER);
            }
            break;
        case PUTSTATIC:
            popValue();
            if(longOrDouble)
            {
                popValue();
            }
            break;
        case PUTFIELD:
            popValue();
            if(longOrDouble)
            {
                popValue();
                popValue();
            }
            break;
        // case GETFIELD:
        default:
            if(longOrDouble)
            {
                pushValue(OTHER);
            }
    }
}

public void visitIntInsn(final int opcode, final int operand){
    mv.visitIntInsn(opcode, operand);

    if(constructor && opcode != NEWARRAY)
    {
        pushValue(OTHER);
    }
}

public void visitLdcInsn(final Object cst){
    mv.visitLdcInsn(cst);

    if(constructor)
}

```

```

    {
    pushValue(OTHER);
    if(cst instanceof Double || cst instanceof Long)
    {
        pushValue(OTHER);
    }
}
}

public void visitMultiANewArrayInsn(final String desc, final int dims){
    mv.visitMultiANewArrayInsn(desc, dims);

    if(constructor)
    {
        for(int i = 0; i < dims; i++)
        {
            popValue();
        }
        pushValue(OTHER);
    }
}

public void visitTypeInsn(final int opcode, final String name){
    mv.visitTypeInsn(opcode, name);

    // ANEWARRAY, CHECKCAST or INSTANCEOF don't change stack
    if(constructor && opcode == NEW)
    {
        pushValue(OTHER);
    }
}

public void visitMethodInsn(
    final int opcode,
    final String owner,
    final String name,
    final String desc){
    mv.visitMethodInsn(opcode, owner, name, desc);

    if(constructor)
    {
        Type[] types = Type.getArgumentTypes(desc);
        for(int i = 0; i < types.length; i++)
        {
            popValue();
            if(types[i].getSize() == 2)
            {
                popValue();
            }
        }
    }
}

```

```

switch(opcode)
{
// case INVOKESTATIC:
// break;

case INVOKEINTERFACE:
case INVOKEVIRTUAL:
    popValue(); // objectref
    break;

case INVOKESPECIAL:
    Object type = popValue(); // objectref
    if(type == THIS && !superInitialized)
    {
        onMethodEnter();
        superInitialized = true;
        // once super has been initialized it is no longer
        // necessary to keep track of stack state
        constructor = false;
    }
    break;
}

Type returnType = Type.getType(desc);
if(returnType != Type.VOID_TYPE)
{
    pushValue(OTHER);
    if(returnType.getSize() == 2)
    {
        pushValue(OTHER);
    }
}
}

public void visitJumpInsn(final int opcode, final Label label){
    mv.visitJumpInsn(opcode, label);

    if(constructor)
    {
        switch(opcode)
        {
        case IFEQ:
        case IFNE:
        case IFLT:
        case IFGE:
        case IFGT:
        case IFLE:
        case IFNULL:
        case IFNONNULL:
        }
    }
}

```

```

        popValue();
        break;

    case IF_ICMPEQ:
    case IF_ICMPNE:
    case IF_ICMPLT:
    case IF_ICMPGE:
    case IF_ICMPGT:
    case IF_ICMPLE:
    case IF_ACMPEQ:
    case IF_ACMPNE:
        popValue();
        popValue();
        break;

    case JSR:
        pushValue(OTHER);
        break;
    }
    addBranch(label);
}
}

public void visitLookupSwitchInsn(
    final Label dflt,
    final int[] keys,
    final Label[] labels){
    mv.visitLookupSwitchInsn(dflt, keys, labels);

    if(constructor)
    {
        popValue();
        addBranches(dflt, labels);
    }
}

public void visitTableSwitchInsn(
    final int min,
    final int max,
    final Label dflt,
    final Label[] labels){
    mv.visitTableSwitchInsn(min, max, dflt, labels);

    if(constructor)
    {
        popValue();
        addBranches(dflt, labels);
    }
}
}

```

```

private void addBranches(final Label dflt, final Label[] labels){
    addBranch(dflt);
    for(int i = 0; i < labels.length; i++)
    {
        addBranch(labels[i]);
    }
}

private void addBranch(final Label label){
    if(branches.containsKey(label))
    {
        return;
    }
    ArrayList frame = new ArrayList();
    frame.addAll(stackFrame);
    branches.put(label, frame);
}

private Object popValue(){
    return stackFrame.remove(stackFrame.size() - 1);
}

private Object peekValue(){
    return stackFrame.get(stackFrame.size() - 1);
}

private void pushValue(final Object o){
    stackFrame.add(o);
}

/**
 * Called at the beginning of the method or after super class class
 * call in the constructor. <br><br>
 * <p>
 * <i>Custom code can use or change all the local variables, but
 * should not change state of the stack.</i>
 */
protected abstract void onMethodEnter();

/**
 * Called before explicit exit from the method using either return
 * or throw. Top element on the stack contains the return value or
 * exception instance.
 * For example:
 * <p/>
 * <pre>
 *   public void onMethodExit(int opcode) {
 *       if(opcode==RETURN) {
 *           visitInsn(ACONST_NULL);
 *       } else if(opcode==ARETURN || opcode==ATHROW) {
 *

```

```

*           dup();
*       } else {
*           if(opcode==LRETURN || opcode==DRETURN) {
*               dup2();
*           } else {
*               dup();
*           }
*           box(Type.getType(this.methodDesc));
*       }
*       visitIntInsn(SIPUSH, opcode);
*       visitMethodInsn(INVOKESTATIC, owner, "onExit",
*                       "(Ljava/lang/Object;I)V");
*   }
* <p/>
* // an actual call back method
* public static void onExit(int opcode, Object param) {
*     ...
* </pre>
* <p/>
* <br><br>
* <p/>
* <i>Custom code can use or change all the local variables, but should
* not change state of the stack.</i>
*
* @param opcode one of the RETURN, IRETURN, FRETURN, ARETURN, LRETURN,
*               DRETURN or ATHROW
*/
protected abstract void onMethodExit(int opcode);

// TODO onException, onMethodCall
}

```

8.2 AnalyzerAdapter.java

(MethodAdapter [314])
— AnalyzerAdapter.java —

```

\getchunk{France Telecom Copyright}
package clojure.asm.commons;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

```

```

import clojure.asm.Label;
import clojure.asm.MethodAdapter;
import clojure.asm.MethodVisitor;
import clojure.asm.Opcodes;
import clojure.asm.Type;

/**
 * A {@link MethodAdapter} that keeps track of stack map frame changes
 * between
 * {@link #visitFrame(int,int,Object[],int,Object[])} visitFrame} calls.
 * This adapter must be used with the
 * {@link clojure.asm.ClassReader#EXPAND_FRAMES} option. Each
 * visit<i>XXX</i> instruction delegates to the next visitor in the
 * chain, if any, and then simulates the effect of this instruction
 * on the stack map frame, represented by {@link #locals} and
 * {@link #stack}. The next visitor in the chain can get the state
 * of the stack map frame <i>before</i> each instruction by reading
 * the value of these fields in its visit<i>XXX</i> methods (this
 * requires a reference to the AnalyzerAdapter that is before it
 * in the chain).
 *
 * @author Eric Bruneton
 */
public class AnalyzerAdapter extends MethodAdapter{

    /**
     * <code>List</code> of the local variable slots for current execution
     * frame. Primitive types are represented by {@link Opcodes#TOP},
     * {@link Opcodes#INTEGER}, {@link Opcodes#FLOAT}, {@link Opcodes#LONG},
     * {@link Opcodes#DOUBLE},{@link Opcodes#NULL} or
     * {@link Opcodes#UNINITIALIZED_THIS} (long and double are represented
     * by a two elements, the second one being TOP). Reference types are
     * represented by String objects (representing internal names, or type
     * descriptors for array types), and uninitialized types by Label
     * objects (this label designates the NEW instruction that created
     * this uninitialized value). This field is <tt>null</tt> for
     * unreachable instructions.
     */
    public List locals;

    /**
     * <code>List</code> of the operand stack slots for current execution
     * frame. Primitive types are represented by {@link Opcodes#TOP},
     * {@link Opcodes#INTEGER}, {@link Opcodes#FLOAT}, {@link Opcodes#LONG},
     * {@link Opcodes#DOUBLE},{@link Opcodes#NULL} or
     * {@link Opcodes#UNINITIALIZED_THIS} (long and double are represented
     * by a two elements, the second one being TOP). Reference types are
     * represented by String objects (representing internal names, or type
     * descriptors for array types), and uninitialized types by Label
     * objects (this label designates the NEW instruction that created this

```

```
* uninitialized value). This field is <tt>null</tt> for unreachable
* instructions.
*/
public List stack;

/**
 * The labels that designate the next instruction to be visited. May be
 * <tt>null</tt>.
 */
private List labels;

/**
 * Information about uninitialized types in the current execution frame.
 * This map associates internal names to Label objects. Each label
 * designates a NEW instruction that created the currently uninitialized
 * types, and the associated internal name represents the NEW operand,
 * i.e. the final, initialized type value.
 */
private Map uninitializedTypes;

/**
 * The maximum stack size of this method.
 */
private int maxStack;

/**
 * The maximum number of local variables of this method.
 */
private int maxLocals;

/**
 * Creates a new {@link AnalyzerAdapter}.
 *
 * @param owner the owner's class name.
 * @param access the method's access flags (see {@link Opcodes}).
 * @param name the method's name.
 * @param desc the method's descriptor (see {@link Type Type}).
 * @param mv the method visitor to which this adapter delegates
 *           calls. May be <tt>null</tt>.
 */
public AnalyzerAdapter(
    final String owner,
    final int access,
    final String name,
    final String desc,
    final MethodVisitor mv){
    super(mv);
    locals = new ArrayList();
    stack = new ArrayList();
    uninitializedTypes = new HashMap();
```

```

if((access & Opcodes.ACC_STATIC) == 0)
{
    if(name.equals("<init>"))
    {
        locals.add(Opcodes.UNINITIALIZED_THIS);
    }
    else
    {
        locals.add(owner);
    }
}
Type[] types = Type.getArgumentTypes(desc);
for(int i = 0; i < types.length; ++i)
{
    Type type = types[i];
    switch(type.getSort())
    {
        case Type.BOOLEAN:
        case Type.CHAR:
        case Type.BYTE:
        case Type.SHORT:
        case Type.INT:
            locals.add(Opcodes.INTEGER);
            break;
        case Type.FLOAT:
            locals.add(Opcodes.FLOAT);
            break;
        case Type.LONG:
            locals.add(Opcodes.LONG);
            locals.add(Opcodes.TOP);
            break;
        case Type.DOUBLE:
            locals.add(Opcodes.DOUBLE);
            locals.add(Opcodes.TOP);
            break;
        case Type.ARRAY:
            locals.add(types[i].getDescriptor());
            break;
        // case Type.OBJECT:
        default:
            locals.add(types[i].getInternalName());
    }
}
}

public void visitFrame(
    final int type,
    final int nLocal,
    final Object[] local,

```

```

        final int nStack,
        final Object[] stack){
    if(type != Opcodes.F_NEW)
        { // uncompressed frame
        throw new IllegalStateException(
        "ClassReader.accept() should be called with EXPAND_FRAMES flag");
        }

    if(mv != null)
    {
        mv.visitFrame(type, nLocal, local, nStack, stack);
    }

    if(this.locals != null)
    {
        this.locals.clear();
        this.stack.clear();
    }
    else
    {
        this.locals = new ArrayList();
        this.stack = new ArrayList();
    }
    visitFrameTypes(nLocal, local, this.locals);
    visitFrameTypes(nStack, stack, this.stack);
    maxStack = Math.max(maxStack, this.stack.size());
}

private void visitFrameTypes(
    final int n,
    final Object[] types,
    final List result){
    for(int i = 0; i < n; ++i)
    {
        Object type = types[i];
        result.add(type);
        if(type == Opcodes.LONG || type == Opcodes.DOUBLE)
        {
            result.add(Opcodes.TOP);
        }
    }
}

public void visitInsn(final int opcode){
    if(mv != null)
    {
        mv.visitInsn(opcode);
    }
    execute(opcode, 0, null);
    if((opcode >= Opcodes.IRETURN && opcode <= Opcodes.RETURN)

```

```

        || opcode == Opcodes.ATHROW)
    {
        this.locals = null;
        this.stack = null;
    }
}

public void visitIntInsn(final int opcode, final int operand){
    if(mv != null)
    {
        mv.visitIntInsn(opcode, operand);
    }
    execute(opcode, operand, null);
}

public void visitVarInsn(final int opcode, final int var){
    if(mv != null)
    {
        mv.visitVarInsn(opcode, var);
    }
    execute(opcode, var, null);
}

public void visitTypeInsn(final int opcode, final String desc){
    if(opcode == Opcodes.NEW)
    {
        if(labels == null)
        {
            Label l = new Label();
            labels = new ArrayList(3);
            labels.add(l);
            if(mv != null)
            {
                mv.visitLabel(l);
            }
        }
        for(int i = 0; i < labels.size(); ++i)
        {
            uninitializedTypes.put(labels.get(i), desc);
        }
    }
    if(mv != null)
    {
        mv.visitTypeInsn(opcode, desc);
    }
    execute(opcode, 0, desc);
}

public void visitFieldInsn(
    final int opcode,

```

```

        final String owner,
        final String name,
        final String desc){
    if(mv != null)
    {
        mv.visitFieldInsn(opcode, owner, name, desc);
    }
    execute(opcode, 0, desc);
}

public void visitMethodInsn(
    final int opcode,
    final String owner,
    final String name,
    final String desc){
    if(mv != null)
    {
        mv.visitMethodInsn(opcode, owner, name, desc);
    }
    pop(desc);
    if(opcode != Opcodes.INVOKESTATIC)
    {
        Object t = pop();
        if(opcode == Opcodes.INVOKESPECIAL && name.charAt(0) == '<')
        {
            Object u;
            if(t == Opcodes.UNINITIALIZED_THIS)
            {
                u = owner;
            }
            else
            {
                u = uninitializedTypes.get(t);
            }
            for(int i = 0; i < locals.size(); ++i)
            {
                if(locals.get(i) == t)
                {
                    locals.set(i, u);
                }
            }
            for(int i = 0; i < stack.size(); ++i)
            {
                if(stack.get(i) == t)
                {
                    stack.set(i, u);
                }
            }
        }
    }
}

```

```

        pushDesc(desc);
        labels = null;
    }

    public void visitJumpInsn(final int opcode, final Label label){
        if(mv != null)
        {
            mv.visitJumpInsn(opcode, label);
        }
        execute(opcode, 0, null);
        if(opcode == Opcodes.GOTO)
        {
            this.locals = null;
            this.stack = null;
        }
    }

    public void visitLabel(final Label label){
        if(mv != null)
        {
            mv.visitLabel(label);
        }
        if(labels == null)
        {
            labels = new ArrayList(3);
        }
        labels.add(label);
    }

    public void visitLdcInsn(final Object cst){
        if(mv != null)
        {
            mv.visitLdcInsn(cst);
        }
        if(cst instanceof Integer)
        {
            push(Opcodes.INTEGER);
        }
        else if(cst instanceof Long)
        {
            push(Opcodes.LONG);
            push(Opcodes.TOP);
        }
        else if(cst instanceof Float)
        {
            push(Opcodes.FLOAT);
        }
        else if(cst instanceof Double)
        {
            push(Opcodes.DOUBLE);
        }
    }
}

```

```

        push(OpCodes.TOP);
    }
    else if(cst instanceof String)
    {
        push("java/lang/String");
    }
    else if(cst instanceof Type)
    {
        push("java/lang/Class");
    }
    else
    {
        throw new IllegalArgumentException();
    }
    labels = null;
}

public void visitIincInsn(final int var, final int increment){
    if(mv != null)
    {
        mv.visitIincInsn(var, increment);
    }
    execute(OpCodes.IINC, var, null);
}

public void visitTableSwitchInsn(
    final int min,
    final int max,
    final Label dflt,
    final Label labels[]){
if(mv != null)
{
    mv.visitTableSwitchInsn(min, max, dflt, labels);
}
execute(OpCodes.TABLESWITCH, 0, null);
this.locals = null;
this.stack = null;
}

public void visitLookupSwitchInsn(
    final Label dflt,
    final int keys[],
    final Label labels[]){
if(mv != null)
{
    mv.visitLookupSwitchInsn(dflt, keys, labels);
}
execute(OpCodes.LOOKUPSWITCH, 0, null);
this.locals = null;
this.stack = null;
}

```

```

    }

    public void visitMultiANewArrayInsn(final String desc, final int dims){
        if(mv != null)
        {
            mv.visitMultiANewArrayInsn(desc, dims);
        }
        execute(Opcodes.MULTIANEWARRAY, dims, desc);
    }

    public void visitMaxs(final int maxStack, final int maxLocals){
        if(mv != null)
        {
            this.maxStack = Math.max(this.maxStack, maxStack);
            this.maxLocals = Math.max(this.maxLocals, maxLocals);
            mv.visitMaxs(this.maxStack, this.maxLocals);
        }
    }

    // -----
    private Object get(final int local){
        maxLocals = Math.max(maxLocals, local);
        return local < locals.size() ? locals.get(local) : Opcodes.TOP;
    }

    private void set(final int local, final Object type){
        maxLocals = Math.max(maxLocals, local);
        while(local >= locals.size())
        {
            locals.add(Opcodes.TOP);
        }
        locals.set(local, type);
    }

    private void push(final Object type){
        stack.add(type);
        maxStack = Math.max(maxStack, stack.size());
    }

    private void pushDesc(final String desc){
        int index = desc.charAt(0) == '(' ? desc.indexOf(')') + 1 : 0;
        switch(desc.charAt(index))
        {
            case 'V':
                return;
            case 'Z':
            case 'C':
            case 'B':
            case 'S':

```

```

        case'I':
            push(OpCodes.INTEGER);
            return;
        case'F':
            push(OpCodes.FLOAT);
            return;
        case'J':
            push(OpCodes.LONG);
            push(OpCodes.TOP);
            return;
        case'D':
            push(OpCodes.DOUBLE);
            push(OpCodes.TOP);
            return;
        case '[':
            if(index == 0)
            {
                push(desc);
            }
            else
            {
                push(desc.substring(index, desc.length()));
            }
            break;
        // case 'L':
        default:
            if(index == 0)
            {
                push(desc.substring(1, desc.length() - 1));
            }
            else
            {
                push(desc.substring(index + 1, desc.length() - 1));
            }
            return;
        }
    }

private Object pop(){
    return stack.remove(stack.size() - 1);
}

private void pop(final int n){
    int size = stack.size();
    int end = size - n;
    for(int i = size - 1; i >= end; --i)
    {
        stack.remove(i);
    }
}

```

```
private void pop(final String desc){
    char c = desc.charAt(0);
    if(c == '(')
    {
        int n = 0;
        Type[] types = Type.getArgumentTypes(desc);
        for(int i = 0; i < types.length; ++i)
        {
            n += types[i].getSize();
        }
        pop(n);
    }
    else if(c == 'J' || c == 'D')
    {
        pop(2);
    }
    else
    {
        pop(1);
    }
}

private void execute(final int opcode,
                     final int iarg,
                     final String sarg){
    if(this.locals == null)
    {
        return;
    }
    Object t1, t2, t3, t4;
    switch(opcode)
    {
        case Opcodes.NOP:
        case Opcodes.INEG:
        case Opcodes.LNEG:
        case Opcodes.FNEG:
        case Opcodes.DNEG:
        case Opcodes.I2B:
        case Opcodes.I2C:
        case Opcodes.I2S:
        case Opcodes.GOTO:
        case Opcodes.RETURN:
            break;
        case Opcodes.ACONST_NULL:
            push(Opcodes.NULL);
            break;
        case Opcodes.ICONST_M1:
        case Opcodes.ICONST_0:
        case Opcodes.ICONST_1:
```

```
case Opcodes.ICONST_2:  
case Opcodes.ICONST_3:  
case Opcodes.ICONST_4:  
case Opcodes.ICONST_5:  
case Opcodes.BIPUSH:  
case Opcodes.SIPUSH:  
    push(Opcodes.INTEGER);  
    break;  
case Opcodes.LCONST_0:  
case Opcodes.LCONST_1:  
    push(Opcodes.LONG);  
    push(Opcodes.TOP);  
    break;  
case Opcodes.FCONST_0:  
case Opcodes.FCONST_1:  
case Opcodes.FCONST_2:  
    push(Opcodes.FLOAT);  
    break;  
case Opcodes.DCONST_0:  
case Opcodes.DCONST_1:  
    push(Opcodes.DOUBLE);  
    push(Opcodes.TOP);  
    break;  
case Opcodes.ILOAD:  
case Opcodes.FLOAD:  
case Opcodes.ALOAD:  
    push(get(iarg));  
    break;  
case Opcodes.LLOAD:  
case Opcodes.DLOAD:  
    push(get(iarg));  
    push(Opcodes.TOP);  
    break;  
case Opcodes.IALOAD:  
case Opcodes.BALOAD:  
case Opcodes.CALOAD:  
case Opcodes.SALOAD:  
    pop(2);  
    push(Opcodes.INTEGER);  
    break;  
case Opcodes.LALOAD:  
case Opcodes.D2L:  
    pop(2);  
    push(Opcodes.LONG);  
    push(Opcodes.TOP);  
    break;  
case Opcodes.FALOAD:  
    pop(2);  
    push(Opcodes.FLOAT);  
    break;
```

```
case Opcodes.DALOAD:
case Opcodes.L2D:
    pop(2);
    push(Opcodes.DOUBLE);
    push(Opcodes.TOP);
    break;
case Opcodes.AALOAD:
    pop(1);
    t1 = pop();
    pushDesc(((String) t1).substring(1));
    break;
case Opcodes.ISTORE:
case Opcodes.FSTORE:
case Opcodes.ASTORE:
    t1 = pop();
    set(iarg, t1);
    if(iarg > 0)
    {
        t2 = get(iarg - 1);
        if(t2 == Opcodes.LONG || t2 == Opcodes.DOUBLE)
        {
            set(iarg - 1, Opcodes.TOP);
        }
    }
    break;
case Opcodes.LSTORE:
case Opcodes.DSTORE:
    pop(1);
    t1 = pop();
    set(iarg, t1);
    set(iarg + 1, Opcodes.TOP);
    if(iarg > 0)
    {
        t2 = get(iarg - 1);
        if(t2 == Opcodes.LONG || t2 == Opcodes.DOUBLE)
        {
            set(iarg - 1, Opcodes.TOP);
        }
    }
    break;
case Opcodes.IASTORE:
case Opcodes.BASTORE:
case Opcodes.CASTORE:
case Opcodes.SASTORE:
case Opcodes.FASTORE:
case Opcodes.AASTORE:
    pop(3);
    break;
case Opcodes.LASTORE:
case Opcodes.DASTORE:
```

```
    pop(4);
    break;
case Opcodes.POP:
case Opcodes.IFEQ:
case Opcodes.IFNE:
case Opcodes.IFLT:
case Opcodes.IFGE:
case Opcodes.IFGT:
case Opcodes.IFILE:
case Opcodes.IRETURN:
case Opcodes.FRETURN:
case Opcodes.ARETURN:
case Opcodes.TABLESWITCH:
case Opcodes.LOOKUPSWITCH:
case Opcodes.ATHROW:
case Opcodes.MONITORENTER:
case Opcodes.MONITOREXIT:
case Opcodes.IFNULL:
case Opcodes.IFNONNULL:
    pop(1);
    break;
case Opcodes.POP2:
case Opcodes.IF_ICMPEQ:
case Opcodes.IF_ICMPNE:
case Opcodes.IF_ICMPLT:
case Opcodes.IF_ICMPGE:
case Opcodes.IF_ICMPGT:
case Opcodes.IF_ICMPLE:
case Opcodes.IF_ACMPEQ:
case Opcodes.IF_ACMPNE:
case Opcodes.LRETURN:
case Opcodes.DRETURN:
    pop(2);
    break;
case Opcodes.DUP:
    t1 = pop();
    push(t1);
    push(t1);
    break;
case Opcodes.DUP_X1:
    t1 = pop();
    t2 = pop();
    push(t1);
    push(t2);
    push(t1);
    break;
case Opcodes.DUP_X2:
    t1 = pop();
    t2 = pop();
    t3 = pop();
```

```
push(t1);
push(t3);
push(t2);
push(t1);
break;
case Opcodes.DUP2:
    t1 = pop();
    t2 = pop();
    push(t2);
    push(t1);
    push(t2);
    push(t1);
    break;
case Opcodes.DUP2_X1:
    t1 = pop();
    t2 = pop();
    t3 = pop();
    push(t2);
    push(t1);
    push(t3);
    push(t2);
    push(t1);
    break;
case Opcodes.DUP2_X2:
    t1 = pop();
    t2 = pop();
    t3 = pop();
    t4 = pop();
    push(t2);
    push(t1);
    push(t4);
    push(t3);
    push(t2);
    push(t1);
    break;
case Opcodes.SWAP:
    t1 = pop();
    t2 = pop();
    push(t1);
    push(t2);
    break;
case Opcodes.IADD:
case Opcodes.ISUB:
case Opcodes.IMUL:
case Opcodes.IDIV:
case Opcodes.IREM:
case Opcodes.IAND:
case Opcodes.IOR:
case Opcodes.IXOR:
case Opcodes.ISHL:
```

```
case Opcodes.ISHR:
case Opcodes.IU SHR:
case Opcodes.L2I:
case Opcodes.D2I:
case Opcodes.FCMPL:
case Opcodes.FCMPG:
    pop(2);
    push(Opcodes.INTEGER);
    break;
case Opcodes.LADD:
case Opcodes.LSUB:
case Opcodes.LMUL:
case Opcodes.LDIV:
case Opcodes.LREM:
case Opcodes.LAND:
case Opcodes.LOR:
case Opcodes.LXOR:
    pop(4);
    push(Opcodes.LONG);
    push(Opcodes.TOP);
    break;
case Opcodes.FADD:
case Opcodes.FSUB:
case Opcodes.FMUL:
case Opcodes.FDIV:
case Opcodes.FREM:
case Opcodes.L2F:
case Opcodes.D2F:
    pop(2);
    push(Opcodes.FLOAT);
    break;
case Opcodes.DADD:
case Opcodes.DSUB:
case Opcodes.DMUL:
case Opcodes.DDIV:
case Opcodes.DREM:
    pop(4);
    push(Opcodes.DOUBLE);
    push(Opcodes.TOP);
    break;
case Opcodes.LSHL:
case Opcodes.LSHR:
case Opcodes.LUSHR:
    pop(3);
    push(Opcodes.LONG);
    push(Opcodes.TOP);
    break;
case Opcodes.IINC:
    set(iarg, Opcodes.INTEGER);
    break;
```

```
case Opcodes.I2L:
case Opcodes.F2L:
    pop(1);
    push(Opcodes.LONG);
    push(Opcodes.TOP);
    break;
case Opcodes.I2F:
    pop(1);
    push(Opcodes.FLOAT);
    break;
case Opcodes.I2D:
case Opcodes.F2D:
    pop(1);
    push(Opcodes.DOUBLE);
    push(Opcodes.TOP);
    break;
case Opcodes.F2I:
case Opcodes.ARRAYLENGTH:
case Opcodes.INSTANCEOF:
    pop(1);
    push(Opcodes.INTEGER);
    break;
case Opcodes.LCMP:
case Opcodes.DCMPL:
case Opcodes.DCMPG:
    pop(4);
    push(Opcodes.INTEGER);
    break;
case Opcodes.JSR:
case Opcodes.RET:
    throw new RuntimeException("JSR/RET are not supported");
case Opcodes.GETSTATIC:
    pushDesc(sarg);
    break;
case Opcodes.PUTSTATIC:
    pop(sarg);
    break;
case Opcodes.GETFIELD:
    pop(1);
    pushDesc(sarg);
    break;
case Opcodes.PUTFIELD:
    pop(sarg);
    pop();
    break;
case Opcodes.NEW:
    push(labels.get(0));
    break;
case Opcodes.NEWARRAY:
    pop();
```

```
switch(iarg)
{
    case Opcodes.T_BOOLEAN:
        pushDesc("[Z");
        break;
    case Opcodes.T_CHAR:
        pushDesc("[C");
        break;
    case Opcodes.T_BYTE:
        pushDesc("[B");
        break;
    case Opcodes.T_SHORT:
        pushDesc("[S");
        break;
    case Opcodes.T_INT:
        pushDesc("[I");
        break;
    case Opcodes.T_FLOAT:
        pushDesc("[F");
        break;
    case Opcodes.T_DOUBLE:
        pushDesc("[D");
        break;
    // case Opcodes.T_LONG:
    default:
        pushDesc("[J");
        break;
}
break;
case Opcodes.ANEWARRAY:
    pop();
    if(sarg.charAt(0) == '[')
    {
        pushDesc("[" + sarg);
    }
    else
    {
        pushDesc("[L" + sarg + ";");
    }
    break;
case Opcodes.CHECKCAST:
    pop();
    if(sarg.charAt(0) == '[')
    {
        pushDesc(sarg);
    }
    else
    {
        push(sarg);
    }
}
```

```
        break;
        // case Opcodes.MULTIANEWARRAY:
    default:
        pop(iarg);
        pushDesc(sarg);
        break;
    }
    labels = null;
}
}
```

8.3 CodeSizeEvaluator.java

(MethodAdapter [314]) (Opcodes [387])
— CodeSizeEvaluator.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm.commons;

import clojure.asm.Label;
import clojure.asm.MethodAdapter;
import clojure.asm.MethodVisitor;
import clojure.asm.Opcodes;

/**
 * A {@link MethodAdapter} that can be used to approximate method size.
 *
 * @author Eugene Kuleshov
 */
public class CodeSizeEvaluator extends MethodAdapter implements Opcodes{

    private int minSize;

    private int maxSize;

    public CodeSizeEvaluator(final MethodVisitor mv){
        super(mv);
    }

    public int getMinSize(){
        return this.minSize;
    }

    public int getMaxSize(){
        return this.maxSize;
    }
}
```

```
public void visitInsn(final int opcode){
    minSize += 1;
    maxSize += 1;
    if(mv != null)
    {
        mv.visitInsn(opcode);
    }
}

public void visitIntInsn(final int opcode, final int operand){
    if(opcode == SIPUSH)
    {
        minSize += 3;
        maxSize += 3;
    }
    else
    {
        minSize += 2;
        maxSize += 2;
    }
    if(mv != null)
    {
        mv.visitIntInsn(opcode, operand);
    }
}

public void visitVarInsn(final int opcode, final int var){
    if(var < 4 && opcode != Opcodes.RET)
    {
        minSize += 1;
        maxSize += 1;
    }
    else if(var >= 256)
    {
        minSize += 4;
        maxSize += 4;
    }
    else
    {
        minSize += 2;
        maxSize += 2;
    }
    if(mv != null)
    {
        mv.visitVarInsn(opcode, var);
    }
}

public void visitTypeInsn(final int opcode, final String desc){
```

```
minSize += 3;
maxSize += 3;
if(mv != null)
{
    mv.visitTypeInsn(opcode, desc);
}
}

public void visitFieldInsn(
    final int opcode,
    final String owner,
    final String name,
    final String desc){
minSize += 3;
maxSize += 3;
if(mv != null)
{
    mv.visitFieldInsn(opcode, owner, name, desc);
}
}

public void visitMethodInsn(
    final int opcode,
    final String owner,
    final String name,
    final String desc){
if(opcode == INVOKEINTERFACE)
{
    minSize += 5;
    maxSize += 5;
}
else
{
    minSize += 3;
    maxSize += 3;
}
if(mv != null)
{
    mv.visitMethodInsn(opcode, owner, name, desc);
}
}

public void visitJumpInsn(final int opcode, final Label label){
minSize += 3;
if(opcode == GOTO || opcode == JSR)
{
    maxSize += 5;
}
else
{
```

```
        maxSize += 8;
    }
    if(mv != null)
    {
        mv.visitJumpInsn(opcode, label);
    }
}

public void visitLdcInsn(final Object cst){
    if(cst instanceof Long || cst instanceof Double)
    {
        minSize += 3;
        maxSize += 3;
    }
    else
    {
        minSize += 2;
        maxSize += 3;
    }
    if(mv != null)
    {
        mv.visitLdcInsn(cst);
    }
}

public void visitIincInsn(final int var, final int increment){
    if(var > 255 || increment > 127 || increment < -128)
    {
        minSize += 6;
        maxSize += 6;
    }
    else
    {
        minSize += 3;
        maxSize += 3;
    }
    if(mv != null)
    {
        mv.visitIincInsn(var, increment);
    }
}

public void visitTableSwitchInsn(
    final int min,
    final int max,
    final Label dflt,
    final Label[] labels){
    minSize += 13 + labels.length * 4;
    maxSize += 16 + labels.length * 4;
    if(mv != null)
```

```

        {
            mv.visitTableSwitchInsn(min, max, dflt, labels);
        }
    }

    public void visitLookupSwitchInsn(
        final Label dflt,
        final int[] keys,
        final Label[] labels){
        minSize += 9 + keys.length * 8;
        maxSize += 12 + keys.length * 8;
        if(mv != null)
        {
            mv.visitLookupSwitchInsn(dflt, keys, labels);
        }
    }

    public void visitMultiANewArrayInsn(final String desc, final int dims){
        minSize += 4;
        maxSize += 4;
        if(mv != null)
        {
            mv.visitMultiANewArrayInsn(desc, dims);
        }
    }
}

```

8.4 EmptyVisitor.java

(ClassVisitor [229]) (FieldVisitor [263]) (MethodVisitor [317]) (AnnotationVisitor [163])

— EmptyVisitor.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm.commons;

import clojure.asm.AnnotationVisitor;
import clojure.asm.Attribute;
import clojure.asm.ClassVisitor;
import clojure.asm.FieldVisitor;
import clojure.asm.Label;
import clojure.asm.MethodVisitor;

/**
 * An empty implementation of the ASM visitor interfaces.
 *
```

```
* @author Eric Bruneton
*/
public class EmptyVisitor implements
    ClassVisitor,
    FieldVisitor,
    MethodVisitor,
    AnnotationVisitor{

    public void visit(
        final int version,
        final int access,
        final String name,
        final String signature,
        final String superName,
        final String[] interfaces){
    }

    public void visitSource(final String source, final String debug){
    }

    public void visitOuterClass(
        final String owner,
        final String name,
        final String desc){
    }

    public AnnotationVisitor visitAnnotation(
        final String desc,
        final boolean visible){
        return this;
    }

    public void visitAttribute(final Attribute attr){
    }

    public void visitInnerClass(
        final String name,
        final String outerName,
        final String innerName,
        final int access){
    }

    public FieldVisitor visitField(
        final int access,
        final String name,
        final String desc,
        final String signature,
        final Object value){
        return this;
    }
}
```

```
public MethodVisitor visitMethod(
    final int access,
    final String name,
    final String desc,
    final String signature,
    final String[] exceptions){
    return this;
}

public void visitEnd(){}

public AnnotationVisitor visitAnnotationDefault(){
    return this;
}

public AnnotationVisitor visitParameterAnnotation(
    final int parameter,
    final String desc,
    final boolean visible){
    return this;
}

public void visitCode(){}

public void visitFrame(
    final int type,
    final int nLocal,
    final Object[] local,
    final int nStack,
    final Object[] stack){
}

public void visitInsn(final int opcode){}

public void visitIntInsn(final int opcode, final int operand){}

public void visitVarInsn(final int opcode, final int var){}

public void visitTypeInsn(final int opcode, final String desc){}

public void visitFieldInsn(
    final int opcode,
    final String owner,
```

```
        final String name,
        final String desc){
    }

    public void visitMethodInsn(
        final int opcode,
        final String owner,
        final String name,
        final String desc){
    }

    public void visitJumpInsn(final int opcode, final Label label){
    }

    public void visitLabel(final Label label){
    }

    public void visitLdcInsn(final Object cst){
    }

    public void visitIincInsn(final int var, final int increment){
    }

    public void visitTableSwitchInsn(
        final int min,
        final int max,
        final Label dflt,
        final Label labels[]){
    }

    public void visitLookupSwitchInsn(
        final Label dflt,
        final int keys[],
        final Label labels[]){
    }

    public void visitMultiANewArrayInsn(final String desc, final int dims){
    }

    public void visitTryCatchBlock(
        final Label start,
        final Label end,
        final Label handler,
        final String type){
    }

    public void visitLocalVariable(
        final String name,
        final String desc,
        final String signature,
```

```
    final Label start,
    final Label end,
    final int index){
}

public void visitLineNumber(final int line, final Label start){
}

public void visitMaxs(final int maxStack, final int maxLocals){
}

public void visit(final String name, final Object value){
}

public void visitEnum(
    final String name,
    final String desc,
    final String value){
}

public AnnotationVisitor visitAnnotation(
    final String name,
    final String desc){
    return this;
}

public AnnotationVisitor visitArray(final String name){
    return this;
}
}
```



8.5 GeneratorAdapter.java

(LocalVariablesSorter [484])

— GeneratorAdapter.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm.commons;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import clojure.asm.ClassVisitor;
import clojure.asm.Label;
import clojure.asm.MethodVisitor;
```

```

import clojure.asm.Opcodes;
import clojure.asm.Type;

/**
 * A {@link clojure.asm.MethodAdapter} with convenient methods to
 * generate code. For example, using this adapter, the class below
 * <p/>
* <pre>
* public class Example {
*     public static void main(String[] args) {
*         System.out.println("Hello world!");
*     }
* }
* </pre>
* <p/>
* can be generated as follows:
* <p/>
* <pre>
* ClassWriter cw = new ClassWriter(true);
* cw.visit(V1_1, ACC_PUBLIC, "Example", null,
*           "java/lang/Object", null);
* <p/>
* Method m = Method.getMethod("void <init> ()");
* GeneratorAdapter mg =
*     new GeneratorAdapter(ACC_PUBLIC, m, null, null, cw);
* mg.loadThis();
* mg.invokeConstructor(Type.getType(Object.class), m);
* mg.returnValue();
* mg.endMethod();
* <p/>
* m = Method.getMethod("void main (String[])");
* mg = new GeneratorAdapter(ACC_PUBLIC + ACC_STATIC, m, null, null, cw);
* mg.getStatic(Type.getType(System.class), "out",
*              Type.getType(PrintStream.class));
* mg.push("Hello world!");
* mg.invokeVirtual(Type.getType(PrintStream.class),
*                  Method.getMethod("void println (String)"));
* mg.returnValue();
* mg.endMethod();
* <p/>
* cw.visitEnd();
* </pre>
*
* @author Juozas Baliuka
* @author Chris Nokleberg
* @author Eric Bruneton
*/
public class GeneratorAdapter extends LocalVariablesSorter{

    private final static Type BYTE_TYPE =

```

```
Type.getObjectType("java/lang/Byte");

private final static Type BOOLEAN_TYPE =
    Type.getObjectType("java/lang/Boolean");

private final static Type SHORT_TYPE =
    Type.getObjectType("java/lang/Short");

private final static Type CHARACTER_TYPE =
    Type.getObjectType("java/lang/Character");

private final static Type INTEGER_TYPE =
    Type.getObjectType("java/lang/Integer");

private final static Type FLOAT_TYPE =
    Type.getObjectType("java/lang/Float");

private final static Type LONG_TYPE =
    Type.getObjectType("java/lang/Long");

private final static Type DOUBLE_TYPE =
    Type.getObjectType("java/lang/Double");

private final static Type NUMBER_TYPE =
    Type.getObjectType("java/lang/Number");

private final static Type OBJECT_TYPE =
    Type.getObjectType("java/lang/Object");

private final static Method BOOLEAN_VALUE =
    Method.getMethod("boolean booleanValue()");

private final static Method CHAR_VALUE =
    Method.getMethod("char charValue()");

private final static Method INT_VALUE =
    Method.getMethod("int intValue()");

private final static Method FLOAT_VALUE =
    Method.getMethod("float floatValue()");

private final static Method LONG_VALUE =
    Method.getMethod("long longValue()");

private final static Method DOUBLE_VALUE =
    Method.getMethod("double doubleValue()");

/**  
 * Constant for the {@link #math math} method.  
 */
```

```
public final static int ADD = Opcodes.IADD;

/**
 * Constant for the {@link #math math} method.
 */
public final static int SUB = Opcodes.ISUB;

/**
 * Constant for the {@link #math math} method.
 */
public final static int MUL = Opcodes.IMUL;

/**
 * Constant for the {@link #math math} method.
 */
public final static int DIV = Opcodes.IDIV;

/**
 * Constant for the {@link #math math} method.
 */
public final static int REM = Opcodes.IREM;

/**
 * Constant for the {@link #math math} method.
 */
public final static int NEG = Opcodes.INEG;

/**
 * Constant for the {@link #math math} method.
 */
public final static int SHL = Opcodes.ISHL;

/**
 * Constant for the {@link #math math} method.
 */
public final static int SHR = Opcodes.ISHR;

/**
 * Constant for the {@link #math math} method.
 */
public final static int USHR = Opcodes.IUSHR;

/**
 * Constant for the {@link #math math} method.
 */
public final static int AND = Opcodes.IAND;

/**
 * Constant for the {@link #math math} method.
 */

```

```
public final static int OR = Opcodes.IOR;

/**
 * Constant for the {@link #math math} method.
 */
public final static int XOR = Opcodes.IXOR;

/**
 * Constant for the {@link #ifCmp ifCmp} method.
 */
public final static int EQ = Opcodes.IFEQ;

/**
 * Constant for the {@link #ifCmp ifCmp} method.
 */
public final static int NE = Opcodes.IFNE;

/**
 * Constant for the {@link #ifCmp ifCmp} method.
 */
public final static int LT = Opcodes.IFLT;

/**
 * Constant for the {@link #ifCmp ifCmp} method.
 */
public final static int GE = Opcodes.IFGE;

/**
 * Constant for the {@link #ifCmp ifCmp} method.
 */
public final static int GT = Opcodes.IFGT;

/**
 * Constant for the {@link #ifCmp ifCmp} method.
 */
public final static int LE = Opcodes.IFILE;

/**
 * Access flags of the method visited by this adapter.
 */
private final int access;

/**
 * Return type of the method visited by this adapter.
 */
private final Type returnType;

/**
 * Argument types of the method visited by this adapter.
 */
```

```
private final Type[] argumentTypes;

/**
 * Types of the local variables of the method visited by this adapter.
 */
private final List localTypes = new ArrayList();

/**
 * Creates a new {@link GeneratorAdapter}.
 *
 * @param mv      the method visitor to which this adapter delegates
 *                calls.
 * @param access  the method's access flags (see {@link Opcodes}).
 * @param name    the method's name.
 * @param desc    the method's descriptor (see {@link Type Type}).
 */
public GeneratorAdapter(
    final MethodVisitor mv,
    final int access,
    final String name,
    final String desc){
    super(access, desc, mv);
    this.access = access;
    this.returnType = Type.getType(desc);
    this.argumentTypes = Type.getArgumentTypes(desc);
}

/**
 * Creates a new {@link GeneratorAdapter}.
 *
 * @param access access flags of the adapted method.
 * @param method the adapted method.
 * @param mv      the method visitor to which this adapter delegates
 *                calls.
 */
public GeneratorAdapter(
    final int access,
    final Method method,
    final MethodVisitor mv){
    super(access, method.getDescriptor(), mv);
    this.access = access;
    this.returnType = method.getType();
    this.argumentTypes = method.getArgumentTypes();
}

/**
 * Creates a new {@link GeneratorAdapter}.
 *
 * @param access      access flags of the adapted method.
 * @param method      the adapted method.
 */
```

```

* @param signature the signature of the adapted method (may be
*                   <tt>null</tt>).
* @param exceptions the exceptions thrown by the adapted method
*                   (may be <tt>null</tt>).
* @param cv         the class visitor to which this adapter delegates
*                   calls.
*/
public GeneratorAdapter(
    final int access,
    final Method method,
    final String signature,
    final Type[] exceptions,
    final ClassVisitor cv){
    this(access, method, cv.visitMethod(access,
                                         method.getName(),
                                         method.getDescriptor(),
                                         signature,
                                         getInternalNames(exceptions)));
}

/**
 * Returns the internal names of the given types.
 *
 * @param types a set of types.
 * @return the internal names of the given types.
 */
private static String[] getInternalNames(final Type[] types){
    if(types == null)
    {
        return null;
    }
    String[] names = new String[types.length];
    for(int i = 0; i < names.length; ++i)
    {
        names[i] = types[i].getInternalName();
    }
    return names;
}

// -----
// Instructions to push constants on the stack
// -----


/**
 * Generates the instruction to push the given value on the stack.
 *
 * @param value the value to be pushed on the stack.
 */
public void push(final boolean value){
    push(value ? 1 : 0);
}

```

```

}

/***
 * Generates the instruction to push the given value on the stack.
 *
 * @param value the value to be pushed on the stack.
 */
public void push(final int value){
    if(value >= -1 && value <= 5)
    {
        mv.visitInsn(Opcodes.ICONST_0 + value);
    }
    else if(value >= Byte.MIN_VALUE && value <= Byte.MAX_VALUE)
    {
        mv.visitIntInsn(Opcodes.BIPUSH, value);
    }
    else if(value >= Short.MIN_VALUE && value <= Short.MAX_VALUE)
    {
        mv.visitIntInsn(Opcodes.SIPUSH, value);
    }
    else
    {
        mv.visitLdcInsn(new Integer(value));
    }
}

/***
 * Generates the instruction to push the given value on the stack.
 *
 * @param value the value to be pushed on the stack.
 */
public void push(final long value){
    if(value == 0L || value == 1L)
    {
        mv.visitInsn(Opcodes.LCONST_0 + (int) value);
    }
    else
    {
        mv.visitLdcInsn(new Long(value));
    }
}

/***
 * Generates the instruction to push the given value on the stack.
 *
 * @param value the value to be pushed on the stack.
 */
public void push(final float value){
    int bits = Float.floatToIntBits(value);
    if(bits == 0L || bits == 0x3f800000 || bits == 0x40000000)

```

```
    { // 0..2
        mv.visitInsn(Opcodes.FCONST_0 + (int) value);
    }
    else
    {
        mv.visitLdcInsn(new Float(value));
    }
}

/**
 * Generates the instruction to push the given value on the stack.
 *
 * @param value the value to be pushed on the stack.
 */
public void push(final double value){
    long bits = Double.doubleToLongBits(value);
    if(bits == 0L || bits == 0x3ff0000000000000L)
        { // +0.0d and 1.0d
            mv.visitInsn(Opcodes.DCONST_0 + (int) value);
        }
    else
    {
        mv.visitLdcInsn(new Double(value));
    }
}

/**
 * Generates the instruction to push the given value on the stack.
 *
 * @param value the value to be pushed on the stack. May be
 *              <tt>null</tt>.
 */
public void push(final String value){
    if(value == null)
    {
        mv.visitInsn(Opcodes.ACONST_NULL);
    }
    else
    {
        mv.visitLdcInsn(value);
    }
}

/**
 * Generates the instruction to push the given value on the stack.
 *
 * @param value the value to be pushed on the stack.
 */
public void push(final Type value){
    if(value == null)
```

```

    {
        mv.visitInsn(Opcodes.ACONST_NULL);
    }
else
{
    mv.visitLdcInsn(value);
}
}

// -----
// Instructions to load and store method arguments
// -----


/***
 * Returns the index of the given method argument in the frame's local
 * variables array.
 *
 * @param arg the index of a method argument.
 * @return the index of the given method argument in the frame's local
 *         variables array.
 */
private int getArgIndex(final int arg){
    int index = (access & Opcodes.ACC_STATIC) == 0 ? 1 : 0;
    for(int i = 0; i < arg; i++)
    {
        index += argumentTypes[i].getSize();
    }
    return index;
}

/***
 * Generates the instruction to push a local variable on the stack.
 *
 * @param type the type of the local variable to be loaded.
 * @param index an index in the frame's local variables array.
 */
private void loadInsn(final Type type, final int index){
    mv.visitVarInsn(type.getOpcode(Opcodes.ILOAD), index);
}

/***
 * Generates the instruction to store the top stack value in a local
 * variable.
 *
 * @param type the type of the local variable to be stored.
 * @param index an index in the frame's local variables array.
 */
private void storeInsn(final Type type, final int index){
    mv.visitVarInsn(type.getOpcode(Opcodes.ISTORE), index);
}

```

```
/**  
 * Generates the instruction to load 'this' on the stack.  
 */  
public void loadThis(){  
    if((access & Opcodes.ACC_STATIC) != 0)  
    {  
        throw new IllegalStateException(  
            "no 'this' pointer within static method");  
    }  
    mv.visitVarInsn(Opcodes.ALOAD, 0);  
}  
  
/**  
 * Generates the instruction to load the given method argument  
 * on the stack.  
 *  
 * @param arg the index of a method argument.  
 */  
public void loadArg(final int arg){  
    loadInsn(argumentTypes[arg], getArgIndex(arg));  
}  
  
/**  
 * Generates the instructions to load the given method arguments  
 * on the stack.  
 *  
 * @param arg the index of the first method argument to be loaded.  
 * @param count the number of method arguments to be loaded.  
 */  
public void loadArgs(final int arg, final int count){  
    int index = getArgIndex(arg);  
    for(int i = 0; i < count; ++i)  
    {  
        Type t = argumentTypes[arg + i];  
        loadInsn(t, index);  
        index += t.getSize();  
    }  
}  
  
/**  
 * Generates the instructions to load all the method arguments  
 * on the stack.  
 */  
public void loadArgs(){  
    loadArgs(0, argumentTypes.length);  
}  
  
/**  
 * Generates the instructions to load all the method arguments
```

```

        * on the stack, as a single object array.
    */
public void loadArgArray(){
    push(argumentTypes.length);
    newArray(OBJECT_TYPE);
    for(int i = 0; i < argumentTypes.length; i++)
    {
        dup();
        push(i);
        loadArg(i);
        box(argumentTypes[i]);
        arrayStore(OBJECT_TYPE);
    }
}

/**
 * Generates the instruction to store the top stack value in the given
 * method argument.
 *
 * @param arg the index of a method argument.
 */
public void storeArg(final int arg){
    storeInsn(argumentTypes[arg], getArgIndex(arg));
}

// -----
// Instructions to load and store local variables
// -----


/**
 * Returns the type of the given local variable.
 *
 * @param local a local variable identifier, as returned by
 *              {@link LocalVariablesSorter#newLocal(Type) newLocal()}.
 * @return the type of the given local variable.
 */
public Type getLocalType(final int local){
    return (Type) localTypes.get(local - firstLocal);
}

protected void setLocalType(final int local, final Type type){
    int index = local - firstLocal;
    while(localTypes.size() < index + 1)
    {
        localTypes.add(null);
    }
    localTypes.set(index, type);
}

/**

```

```
* Generates the instruction to load the given local variable
* on the stack.
*
* @param local a local variable identifier, as returned by
*              {@link LocalVariablesSorter#newLocal(Type) newLocal()}.
*/
public void loadLocal(final int local){
    loadInsn(getLocalType(local), local);
}

/**
 * Generates the instruction to load the given local variable
 * on the stack.
 *
 * @param local a local variable identifier, as returned by
 *              {@link LocalVariablesSorter#newLocal(Type) newLocal()}.
 * @param type the type of this local variable.
 */
public void loadLocal(final int local, final Type type){
    setLocalType(local, type);
    loadInsn(type, local);
}

/**
 * Generates the instruction to store the top stack value in the given
 * local variable.
 *
 * @param local a local variable identifier, as returned by
 *              {@link LocalVariablesSorter#newLocal(Type) newLocal()}.
 */
public void storeLocal(final int local){
    storeInsn(getLocalType(local), local);
}

/**
 * Generates the instruction to store the top stack value in the given
 * local variable.
 *
 * @param local a local variable identifier, as returned by
 *              {@link LocalVariablesSorter#newLocal(Type) newLocal()}.
 * @param type the type of this local variable.
 */
public void storeLocal(final int local, final Type type){
    setLocalType(local, type);
    storeInsn(type, local);
}

/**
 * Generates the instruction to load an element from an array.
*
```

```
* @param type the type of the array element to be loaded.  
*/  
public void arrayLoad(final Type type){  
    mv.visitInsn(type.getOpcode(0pcodes.IALOAD));  
}  
  
/**  
 * Generates the instruction to store an element in an array.  
 *  
 * @param type the type of the array element to be stored.  
 */  
public void arrayStore(final Type type){  
    mv.visitInsn(type.getOpcode(0pcodes.IASTORE));  
}  
  
// -----  
// Instructions to manage the stack  
// -----  
  
/**  
 * Generates a POP instruction.  
 */  
public void pop(){  
    mv.visitInsn(0pcodes.POP);  
}  
  
/**  
 * Generates a POP2 instruction.  
 */  
public void pop2(){  
    mv.visitInsn(0pcodes.POP2);  
}  
  
/**  
 * Generates a DUP instruction.  
 */  
public void dup(){  
    mv.visitInsn(0pcodes.DUP);  
}  
  
/**  
 * Generates a DUP2 instruction.  
 */  
public void dup2(){  
    mv.visitInsn(0pcodes.DUP2);  
}  
  
/**  
 * Generates a DUP_X1 instruction.  
 */
```

```
public void dupX1(){
    mv.visitInsn(Opcodes.DUP_X1);
}

/**
 * Generates a DUP_X2 instruction.
 */
public void dupX2(){
    mv.visitInsn(Opcodes.DUP_X2);
}

/**
 * Generates a DUP2_X1 instruction.
 */
public void dup2X1(){
    mv.visitInsn(Opcodes.DUP2_X1);
}

/**
 * Generates a DUP2_X2 instruction.
 */
public void dup2X2(){
    mv.visitInsn(Opcodes.DUP2_X2);
}

/**
 * Generates a SWAP instruction.
 */
public void swap(){
    mv.visitInsn(Opcodes.SWAP);
}

/**
 * Generates the instructions to swap the top two stack values.
 *
 * @param prev type of the top - 1 stack value.
 * @param type type of the top stack value.
 */
public void swap(final Type prev, final Type type){
    if(type.getSize() == 1)
    {
        if(prev.getSize() == 1)
        {
            swap(); // same as dupX1(), pop();
        }
    }
    else
    {
        dupX2();
        pop();
    }
}
```

```

        }
    else
    {
        if(prev.getSize() == 1)
        {
            dup2X1();
            pop2();
        }
        else
        {
            dup2X2();
            pop2();
        }
    }
}

// -----
// Instructions to do mathematical and logical operations
// -----


/***
 * Generates the instruction to do the specified mathematical or logical
 * operation.
 *
 * @param op a mathematical or logical operation. Must be one of ADD,
 *           SUB, MUL, DIV, REM, NEG, SHL, SHR, USHR, AND, OR, XOR.
 * @param type the type of the operand(s) for this operation.
 */
public void math(final int op, final Type type){
    mv.visitInsn(type.getOpcode(op));
}

/***
 * Generates the instructions to compute the bitwise negation of the top
 * stack value.
 */
public void not(){
    mv.visitInsn(OpCodes.ICONST_1);
    mv.visitInsn(OpCodes.IXOR);
}

/***
 * Generates the instruction to increment the given local variable.
 *
 * @param local the local variable to be incremented.
 * @param amount the amount by which the local variable must be
 *               incremented.
 */
public void iinc(final int local, final int amount){
    mv.visitIincInsn(local, amount);
}

```

```
}

/***
 * Generates the instructions to cast a numerical value from one type to
 * another.
 *
 * @param from the type of the top stack value
 * @param to   the type into which this value must be cast.
 */
public void cast(final Type from, final Type to){
    if(from != to)
    {
        if(from == Type.DOUBLE_TYPE)
        {
            if(to == Type.FLOAT_TYPE)
            {
                mv.visitInsn(OpCodes.D2F);
            }
            else if(to == Type.LONG_TYPE)
            {
                mv.visitInsn(OpCodes.D2L);
            }
            else
            {
                mv.visitInsn(OpCodes.D2I);
                cast(Type.INT_TYPE, to);
            }
        }
        else if(from == Type.FLOAT_TYPE)
        {
            if(to == Type.DOUBLE_TYPE)
            {
                mv.visitInsn(OpCodes.F2D);
            }
            else if(to == Type.LONG_TYPE)
            {
                mv.visitInsn(OpCodes.F2L);
            }
            else
            {
                mv.visitInsn(OpCodes.F2I);
                cast(Type.INT_TYPE, to);
            }
        }
        else if(from == Type.LONG_TYPE)
        {
            if(to == Type.DOUBLE_TYPE)
            {
                mv.visitInsn(OpCodes.L2D);
            }
        }
    }
}
```

```

        else if(to == Type.FLOAT_TYPE)
        {
            mv.visitInsn(Opcodes.L2F);
        }
        else
        {
            mv.visitInsn(Opcodes.L2I);
            cast(Type.INT_TYPE, to);
        }
    }
    else
    {
        if(to == Type.BYTE_TYPE)
        {
            mv.visitInsn(Opcodes.I2B);
        }
        else if(to == Type.CHAR_TYPE)
        {
            mv.visitInsn(Opcodes.I2C);
        }
        else if(to == Type.DOUBLE_TYPE)
        {
            mv.visitInsn(Opcodes.I2D);
        }
        else if(to == Type.FLOAT_TYPE)
        {
            mv.visitInsn(Opcodes.I2F);
        }
        else if(to == Type.LONG_TYPE)
        {
            mv.visitInsn(Opcodes.I2L);
        }
        else if(to == Type.SHORT_TYPE)
        {
            mv.visitInsn(Opcodes.I2S);
        }
    }
}
}

// -----
// Instructions to do boxing and unboxing operations
// -----


/***
 * Generates the instructions to box the top stack value. This value is
 * replaced by its boxed equivalent on top of the stack.
 *
 * @param type the type of the top stack value.
 */

```

```
public void box(final Type type){
    if(type.getSort() == Type.OBJECT || type.getSort() == Type.ARRAY)
    {
        return;
    }
    if(type == Type.VOID_TYPE)
    {
        push((String) null);
    }
    else
    {
        Type boxed = type;
        switch(type.getSort())
        {
            case Type.BYTE:
                boxed = BYTE_TYPE;
                break;
            case Type.BOOLEAN:
                boxed = BOOLEAN_TYPE;
                break;
            case Type.SHORT:
                boxed = SHORT_TYPE;
                break;
            case Type.CHAR:
                boxed = CHARACTER_TYPE;
                break;
            case Type.INT:
                boxed = INTEGER_TYPE;
                break;
            case Type.FLOAT:
                boxed = FLOAT_TYPE;
                break;
            case Type.LONG:
                boxed = LONG_TYPE;
                break;
            case Type.DOUBLE:
                boxed = DOUBLE_TYPE;
                break;
        }
        newInstance(boxed);
        if(type.getSize() == 2)
        {
            // Pp -> Ppo -> oPpo -> ooPpo -> ooPp -> o
            dupX2();
            dupX2();
            pop();
        }
    }
}
```

```

        dupX1();
        swap();
    }
    invokeConstructor(boxed, new Method("<init>",
                                         Type.VOID_TYPE,
                                         new Type[]{type}));
}
}

/**
 * Generates the instructions to unbox the top stack value. This value is
 * replaced by its unboxed equivalent on top of the stack.
 *
 * @param type the type of the top stack value.
 */
public void unbox(final Type type){
    Type t = NUMBER_TYPE;
    Method sig = null;
    switch(type.getSort()){
        case Type.VOID:
            return;
        case Type.CHAR:
            t = CHARACTER_TYPE;
            sig = CHAR_VALUE;
            break;
        case Type.BOOLEAN:
            t = BOOLEAN_TYPE;
            sig = BOOLEAN_VALUE;
            break;
        case Type.DOUBLE:
            sig = DOUBLE_VALUE;
            break;
        case Type.FLOAT:
            sig = FLOAT_VALUE;
            break;
        case Type.LONG:
            sig = LONG_VALUE;
            break;
        case Type.INT:
        case Type.SHORT:
        case Type.BYTE:
            sig = INT_VALUE;
    }
    if(sig == null)
    {
        checkCast(type);
    }
    else
    {

```

```
        checkCast(t);
        invokeVirtual(t, sig);
    }
}

// -----
// Instructions to jump to other instructions
// -----


/***
 * Creates a new {@link Label}.
 *
 * @return a new {@link Label}.
 */
public Label newLabel(){
    return new Label();
}

/***
 * Marks the current code position with the given label.
 *
 * @param label a label.
 */
public void mark(final Label label){
    mv.visitLabel(label);
}

/***
 * Marks the current code position with a new label.
 *
 * @return the label that was created to mark the current code position.
 */
public Label mark(){
    Label label = new Label();
    mv.visitLabel(label);
    return label;
}

/***
 * Generates the instructions to jump to a label based on the
 * comparison of the top two stack values.
 *
 * @param type the type of the top two stack values.
 * @param mode how these values must be compared. One of EQ, NE, LT,
 *             GE, GT, LE.
 * @param label where to jump if the comparison result is <tt>true</tt>.
 */
public void ifCmp(final Type type, final int mode, final Label label){
    int intOp = -1;
    switch(type.getSort())
```

```

{
case Type.LONG:
    mv.visitInsn(0pcodes.LCMP);
    break;
case Type.DOUBLE:
    mv.visitInsn(0pcodes.DCMPG);
    break;
case Type.FLOAT:
    mv.visitInsn(0pcodes.FCMPG);
    break;
case Type.ARRAY:
case Type.OBJECT:
    switch(mode)
    {
        case EQ:
            mv.visitJumpInsn(0pcodes.IF_ACMPEQ, label);
            return;
        case NE:
            mv.visitJumpInsn(0pcodes.IF_ACMPNE, label);
            return;
    }
    throw new IllegalArgumentException("Bad comparison for type "
        + type);
default:
    switch(mode)
    {
        case EQ:
            intOp = 0pcodes.IF_ICMPEQ;
            break;
        case NE:
            intOp = 0pcodes.IF_ICMPNE;
            break;
        case GE:
            intOp = 0pcodes.IF_ICMPGE;
            break;
        case LT:
            intOp = 0pcodes.IF_ICMPLT;
            break;
        case LE:
            intOp = 0pcodes.IF_ICMPLE;
            break;
        case GT:
            intOp = 0pcodes.IF_ICMPGT;
            break;
    }
    mv.visitJumpInsn(intOp, label);
    return;
}
int jumpMode = mode;
switch(mode)

```

```
{  
    case GE:  
        jumpMode = LT;  
        break;  
    case LE:  
        jumpMode = GT;  
        break;  
    }  
    mv.visitJumpInsn(jumpMode, label);  
}  
  
/**  
 * Generates the instructions to jump to a label based on the  
 * comparison of the top two integer stack values.  
 *  
 * @param mode how these values must be compared. One of EQ, NE, LT,  
 *             GE, GT, LE.  
 * @param label where to jump if the comparison result is <tt>true</tt>.   
 */  
public void ifICmp(final int mode, final Label label){  
    ifCmp(Type.INT_TYPE, mode, label);  
}  
  
/**  
 * Generates the instructions to jump to a label based on the  
 * comparison of the top integer stack value with zero.  
 *  
 * @param mode how these values must be compared. One of EQ, NE, LT,  
 *             GE, GT, LE.  
 * @param label where to jump if the comparison result is <tt>true</tt>.   
 */  
public void ifZCmp(final int mode, final Label label){  
    mv.visitJumpInsn(mode, label);  
}  
  
/**  
 * Generates the instruction to jump to the given label if the top stack  
 * value is null.  
 *  
 * @param label where to jump if the condition is <tt>true</tt>.   
 */  
public void ifNull(final Label label){  
    mv.visitJumpInsn(Opcodes.IFNULL, label);  
}  
  
/**  
 * Generates the instruction to jump to the given label if the top stack  
 * value is not null.  
 *  
 * @param label where to jump if the condition is <tt>true</tt>.   
 */
```

```

/*
public void ifNonNull(final Label label){
    mv.visitJumpInsn(Opcodes.IFNONNULL, label);
}

/**
 * Generates the instruction to jump to the given label.
 *
 * @param label where to jump if the condition is <tt>true</tt>.
 */
public void goTo(final Label label){
    mv.visitJumpInsn(Opcodes.GOTO, label);
}

/**
 * Generates a RET instruction.
 *
 * @param local a local variable identifier, as returned by
 *              {@link LocalVariablesSorter#newLocal(Type) newLocal()}.
 */
public void ret(final int local){
    mv.visitVarInsn(Opcodes.RET, local);
}

/**
 * Generates the instructions for a switch statement.
 *
 * @param keys      the switch case keys.
 * @param generator a generator to generate the code for the switch
 *                  cases.
 */
public void tableSwitch(
    final int[] keys,
    final TableSwitchGenerator generator){
    float density;
    if(keys.length == 0)
    {
        density = 0;
    }
    else
    {
        density = (float) keys.length
                  / (keys[keys.length - 1] - keys[0] + 1);
    }
    tableSwitch(keys, generator, density >= 0.5f);
}

/**
 * Generates the instructions for a switch statement.
 *

```

```

* @param keys      the switch case keys.
* @param generator a generator to generate the code for the switch
*                  cases.
* @param useTable  <tt>true</tt> to use a TABLESWITCH instruction, or
*                  <tt>false</tt> to use a LOOKUPSWITCH instruction.
*/
public void tableSwitch(
    final int[] keys,
    final TableSwitchGenerator generator,
    final boolean useTable){
    for(int i = 1; i < keys.length; ++i)
    {
        if(keys[i] < keys[i - 1])
        {
            throw new IllegalArgumentException(
                "keys must be sorted ascending");
        }
    }
    Label def = newLabel();
    Label end = newLabel();
    if(keys.length > 0)
    {
        int len = keys.length;
        int min = keys[0];
        int max = keys[len - 1];
        int range = max - min + 1;
        if(useTable)
        {
            Label[] labels = new Label[range];
            Arrays.fill(labels, def);
            for(int i = 0; i < len; ++i)
            {
                labels[keys[i] - min] = newLabel();
            }
            mv.visitTableSwitchInsn(min, max, def, labels);
            for(int i = 0; i < range; ++i)
            {
                Label label = labels[i];
                if(label != def)
                {
                    mark(label);
                    generator.generateCase(i + min, end);
                }
            }
        }
        else
        {
            Label[] labels = new Label[len];
            for(int i = 0; i < len; ++i)
            {

```

```

        labels[i] = newLabel();
    }
    mv.visitLookupSwitchInsn(def, keys, labels);
    for(int i = 0; i < len; ++i)
    {
        mark(labels[i]);
        generator.generateCase(keys[i], end);
    }
}
mark(def);
generator.generateDefault();
mark(end);
}

/**
 * Generates the instruction to return the top stack value to the caller.
 */
public void returnValue(){
    mv.visitInsn(returnType.getOpcode(Opcodes.IRETURN));
}

// -----
// Instructions to load and store fields
// -----


/**
 * Generates a get field or set field instruction.
 *
 * @param opcode    the instruction's opcode.
 * @param ownerType the class in which the field is defined.
 * @param name      the name of the field.
 * @param fieldType the type of the field.
 */
private void fieldInsn(
    final int opcode,
    final Type ownerType,
    final String name,
    final Type fieldType){
    mv.visitFieldInsn(opcode,
                     ownerType.getInternalName(),
                     name,
                     fieldType.getDescriptor());
}

/**
 * Generates the instruction to push the value of a static field on the
 * stack.
 *
 * @param owner the class in which the field is defined.
 */

```

```
* @param name the name of the field.
* @param type the type of the field.
*/
public void getStatic(final Type owner,
                      final String name,
                      final Type type){
    fieldInsn(OpCodes.GETSTATIC, owner, name, type);
}

/**
 * Generates the instruction to store the top stack value
 * in a static field.
 *
 * @param owner the class in which the field is defined.
 * @param name the name of the field.
 * @param type the type of the field.
 */
public void putStatic(final Type owner,
                      final String name,
                      final Type type){
    fieldInsn(OpCodes.PUTSTATIC, owner, name, type);
}

/**
 * Generates the instruction to push the value of a non static field
 * on the stack.
 *
 * @param owner the class in which the field is defined.
 * @param name the name of the field.
 * @param type the type of the field.
 */
public void getField(final Type owner,
                     final String name,
                     final Type type){
    fieldInsn(OpCodes.GETFIELD, owner, name, type);
}

/**
 * Generates the instruction to store the top stack value in a
 * non static field.
 *
 * @param owner the class in which the field is defined.
 * @param name the name of the field.
 * @param type the type of the field.
 */
public void putField(final Type owner,
                     final String name,
                     final Type type){
    fieldInsn(OpCodes.PUTFIELD, owner, name, type);
}
```

```
// -----
// Instructions to invoke methods
// -----
```

```
/*
 * Generates an invoke method instruction.
 *
 * @param opcode the instruction's opcode.
 * @param type the class in which the method is defined.
 * @param method the method to be invoked.
 */
private void invokeInsn(
    final int opcode,
    final Type type,
    final Method method){
    String owner = type.getSort() == Type.ARRAY
        ? type.getDescriptor()
        : type.getInternalName();
    mv.visitMethodInsn(opcode,
        owner,
        method.getName(),
        method.getDescriptor());
}
```

```
/*
 * Generates the instruction to invoke a normal method.
 *
 * @param owner the class in which the method is defined.
 * @param method the method to be invoked.
 */
public void invokeVirtual(final Type owner, final Method method){
    invokeInsn(Opcodes.INVOKEVIRTUAL, owner, method);
}
```

```
/*
 * Generates the instruction to invoke a constructor.
 *
 * @param type the class in which the constructor is defined.
 * @param method the constructor to be invoked.
 */
public void invokeConstructor(final Type type, final Method method){
    invokeInsn(Opcodes.INVOKESPECIAL, type, method);
}
```

```
/*
 * Generates the instruction to invoke a static method.
 *
 * @param owner the class in which the method is defined.
 * @param method the method to be invoked.
 */
```

```
/*
public void invokeStatic(final Type owner, final Method method){
    invokeInsn(OpCodes.INVOKESTATIC, owner, method);
}

/**
 * Generates the instruction to invoke an interface method.
 *
 * @param owner the class in which the method is defined.
 * @param method the method to be invoked.
 */
public void invokeInterface(final Type owner, final Method method){
    invokeInsn(OpCodes.INVOKEINTERFACE, owner, method);
}

// -----
// Instructions to create objects and arrays
// -----


/**
 * Generates a type dependent instruction.
 *
 * @param opcode the instruction's opcode.
 * @param type the instruction's operand.
 */
private void typeInsn(final int opcode, final Type type){
    String desc;
    if(type.getSort() == Type.ARRAY)
    {
        desc = type.getDescriptor();
    }
    else
    {
        desc = type.getInternalName();
    }
    mv.visitTypeInsn(opcode, desc);
}

/**
 * Generates the instruction to create a new object.
 *
 * @param type the class of the object to be created.
 */
public void newInstance(final Type type){
    typeInsn(OpCodes.NEW, type);
}

/**
 * Generates the instruction to create a new array.
 *
```

```

 * @param type the type of the array elements.
 */
public void newArray(final Type type){
    int typ;
    switch(type.getSort())
    {
        case Type.BOOLEAN:
            typ = Opcodes.T_BOOLEAN;
            break;
        case Type.CHAR:
            typ = Opcodes.T_CHAR;
            break;
        case Type.BYTE:
            typ = Opcodes.T_BYTE;
            break;
        case Type.SHORT:
            typ = Opcodes.T_SHORT;
            break;
        case Type.INT:
            typ = Opcodes.T_INT;
            break;
        case Type.FLOAT:
            typ = Opcodes.T_FLOAT;
            break;
        case Type.LONG:
            typ = Opcodes.T_LONG;
            break;
        case Type.DOUBLE:
            typ = Opcodes.T_DOUBLE;
            break;
        default:
            typeInsn(Opcodes.ANEWARRAY, type);
            return;
    }
    mv.visitIntInsn(Opcodes.NEWARRAY, typ);
}

// -----
// Miscellaneous instructions
// -----


/***
 * Generates the instruction to compute the length of an array.
 */
public void arrayLength(){
    mv.visitInsn(Opcodes.ARRAYLENGTH);
}

/***
 * Generates the instruction to throw an exception.
*/

```

```
*/  
public void throwException(){  
    mv.visitInsn(Opcodes.ATHROW);  
}  
  
/**  
 * Generates the instructions to create and throw an exception. The  
 * exception class must have a constructor with a single String argument.  
 *  
 * @param type the class of the exception to be thrown.  
 * @param msg the detailed message of the exception.  
 */  
public void throwException(final Type type, final String msg){  
    newInstance(type);  
    dup();  
    push(msg);  
    invokeConstructor(type, Method.getMethod("void <init> (String)"));  
    throwException();  
}  
  
/**  
 * Generates the instruction to check that the top stack value is of the  
 * given type.  
 *  
 * @param type a class or interface type.  
 */  
public void checkCast(final Type type){  
    if(!type.equals(OBJECT_TYPE))  
    {  
        typeInsn(Opcodes.CHECKCAST, type);  
    }  
}  
  
/**  
 * Generates the instruction to test if the top stack value is  
 * of the given type.  
 *  
 * @param type a class or interface type.  
 */  
public void instanceOf(final Type type){  
    typeInsn(Opcodes.INSTANCEOF, type);  
}  
  
/**  
 * Generates the instruction to get the monitor of the top stack value.  
 */  
public void monitorEnter(){  
    mv.visitInsn(Opcodes.MONITORENTER);  
}
```

```

/**
 * Generates the instruction to release the monitor of
 * the top stack value.
 */
public void monitorExit(){
    mv.visitInsn(OpCodes.MONITOREXIT);
}

// -----
// Non instructions
// -----


/**
 * Marks the end of the visited method.
 */
public void endMethod(){
    if((access & OpCodes.ACC_ABSTRACT) == 0)
    {
        mv.visitMaxs(0, 0);
    }
    mv.visitEnd();
}

/**
 * Marks the start of an exception handler.
 *
 * @param start      beginning of the exception handler's scope
 *                   (inclusive).
 * @param end        end of the exception handler's scope (exclusive).
 * @param exception internal name of the type of exceptions handled
 *                   by the handler.
 */
public void catchException(
    final Label start,
    final Label end,
    final Type exception){
    mv.visitTryCatchBlock(start, end, mark(),
        exception.getInternalName());
}
}

```

8.6 LocalVariablesSorter.java

(MethodAdapter [314])
— LocalVariablesSorter.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm.commons;

import clojure.asm.Label;
import clojure.asm.MethodAdapter;
import clojure.asm.MethodVisitor;
import clojure.asm.Opcodes;
import clojure.asm.Type;

/**
 * A {@link MethodAdapter} that renames local variables in their
 * order of appearance. This adapter allows one to easily add new
 * local variables to a method. It may be used by inheriting from
 * this class, but the preferred way of using it is via delegation:
 * the next visitor in the chain can indeed add new locals when needed
 * by calling {@link #newLocal} on this adapter (this requires a
 * reference back to this {@link LocalVariablesSorter}).
 *
 * @author Chris Nokleberg
 * @author Eugene Kuleshov
 * @author Eric Bruneton
 */
public class LocalVariablesSorter extends MethodAdapter{

    private final static Type OBJECT_TYPE =
        Type.getObjectType("java/lang/Object");

    /**
     * Mapping from old to new local variable indexes. A local variable
     * at index i of size 1 is remapped to 'mapping[2*i]', while a local
     * variable at index i of size 2 is remapped to 'mapping[2*i+1]'.
     */
    private int[] mapping = new int[40];

    /**
     * Array used to store stack map local variable types after remapping.
     */
    private Object[] newLocals = new Object[20];

    /**
     * Index of the first local variable, after formal parameters.
     */
    protected final int firstLocal;

    /**
     * Index of the next local variable to be created by {@link #newLocal}.
     */
    protected int nextLocal;

    /**

```

```

 * Indicates if at least one local variable has moved due to remapping.
 */
private boolean changed;

/**
 * Creates a new {@link LocalVariablesSorter}.
 *
 * @param access access flags of the adapted method.
 * @param desc   the method's descriptor (see {@link Type Type}).
 * @param mv     the method visitor to which this adapter delegates
 *               calls.
 */
public LocalVariablesSorter(
    final int access,
    final String desc,
    final MethodVisitor mv){
    super(mv);
    Type[] args = Type.getArgumentTypes(desc);
    nextLocal = (Opcodes.ACC_STATIC & access) != 0 ? 0 : 1;
    for(int i = 0; i < args.length; i++)
    {
        nextLocal += args[i].getSize();
    }
    firstLocal = nextLocal;
}

public void visitVarInsn(final int opcode, final int var){
    Type type;
    switch(opcode)
    {
        case Opcodes.LLOAD:
        case Opcodes.LSTORE:
            type = Type.LONG_TYPE;
            break;

        case Opcodes.DLOAD:
        case Opcodes.DSTORE:
            type = Type.DOUBLE_TYPE;
            break;

        case Opcodes.FLOAD:
        case Opcodes.FSTORE:
            type = Type.FLOAT_TYPE;
            break;

        case Opcodes.ILOAD:
        case Opcodes.ISTORE:
            type = Type.INT_TYPE;
            break;
    }
}

```

```

        case Opcodes.ALOAD:
        case Opcodes.ASTORE:
            type = OBJECT_TYPE;
            break;

        // case RET:
        default:
            type = Type.VOID_TYPE;
        }
        mv.visitVarInsn(opcode, remap(var, type));
    }

    public void visitIincInsn(final int var, final int increment){
        mv.visitIincInsn(remap(var, Type.INT_TYPE), increment);
    }

    public void visitMaxs(final int maxStack, final int maxLocals){
        mv.visitMaxs(maxStack, nextLocal);
    }

    public void visitLocalVariable(
        final String name,
        final String desc,
        final String signature,
        final Label start,
        final Label end,
        final int index){
        int size = "J".equals(desc) || "D".equals(desc) ? 2 : 1;
        int newIndex = remap(index, size);
        mv.visitLocalVariable(name, desc, signature, start, end, newIndex);
    }

    public void visitFrame(
        final int type,
        final int nLocal,
        final Object[] local,
        final int nStack,
        final Object[] stack){
        if(type != Opcodes.F_NEW)
            { // uncompressed frame
            throw new IllegalStateException(
            "ClassReader.accept() should be called with EXPAND_FRAMES flag");
            }

        if(!changed)
            { // optimization for the case where mapping = identity
            mv.visitFrame(type, nLocal, local, nStack, stack);
            return;
            }
    }
}

```

```

// creates a copy of newLocals
Object[] oldLocals = new Object[newLocals.length];
System.arraycopy(newLocals, 0, oldLocals, 0, oldLocals.length);

// copies types from 'local' to 'newLocals'
// 'newLocals' already contains the variables added with 'newLocal'

int index = 0; // old local variable index
int number = 0; // old local variable number
for(; number < nLocal; ++number)
{
    Object t = local[number];
    int size = t == Opcodes.LONG ||
               t == Opcodes.DOUBLE ? 2 : 1;
    if(t != Opcodes.TOP)
    {
        setFrameLocal(remap(index, size), t);
    }
    index += size;
}

// removes TOP after long and double types as well as trailing TOPs

index = 0;
number = 0;
for(int i = 0; index < newLocals.length; ++i)
{
    Object t = newLocals[index++];
    if(t != null && t != Opcodes.TOP)
    {
        newLocals[i] = t;
        number = i + 1;
        if(t == Opcodes.LONG || t == Opcodes.DOUBLE)
        {
            index += 1;
        }
    }
    else
    {
        newLocals[i] = Opcodes.TOP;
    }
}

// visits remapped frame
mv.visitFrame(type, number, newLocals, nStack, stack);

// restores original value of 'newLocals'
newLocals = oldLocals;
}

```

```
// -----  
  
/**  
 * Creates a new local variable of the given type.  
 *  
 * @param type the type of the local variable to be created.  
 * @return the identifier of the newly created local variable.  
 */  
public int newLocal(final Type type){  
    Object t;  
    switch(type.getSort())  
    {  
        case Type.BOOLEAN:  
        case Type.CHAR:  
        case Type.BYTE:  
        case Type.SHORT:  
        case Type.INT:  
            t = Opcodes.INTEGER;  
            break;  
        case Type.FLOAT:  
            t = Opcodes.FLOAT;  
            break;  
        case Type.LONG:  
            t = Opcodes.LONG;  
            break;  
        case Type.DOUBLE:  
            t = Opcodes.DOUBLE;  
            break;  
        case Type.ARRAY:  
            t = type.getDescriptor();  
            break;  
        // case Type.OBJECT:  
        default:  
            t = type.getInternalName();  
            break;  
    }  
    int local = nextLocal;  
    setLocalType(local, type);  
    setFrameLocal(local, t);  
    nextLocal += type.getSize();  
    return local;  
}  
  
/**  
 * Sets the current type of the given local variable. The default  
 * implementation of this method does nothing.  
 *  
 * @param local a local variable identifier, as returned by  
 *              {@link #newLocal newLocal()}.  
 * @param type the type of the value being stored in the local
```

```

*
     variable
*/
protected void setLocalType(final int local, final Type type){
}

private void setFrameLocal(final int local, final Object type){
    int l = newLocals.length;
    if(local >= l)
    {
        Object[] a = new Object[Math.max(2 * l, local + 1)];
        System.arraycopy(newLocals, 0, a, 0, l);
        newLocals = a;
    }
    newLocals[local] = type;
}

private int remap(final int var, final Type type){
    if(var < firstLocal)
    {
        return var;
    }
    int key = 2 * var + type.getSize() - 1;
    int size = mapping.length;
    if(key >= size)
    {
        int[] newMapping = new int[Math.max(2 * size, key + 1)];
        System.arraycopy(mapping, 0, newMapping, 0, size);
        mapping = newMapping;
    }
    int value = mapping[key];
    if(value == 0)
    {
        value = nextLocal + 1;
        mapping[key] = value;
        setLocalType(nextLocal, type);
        nextLocal += type.getSize();
    }
    if(value - 1 != var)
    {
        changed = true;
    }
    return value - 1;
}

private int remap(final int var, final int size){
    if(var < firstLocal || !changed)
    {
        return var;
    }
    int key = 2 * var + size - 1;
}

```

```
int value = key < mapping.length ? mapping[key] : 0;
if(value == 0)
{
    throw new
        IllegalStateException("Unknown local variable " + var);
}
return value - 1;
}
```

8.7 Method.java

— Method.java —

```
\getchunk{France Telecom Copyright}
package clojure.asm.commons;

import java.util.HashMap;
import java.util.Map;

import clojure.asm.Type;

/**
 * A named method descriptor.
 *
 * @author Juozas Baliuka
 * @author Chris Nokleberg
 * @author Eric Bruneton
 */
public class Method{

    /**
     * The method name.
     */
    private final String name;

    /**
     * The method descriptor.
     */
    private final String desc;

    /**
     * Maps primitive Java type names to their descriptors.
     */
    private final static Map DESCRIPTORS;
```

```

static
{
    DESCRIPTORS = new HashMap();
    DESCRIPTORS.put("void", "V");
    DESCRIPTORS.put("byte", "B");
    DESCRIPTORS.put("char", "C");
    DESCRIPTORS.put("double", "D");
    DESCRIPTORS.put("float", "F");
    DESCRIPTORS.put("int", "I");
    DESCRIPTORS.put("long", "J");
    DESCRIPTORS.put("short", "S");
    DESCRIPTORS.put("boolean", "Z");
}

/**
 * Creates a new {@link Method}.
 *
 * @param name the method's name.
 * @param desc the method's descriptor.
 */
public Method(final String name, final String desc){
    this.name = name;
    this.desc = desc;
}

/**
 * Creates a new {@link Method}.
 *
 * @param name      the method's name.
 * @param returnType the method's return type.
 * @param argumentTypes the method's argument types.
 */
public Method(
    final String name,
    final Type returnType,
    final Type[] argumentTypes){
    this(name, Type.getMethodDescriptor(returnType, argumentTypes));
}

/**
 * Returns a {@link Method} corresponding to the given Java method
 * declaration.
 *
 * @param method a Java method declaration, without argument names, of
 *               the form
 *               "returnType name (argumentType1, ... argumentTypeN)",
 *               where the types are in plain Java (e.g. "int", "float",
 *               "java.util.List", ...). Classes of the
 *               java.lang package can be specified by their

```

```

*
*           unqualified name; all other classes names must be
*           fully qualified.
* @return a {@link Method} corresponding to the given Java method
*         declaration.
* @throws IllegalArgumentException if <code>method</code> could not get
*           parsed.
*/
public static Method getMethod(final String method)
    throws IllegalArgumentException{
    return getMethod(method, false);
}

/**
* Returns a {@link Method} corresponding to the given Java method
* declaration.
*
* @param method      a Java method declaration, without argument
*                    names, of the form
*                    "returnType name (argumentType1,...argumentTypeN)",
*                    where the types are in plain Java (e.g. "int",
*                    "float", "java.util.List", ...). Classes of the
*                    java.lang package may be specified by their
*                    unqualified name, depending on the
*                    defaultPackage argument; all other classes
*                    names must be fully qualified.
* @param defaultPackage true if unqualified class names belong to the
*                      default package, or false if they correspond
*                      to java.lang classes. For instance "Object"
*                      means "Object" if this option is true, or
*                      "java.lang.Object" otherwise.
* @return a {@link Method} corresponding to the given Java method
*         declaration.
* @throws IllegalArgumentException if <code>method</code> could not get
*           parsed.
*/
public static Method getMethod(
    final String method,
    final boolean defaultPackage) throws IllegalArgumentException{
    int space = method.indexOf(' ');
    int start = method.indexOf('(', space) + 1;
    int end = method.indexOf(')', start);
    if(space == -1 || start == -1 || end == -1)
    {
        throw new IllegalArgumentException();
    }
    // TODO: Check validity of returnType, methodName and arguments.
    String returnType = method.substring(0, space);
    String methodName = method.substring(space + 1, start - 1).trim();
    StringBuffer sb = new StringBuffer();
    sb.append('(');

```

```

int p;
do
{
    String s;
    p = method.indexOf(',', start);
    if(p == -1)
    {
        s = map(method.substring(start, end).trim(), defaultPackage);
    }
    else
    {
        s = map(method.substring(start, p).trim(), defaultPackage);
        start = p + 1;
    }
    sb.append(s);
} while(p != -1);
sb.append(')');
sb.append(map(returnType, defaultPackage));
return new Method(methodName, sb.toString());
}

private static String map(final String type,
                        final boolean defaultPackage){
if(type.equals(""))
{
    return type;
}

StringBuffer sb = new StringBuffer();
int index = 0;
while((index = type.indexOf("[]", index) + 1) > 0)
{
    sb.append('[');
}

String t = type.substring(0, type.length() - sb.length() * 2);
String desc = (String) DESCRIPTORS.get(t);
if(desc != null)
{
    sb.append(desc);
}
else
{
    sb.append('L');
    if(t.indexOf('.') < 0)
    {
        if(!defaultPackage)
        {
            sb.append("java/lang/");
        }
    }
}
}

```

```
        sb.append(t);
    }
    else
    {
        sb.append(t.replace('.', '/'));
    }
    sb.append(';');
}
return sb.toString();
}

/**
 * Returns the name of the method described by this object.
 *
 * @return the name of the method described by this object.
 */
public String getName(){
    return name;
}

/**
 * Returns the descriptor of the method described by this object.
 *
 * @return the descriptor of the method described by this object.
 */
public String getDescriptor(){
    return desc;
}

/**
 * Returns the return type of the method described by this object.
 *
 * @return the return type of the method described by this object.
 */
public Type getReturnType(){
    return Type.getReturnType(desc);
}

/**
 * Returns the argument types of the method described by this object.
 *
 * @return the argument types of the method described by this object.
 */
public Type[] getArgumentTypes(){
    return Type.getArgumentTypes(desc);
}

public String toString(){
    return name + desc;
}
```

```

public boolean equals(final Object o){
    if(!(o instanceof Method))
    {
        return false;
    }
    Method other = (Method) o;
    return name.equals(other.name) && desc.equals(other.desc);
}

public int hashCode(){
    return name.hashCode() ^ desc.hashCode();
}
}

```

—————

8.8 SerialVersionUIDAdder.java

(ClassAdapter [183])
 — SerialVersionUIDAdder.java —

```

\getchunk{France Telecom Copyright}
package clojure.asm.commons;

import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.security.MessageDigest;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;

import clojure.asm.ClassAdapter;
import clojure.asm.ClassVisitor;
import clojure.asm.FieldVisitor;
import clojure.asm.MethodVisitor;
import clojure.asm.Opcodes;

/**
 * A {@link ClassAdapter} that adds a serial version unique identifier
 * to a class if missing. Here is typical usage of this class:
 * <p>
 * <pre>
 *   ClassWriter cw = new ClassWriter(...);
 *   ClassVisitor sv = new SerialVersionUIDAdder(cw);
 *   ClassVisitor ca = new MyClassAdapter(sv);
 *   new ClassReader(orginalClass).accept(ca, false);

```

```
* </pre>
* <p/>
* The SUID algorithm can be found on the java.sun.com website under
* "j2se/1.4.2/docs/guide/serialization/spec/class.html"
* <p/>
* <pre>
* The serialVersionUID is computed using the signature of a stream of
* bytes that reflect the class definition. The National Institute of
* Standards and Technology (NIST) Secure Hash Algorithm (SHA-1) is
* used to compute a signature for the stream. The first two 32-bit
* quantities are used to form a 64-bit hash. A
* java.lang.DataOutputStream is used to convert primitive data types
* to a sequence of bytes. The values input to the stream are defined
* by the Java Virtual Machine (VM) specification for classes.
* <p/>
* The sequence of items in the stream is as follows:
* <p/>
* 1. The class name written using UTF encoding.
* 2. The class modifiers written as a 32-bit integer.
* 3. The name of each interface sorted by name written using UTF
*    encoding.
* 4. For each field of the class sorted by field name (except
*    private static and private transient fields):
* 1. The name of the field in UTF encoding.
* 2. The modifiers of the field written as a 32-bit integer.
* 3. The descriptor of the field in UTF encoding
* 5. If a class initializer exists, write out the following:
* 1. The name of the method, &lt;clinit&gt;, in UTF encoding.
* 2. The modifier of the method, java.lang.reflect.Modifier STATIC,
*    written as a 32-bit integer.
* 3. The descriptor of the method, ()V, in UTF encoding.
* 6. For each non-private constructor sorted by method name and
*    signature:
* 1. The name of the method, &lt;init&gt;, in UTF encoding.
* 2. The modifiers of the method written as a 32-bit integer.
* 3. The descriptor of the method in UTF encoding.
* 7. For each non-private method sorted by method name and signature:
* 1. The name of the method in UTF encoding.
* 2. The modifiers of the method written as a 32-bit integer.
* 3. The descriptor of the method in UTF encoding.
* 8. The SHA-1 algorithm is executed on the stream of bytes produced by
* DataOutputStream and produces five 32-bit values sha[0..4].
* <p/>
* 9. The hash value is assembled from the first and second 32-bit
*    values of the SHA-1 message digest. If the result of the message
*    digest, the five 32-bit words H0 H1 H2 H3 H4, is in an array of
*    five int values named sha, the hash value would be computed as
*    follows:
* <p/>
* long hash = ((sha[0] &gt;&gt;&gt; 24) &amp; 0xFF) |
```

```
* ((sha[0] &gt;&gt; 16) &amp; 0xFF) &lt;&lt; 8 |
* ((sha[0] &gt;&gt; 8) &amp; 0xFF) &lt;&lt; 16 |
* ((sha[0] &gt;&gt; 0) &amp; 0xFF) &lt;&lt; 24 |
* ((sha[1] &gt;&gt; 24) &amp; 0xFF) &lt;&lt; 32 |
* ((sha[1] &gt;&gt; 16) &amp; 0xFF) &lt;&lt; 40 |
* ((sha[1] &gt;&gt; 8) &amp; 0xFF) &lt;&lt; 48 |
* ((sha[1] &gt;&gt; 0) &amp; 0xFF) &lt;&lt; 56;
* </pre>
*
* @author Rajendra Inamdar, Vishal Vishnoi
*/
public class SerialVersionUIDAdder extends ClassAdapterf

/**
 * Flag that indicates if we need to compute SVUID.
 */
protected boolean computeSVUID;

/**
 * Set to true if the class already has SVUID.
 */
protected boolean hasSVUID;

/**
 * Classes access flags.
 */
protected int access;

/**
 * Internal name of the class
 */
protected String name;

/**
 * Interfaces implemented by the class.
 */
protected String[] interfaces;

/**
 * Collection of fields. (except private static and private transient
 * fields)
 */
protected Collection svuidFields;

/**
 * Set to true if the class has static initializer.
 */
protected boolean hasStaticInitializer;

/**
```

```
* Collection of non-private constructors.  
*/  
protected Collection svuidConstructors;  
  
/**  
 * Collection of non-private methods.  
 */  
protected Collection svuidMethods;  
  
/**  
 * Creates a new {@link SerialVersionUIDAdder}.  
 *  
 * @param cv a {@link ClassVisitor} to which this visitor will delegate  
 *           calls.  
 */  
public SerialVersionUIDAdder(final ClassVisitor cv){  
    super(cv);  
    svuidFields = new ArrayList();  
    svuidConstructors = new ArrayList();  
    svuidMethods = new ArrayList();  
}  
  
// -----  
// Overridden methods  
// -----  
  
/*  
 * Visit class header and get class name, access , and interfaces  
 * informatoin (step 1,2, and 3) for SVUID computation.  
 */  
  
public void visit(  
    final int version,  
    final int access,  
    final String name,  
    final String signature,  
    final String superName,  
    final String[] interfaces){  
    computeSVUID = (access & Opcodes.ACC_INTERFACE) == 0;  
  
    if(computeSVUID)  
    {  
        this.name = name;  
        this.access = access;  
        this.interfaces = interfaces;  
    }  
  
    super.visit(version, access, name, signature, superName, interfaces);  
}
```

```

/*
 * Visit the methods and get constructor and method information
 * (step 5 and 7). Also determine if there is a class initializer
 * (step 6).
 */
public MethodVisitor visitMethod(
    final int access,
    final String name,
    final String desc,
    final String signature,
    final String[] exceptions){
    if(computeSVUID)
    {
        if(name.equals("<clinit>"))
        {
            hasStaticInitializer = true;
        }
        /*
         * Remembers non private constructors and methods for SVUID
         * computation For constructor and method modifiers, only the
         * ACC_PUBLIC, ACC_PRIVATE, ACC_PROTECTED, ACC_STATIC,
         * ACC_FINAL, ACC_SYNCHRONIZED, ACC_NATIVE, ACC_ABSTRACT and
         * ACC_STRICT flags are used.
        */
        int mods = access
            & (Opcodes.ACC_PUBLIC | Opcodes.ACC_PRIVATE
                | Opcodes.ACC_PROTECTED | Opcodes.ACC_STATIC
                | Opcodes.ACC_FINAL | Opcodes.ACC_SYNCHRONIZED
                | Opcodes.ACC_NATIVE | Opcodes.ACC_ABSTRACT
                | Opcodes.ACC_STRICT);

        // all non private methods
        if((access & Opcodes.ACC_PRIVATE) == 0)
        {
            if(name.equals("<init>"))
            {
                svuidConstructors.add(new Item(name, mods, desc));
            }
            else if(!name.equals("<clinit>"))
            {
                svuidMethods.add(new Item(name, mods, desc));
            }
        }
    }

    return cv.visitMethod(access, name, desc, signature, exceptions);
}

/*
 * Gets class field information for step 4 of the algorithm. Also

```

```

 * determines if the class already has a SVUID.
 */
public FieldVisitor visitField(
    final int access,
    final String name,
    final String desc,
    final String signature,
    final Object value){
    if(computeSVUID)
    {
        if(name.equals("serialVersionUID"))
        {
            // since the class already has SVUID, we won't be
            // computing it.
            computeSVUID = false;
            hasSVUID = true;
        }
    /*
     * Remember field for SVUID computation For field modifiers, only
     * the ACC_PUBLIC, ACC_PRIVATE, ACC_PROTECTED, ACC_STATIC,
     * ACC_FINAL, ACC_VOLATILE, and ACC_TRANSIENT flags are used when
     * computing serialVersionUID values.
    */
    int mods = access
        & (Opcodes.ACC_PUBLIC | Opcodes.ACC_PRIVATE
            | Opcodes.ACC_PROTECTED | Opcodes.ACC_STATIC
            | Opcodes.ACC_FINAL | Opcodes.ACC_VOLATILE
            | Opcodes.ACC_TRANSIENT);

    if((access & Opcodes.ACC_PRIVATE) == 0
        || (access & (Opcodes.ACC_STATIC |
                      Opcodes.ACC_TRANSIENT)) == 0)
    {
        svuidFields.add(new Item(name, mods, desc));
    }
}

return super.visitField(access, name, desc, signature, value);
}

/*
 * Add the SVUID if class doesn't have one
 */
public void visitEnd(){
    // compute SVUID and add it to the class
    if(computeSVUID && !hasSVUID)
    {
        try
        {
            cv.visitField(Opcodes.ACC_FINAL + Opcodes.ACC_STATIC,

```

```

        "serialVersionUID",
        "J",
        null,
        new Long(computeSVID()));
    }
    catch(Throwable e)
    {
        throw new RuntimeException("Error while computing SVID for "
            + name, e);
    }
}

super.visitEnd();
}

// -----
// Utility methods
// -----


/***
 * Returns the value of SVID if the class doesn't have one already.
 * Please note that 0 is returned if the class already has SVID, thus
 * use <code>isHasSVID</code> to determine if the class already had
 * an SVID.
 *
 * @return Returns the serial version UID
 * @throws IOException
 */
protected long computeSVID() throws IOException{
    ByteArrayOutputStream bos = null;
    DataOutputStream dos = null;
    long svuid = 0;

    try
    {
        bos = new ByteArrayOutputStream();
        dos = new DataOutputStream(bos);

        /*
         * 1. The class name written using UTF encoding.
         */
        dos.writeUTF(name.replace('/', '.'));

        /*
         * 2. The class modifiers written as a 32-bit integer.
         */
        dos.writeInt(access
            & (Opcodes.ACC_PUBLIC | Opcodes.ACC_FINAL
            | Opcodes.ACC_INTERFACE | Opcodes.ACC_ABSTRACT));
    }
}

```

```
/*
 * 3. The name of each interface sorted by name written using UTF
 * encoding.
 */
Arrays.sort(interfaces);
for(int i = 0; i < interfaces.length; i++)
{
    dos.writeUTF(interfaces[i].replace('/', '.'));

}

/*
 * 4. For each field of the class sorted by field name (except
 * private static and private transient fields):
 *
 * 1. The name of the field in UTF encoding. 2. The modifiers
 * of the field written as a 32-bit integer. 3. The descriptor
 * of the field in UTF encoding
 *
 * Note that field signatures are not dot separated. Method and
 * constructor signatures are dot separated. Go figure...
 */
writeItems(svuidFields, dos, false);

/*
 * 5. If a class initializer exists, write out the following: 1.
 * The name of the method, <clinit>, in UTF encoding. 2. The
 * modifier of the method, java.lang.reflect.Modifier STATIC,
 * written as a 32-bit integer. 3. The descriptor of the method,
 * ()V, in UTF encoding.
 */
if(hasStaticInitializer)
{
    dos.writeUTF("<clinit>");
    dos.writeInt(Opcodes.ACC_STATIC);
    dos.writeUTF("()V");
} // if..

/*
 * 6. For each non-private constructor sorted by method name
 * and signature: 1. The name of the method, <init>, in UTF
 * encoding. 2. The modifiers of the method written as a
 * 32-bit integer. 3. The descriptor of the method in UTF
 * encoding.
 */
writeItems(svuidConstructors, dos, true);

/*
 * 7. For each non-private method sorted by method name and
 * signature: 1. The name of the method in UTF encoding. 2. The
 * modifiers of the method written as a 32-bit integer. 3. The

```

```

        * descriptor of the method in UTF encoding.
        */
writeItems(svuidMethods, dos, true);

dos.flush();

/*
 * 8. The SHA-1 algorithm is executed on the stream of bytes
 * produced by DataOutputStream and produces five 32-bit values
 * sha[0..4].
 */
byte[] hashBytes = computeSHA1Digest(bos.toByteArray());

/*
 * 9. The hash value is assembled from the first and second
 * 32-bit values of the SHA-1 message digest. If the result
 * of the message digest, the five 32-bit words H0 H1 H2 H3 H4,
 * is in an array of five int values named sha, the hash value
 * would be computed as follows:
 *
 * long hash = ((sha[0] >>> 24) & 0xFF) |
 *              ((sha[0] >>> 16) & 0xFF) << 8 |
 *              ((sha[0] >>> 8) & 0xFF) << 16 |
 *              ((sha[0] >>> 0) & 0xFF) << 24 |
 *              ((sha[1] >>> 24) & 0xFF) << 32 |
 *              ((sha[1] >>> 16) & 0xFF) << 40 |
 *              ((sha[1] >>> 8) & 0xFF) << 48 |
 *              ((sha[1] >>> 0) & 0xFF) << 56;
 */
for(int i = Math.min(hashBytes.length, 8) - 1; i >= 0; i--)
{
    svuid = (svuid << 8) | (hashBytes[i] & 0xFF);
}
finally
{
    // close the stream (if open)
    if(dos != null)
    {
        dos.close();
    }
}

return svuid;
}

/**
 * Returns the SHA-1 message digest of the given value.
 *
 * @param value the value whose SHA message digest must be computed.

```

```
* @return the SHA-1 message digest of the given value.
*/
protected byte[] computeSHAdigest(final byte[] value){
    try
    {
        return MessageDigest.getInstance("SHA").digest(value);
    }
    catch(Exception e)
    {
        throw new UnsupportedOperationException(e);
    }
}

/**
 * Sorts the items in the collection and writes it to the data output
 * stream
 *
 * @param itemCollection collection of items
 * @param dos             a <code>DataOutputStream</code> value
 * @param dotted          a <code>boolean</code> value
 * @throws IOException if an error occurs
 */
private void writeItems(
    final Collection itemCollection,
    final DataOutputStream dos,
    final boolean dotted) throws IOException{
    int size = itemCollection.size();
    Item items[] = (Item[]) itemCollection.toArray(new Item[size]);
    Arrays.sort(items);
    for(int i = 0; i < size; i++)
    {
        dos.writeUTF(items[i].name);
        dos.writeInt(items[i].access);
        dos.writeUTF(dotted
            ? items[i].desc.replace('/', '.')
            : items[i].desc);
    }
}

// -----
// Inner classes
// -----


static class Item implements Comparable{

    String name;

    int access;

    String desc;
```

```

Item(final String name, final int access, final String desc){
    this.name = name;
    this.access = access;
    this.desc = desc;
}

public int compareTo(final Object o){
    Item other = (Item) o;
    int retVal = name.compareTo(other.name);
    if(retVal == 0)
    {
        retVal = desc.compareTo(other.desc);
    }
    return retVal;
}
}
}

```

8.9 StaticInitMerger.java

(ClassAdapter [183])
— StaticInitMerger.java —

```

\getchunk{France Telecom Copyright}
package clojure.asm.commons;

import clojure.asm.ClassAdapter;
import clojure.asm.ClassVisitor;
import clojure.asm.MethodVisitor;
import clojure.asm.Opcodes;

/**
 * A {@link ClassAdapter} that merges clinit methods into a single one.
 *
 * @author Eric Bruneton
 */
public class StaticInitMerger extends ClassAdapter{

    private String name;

    private MethodVisitor clinit;

    private String prefix;

    private int counter;

```

```
public StaticInitMerger(final String prefix, final ClassVisitor cv){
    super(cv);
    this.prefix = prefix;
}

public void visit(
    final int version,
    final int access,
    final String name,
    final String signature,
    final String superName,
    final String[] interfaces){
    cv.visit(version, access, name, signature, superName, interfaces);
    this.name = name;
}

public MethodVisitor visitMethod(
    final int access,
    final String name,
    final String desc,
    final String signature,
    final String[] exceptions){
    MethodVisitor mv;
    if(name.equals("<clinit>")){
        int a = Opcodes.ACC_PRIVATE + Opcodes.ACC_STATIC;
        String n = prefix + counter++;
        mv = cv.visitMethod(a, n, desc, signature, exceptions);

        if(clinit == null){
            clinit = cv.visitMethod(a, name, desc, null, null);
        }
        clinit.visitMethodInsn(Opcodes.INVOKESTATIC, this.name, n, desc);
    }
    else{
        mv = cv.visitMethod(access, name, desc, signature, exceptions);
    }
    return mv;
}

public void visitEnd(){
    if(clinit != null){
        clinit.visitInsn(Opcodes.RETURN);
        clinit.visitMaxs(0, 0);
    }
    cv.visitEnd();
}
```

```
}\n}
```

—————

8.10 TableSwitchGenerator.java

— TableSwitchGenerator.java —

```
\getchunk{France Telecom Copyright}\npackage clojure.asm.commons;\n\nimport clojure.asm.Label;\n\n/**\n * A code generator for switch statements.\n *\n * @author Juozas Baliuka\n * @author Chris Nokleberg\n * @author Eric Bruneton\n */\npublic interface TableSwitchGenerator{\n\n    /**\n     * Generates the code for a switch case.\n     *\n     * @param key the switch case key.\n     * @param end a label that corresponds to the end of the switch\n     *           statement.\n     */\n    void generateCase(int key, Label end);\n\n    /**\n     * Generates the code for the default switch case.\n     */\n    void generateDefault();\n}
```

—————

Chapter 9

jvm/clojure/lang/

9.1 AFn.java

```
(IFn [774])
— AFn.java —

/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 25, 2006 4:05:37 PM */

package clojure.lang;

public abstract class AFn implements IFn {

    public Object call() throws Exception{
        return invoke();
    }

    public void run(){
        try
        {
            invoke();
        }
        catch(Exception e)
        {
            throw new RuntimeException(e);
        }
    }

    public Object invoke()
}
```

```
        throws Exception{
    return throwArity(0);
}

public Object invoke(Object arg1)
        throws Exception{
    return throwArity(1);
}

public Object invoke(Object arg1, Object arg2)
        throws Exception{
    return throwArity(2);
}

public Object invoke(Object arg1, Object arg2, Object arg3)
        throws Exception{
    return throwArity(3);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
        Object arg4)
        throws Exception{
    return throwArity(4);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
        Object arg4, Object arg5)
        throws Exception{
    return throwArity(5);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
        Object arg4, Object arg5, Object arg6)
        throws Exception{
    return throwArity(6);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
        Object arg4, Object arg5, Object arg6,
        Object arg7)
        throws Exception{
    return throwArity(7);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8) throws Exception{
    return throwArity(8);
}
```

```
public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9)
    throws Exception{
    return throwArity(9);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10)
    throws Exception{
    return throwArity(10);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11)
    throws Exception{
    return throwArity(11);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12)
    throws Exception{
    return throwArity(12);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13)
    throws Exception{
    return throwArity(13);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14)
    throws Exception{
    return throwArity(14);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
```

```
        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12,
        Object arg13, Object arg14, Object arg15)
    throws Exception{
    return throwArity(15);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12,
        Object arg13, Object arg14, Object arg15,
        Object arg16)
    throws Exception{
    return throwArity(16);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12,
        Object arg13, Object arg14, Object arg15,
        Object arg16, Object arg17)
    throws Exception{
    return throwArity(17);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12,
        Object arg13, Object arg14, Object arg15,
        Object arg16, Object arg17, Object arg18)
    throws Exception{
    return throwArity(18);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12,
        Object arg13, Object arg14, Object arg15,
        Object arg16, Object arg17, Object arg18,
        Object arg19)
    throws Exception{
    return throwArity(19);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
```

```

        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12,
        Object arg13, Object arg14, Object arg15,
        Object arg16, Object arg17, Object arg18,
        Object arg19, Object arg20)
    throws Exception{
    return throwArity(20);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20, Object... args)
    throws Exception{
    return throwArity(21);
}

public Object applyTo(ISeq arglist) throws Exception{
    return applyToHelper(this, Util.ret1(arglist,arglist = null));
}

static public Object applyToHelper(IFn ifn, ISeq arglist)
throws Exception{
    switch(RT.boundedLength(arglist, 20))
    {
        case 0:
            arglist = null;
            return ifn.invoke();
        case 1:
            Object a1 = arglist.first();
            arglist = null;
            return ifn.invoke(a1);
        case 2:
            return ifn.invoke(arglist.first()
                            , Util.ret1(
                                (arglist = arglist.next()).first(),arglist = null)
                            );
        case 3:
            return ifn.invoke(arglist.first()
                            , (arglist = arglist.next()).first()
                            , Util.ret1(
                                (arglist = arglist.next()).first(),arglist = null)
                            );
        case 4:
            return ifn.invoke(arglist.first())
    }
}

```

```
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
        , Util.ret1(
            (arglist = arglist.next()).first(),arglist = null)
    );
case 5:
    return ifn.invoke(arglist.first()
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
        , Util.ret1(
            (arglist = arglist.next()).first(),arglist = null)
    );
case 6:
    return ifn.invoke(arglist.first()
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
        , Util.ret1(
            (arglist = arglist.next()).first(),arglist = null)
    );
case 7:
    return ifn.invoke(arglist.first()
        , (arglist = arglist.next()).first()
        , Util.ret1(
            (arglist = arglist.next()).first(),arglist = null)
    );
case 8:
    return ifn.invoke(arglist.first()
        , (arglist = arglist.next()).first()
        , Util.ret1(
            (arglist = arglist.next()).first(),arglist = null)
    );
case 9:
    return ifn.invoke(arglist.first()
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
        , (arglist = arglist.next()).first()
```

```
, (arglist = arglist.next()).first()
, (arglist = arglist.next()).first()
, Util.ret1(
    (arglist = arglist.next()).first(),arglist = null)
);
case 10:
    return ifn.invoke(arglist.first()
        , (arglist = arglist.next()).first()
        , Util.ret1(
            (arglist = arglist.next()).first(),arglist = null)
    );
case 11:
    return ifn.invoke(arglist.first()
        , (arglist = arglist.next()).first()
        , Util.ret1(
            (arglist = arglist.next()).first(),arglist = null)
    );
case 12:
    return ifn.invoke(arglist.first()
        , (arglist = arglist.next()).first()
        , Util.ret1(
            (arglist = arglist.next()).first(),arglist = null)
    );
case 13:
    return ifn.invoke(arglist.first()
        , (arglist = arglist.next()).first()
```

```
, (arglist = arglist.next()).first()
, Util.ret1(
    (arglist = arglist.next()).first(),arglist = null)
);
case 14:
    return ifn.invoke(arglist.first())
    , (arglist = arglist.next()).first()
    , Util.ret1(
        (arglist = arglist.next()).first(),arglist = null)
);
case 15:
    return ifn.invoke(arglist.first())
    , (arglist = arglist.next()).first()
    , Util.ret1(
        (arglist = arglist.next()).first(),arglist = null)
);
case 16:
    return ifn.invoke(arglist.first())
```

```
, (arglist = arglist.next()).first()
, Util.ret1(
    (arglist = arglist.next()).first(),arglist = null)
);
case 17:
    return ifn.invoke(arglist.first())
    , (arglist = arglist.next()).first()
    , Util.ret1(
        (arglist = arglist.next()).first(),arglist = null)
);
case 18:
    return ifn.invoke(arglist.first())
    , (arglist = arglist.next()).first()
    , (arglist = arglist.next()).first()
```



```

        , Util.ret1(
            (arglist = arglist.next()).first(), arglist = null)
    );
default:
    return ifn.invoke(arglist.first())
        , (arglist = arglist.next()).first()
        , RT.seqToArray(
            Util.ret1(arglist.next(), arglist = null)));
}
}

public Object throwArity(int n){
    String name = getClass().getSimpleName();
    int suffix = name.lastIndexOf("--");
    throw new ArityException(n,
        (suffix == -1
            ? name
            : name.substring(0, suffix)).replace('_', '-'));
}
}

```

9.2 AFunction.java

(AFn [509]) (IObj [800]) (Comparator [1723]) (Fn [772]) (Serializable [1723])
 — AFunction.java —

```
/*
\getchunk{Clojure Copyright}
```

```

*/
/* rich Dec 16, 2008 */

package clojure.lang;

import java.io.Serializable;
import java.util.Comparator;

public abstract class AFunction
    extends AFn
    implements IObj, Comparator, Fn, Serializable {

    public volatile MethodImplCache __methodImplCache;

    public int compare(Object o1, Object o2){
        try {
            Object o = invoke(o1, o2);

            if(o instanceof Boolean)
            {
                if(RT.booleanCast(o))
                    return -1;
                return RT.booleanCast(invoke(o2,o1))? 1 : 0;
            }

            Number n = (Number) o;
            return n.intValue();
        }
        catch(Exception e)
        {
            throw new RuntimeException(e);
        }
    }
}

```

9.3 Agent.java

(ARef [553])
 — Agent.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Nov 17, 2007 */

```

```
package clojure.lang;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.ThreadFactory;
import java.util.concurrent.atomic.AtomicLong;
import java.util.concurrent.atomic.AtomicReference;

public class Agent extends ARef {

    static class ActionQueue {
        public final IPersistentStack q;
        public final Throwable error; // non-null indicates fail state
        static final ActionQueue EMPTY =
            new ActionQueue(PersistentQueue.EMPTY, null);

        public ActionQueue( IPersistentStack q, Throwable error )
        {
            this.q = q;
            this.error = error;
        }
    }

    static final Keyword CONTINUE = Keyword.intern(null, "continue");
    static final Keyword FAIL = Keyword.intern(null, "fail");

    volatile Object state;
    AtomicReference<ActionQueue> aq =
        new AtomicReference<ActionQueue>(ActionQueue.EMPTY);

    volatile Keyword errorMessage = CONTINUE;
    volatile IFn errorHandler = null;

    final private static AtomicLong sendThreadPoolCounter =
        new AtomicLong(0);

    final private static AtomicLong sendOffThreadPoolCounter =
        new AtomicLong(0);

    final public static ExecutorService pooledExecutor =
        Executors.newFixedThreadPool(
            2 + Runtime.getRuntime().availableProcessors(),
            createThreadFactory("clojure-agent-send-pool-%d",
                sendThreadPoolCounter));

    final public static ExecutorService soloExecutor =
        Executors.newCachedThreadPool(
            createThreadFactory("clojure-agent-send-off-pool-%d",
                sendOffThreadPoolCounter));
}
```

```

final static ThreadLocal<IPersistentVector> nested =
    new ThreadLocal<IPersistentVector>();

private static ThreadFactory
createThreadFactory(final String format,
                    final AtomicLong threadPoolCounter) {
    return new ThreadFactory() {
        public Thread newThread(Runnable runnable) {
            Thread thread = new Thread(runnable);
            thread.setName(String.format(format,
                threadPoolCounter.getAndIncrement()));
            return thread;
        }
    };
}

public static void shutdown(){
    soloExecutor.shutdown();
    pooledExecutor.shutdown();
}

static class Action implements Runnable{
    final Agent agent;
    final IFn fn;
    final ISeq args;
    final boolean solo;

    public Action(Agent agent, IFn fn, ISeq args, boolean solo){
        this.agent = agent;
        this.args = args;
        this.fn = fn;
        this.solo = solo;
    }

    void execute(){
        try
        {
            if(solo)
                soloExecutor.execute(this);
            else
                pooledExecutor.execute(this);
        }
        catch(Throwable error)
        {
            if(agent.errorHandler != null)
            {
                try
                {
                    agent.errorHandler.invoke(agent, error);
                }
                catch(Throwable e)
                {
                    System.out.println("Error in error handler: " + e.getMessage());
                }
            }
        }
    }
}

```

```
        }
        catch(Throwable e) {} // ignore errorHandler errors
    }
}
}

static void doRun(Action action){
    try
    {
        nested.set(PersistentVector.EMPTY);

        Throwable error = null;
        try
        {
            Object oldval = action.agent.state;
            Object newval =
                action.fn.applyTo(
                    RT.cons(action.agent.state, action.args));
            action.agent.setState(newval);
            action.agent.notifyWatches(oldval,newval);
        }
        catch(Throwable e)
        {
            error = e;
        }

        if(error == null)
        {
            releasePendingSends();
        }
        else
        {
            nested.set(null); // allow errorHandler to send
            if(action.agent.errorHandler != null)
            {
                try
                {
                    action.agent.errorHandler.invoke(action.agent,
                        error);
                }
                catch(Throwable e) {} // ignore errorHandler errors
            }
            if(action.agent.errorMode == CONTINUE)
            {
                error = null;
            }
        }
    }

    boolean popped = false;
    ActionQueue next = null;
```

```

        while(!popped)
        {
            ActionQueue prior = action.agent.aq.get();
            next = new ActionQueue(prior.q.pop(), error);
            popped = action.agent.aq.compareAndSet(prior, next);
        }

        if(error == null && next.q.count() > 0)
            ((Action) next.q.peek()).execute();
        }
        finally
        {
            nested.set(null);
        }
    }

    public void run(){
        doRun(this);
    }
}

public Agent(Object state) throws Exception{
    this(state,null);
}

public Agent(Object state, IPersistentMap meta) throws Exception {
    super(meta);
    setState(state);
}

boolean setState(Object newState) throws Exception{
    validate(newState);
    boolean ret = state != newState;
    state = newState;
    return ret;
}

public Object deref() throws Exception{
    return state;
}

public Throwable getError(){
    return aq.get().error;
}

public void setErrorMode(Keyword k){
    errorMode = k;
}

public Keyword getErrorMode(){
}

```

```
        return errorMode;
    }

    public void setErrorHandler(IFn f){
        errorHandler = f;
    }

    public IFn getErrorHandler(){
        return errorHandler;
    }

    synchronized public Object restart(Object newState,
                                       boolean clearActions){
        if(getError() == null)
        {
            throw new RuntimeException("Agent does not need a restart");
        }
        validate(newState);
        state = newState;

        if(clearActions)
            aq.set(ActionQueue.EMPTY);
        else
        {
            boolean restarted = false;
            ActionQueue prior = null;
            while(!restarted)
            {
                prior = aq.get();
                restarted =
                    aq.compareAndSet(prior, new ActionQueue(prior.q, null));
            }

            if(prior.q.count() > 0)
                ((Action) prior.q.peek()).execute();
        }

        return newState;
    }

    public Object dispatch(IFn fn, ISeq args, boolean solo) {
        Throwable error = getError();
        if(error != null)
        {
            throw new RuntimeException("Agent is failed, needs restart",
                                       error);
        }
        Action action = new Action(this, fn, args, solo);
        dispatchAction(action);
    }
}
```

```

        return this;
    }

    static void dispatchAction(Action action){
        LockingTransaction trans = LockingTransaction.getRunning();
        if(trans != null)
            trans.enqueue(action);
        else if(nested.get() != null)
        {
            nested.set(nested.get().cons(action));
        }
        else
            action.agent.enqueue(action);
    }

    void enqueue(Action action){
        boolean queued = false;
        ActionQueue prior = null;
        while(!queued)
        {
            prior = aq.get();
            queued =
                aq.compareAndSet(prior,
                    new ActionQueue((IPersistentStack)prior.q.cons(action),
                        prior.error));
        }

        if(prior.q.count() == 0 && prior.error == null)
            action.execute();
    }

    public int getQueueCount(){
        return aq.get().q.count();
    }

    static public int releasePendingSends(){
        IPersistentVector sends = nested.get();
        if(sends == null)
            return 0;
        for(int i=0;i<sends.count();i++)
        {
            Action a = (Action) sends.valAt(i);
            a.agent.enqueue(a);
        }
        nested.set(PersistentVector.EMPTY);
        return sends.count();
    }
}

```

9.4 AMapEntry.java

(APersistentVector [541]) (IMapEntry [798])
— AMapEntry.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 1, 2008 */

package clojure.lang;

import java.io.StringWriter;

public abstract class AMapEntry
    extends APersistentVector implements IMapEntry{

    public Object nth(int i){
        if(i == 0)
            return key();
        else if(i == 1)
            return val();
        else
            throw new IndexOutOfBoundsException();
    }

    private IPersistentVector asVector(){
        return LazilyPersistentVector.createOwning(key(), val());
    }

    public IPersistentVector assocN(int i, Object val){
        return asVector().assocN(i, val);
    }

    public int count(){
        return 2;
    }

    public ISeq seq(){
        return asVector().seq();
    }

    public IPersistentVector cons(Object o){
        return asVector().cons(o);
    }

    public IPersistentCollection empty(){
        return null;
    }
}
```

```
public IPersistentStack pop(){
    return LazilyPersistentVector.createOwning(key());
}

public Object setValue(Object value){
    throw new UnsupportedOperationException();
}

/*
public boolean equals(Object obj){
    return APersistentVector.doEquals(this, obj);
}

public int hashCode(){
    //must match logic in APersistentVector
    return 31 * (31 + Util.hash(key())) + Util.hash(val());
//    return Util.hashCombine(
//        Util.hashCombine(0, Util.hash(key())), Util.hash(val()));
}

public String toString(){
    StringWriter sw = new StringWriter();
    try {
        RT.print(this, sw);
    }
    catch(Exception e)
    {
        //checked exceptions stink!
        throw new RuntimeException(e);
    }
    return sw.toString();
}

public int length(){
    return 2;
}

public Object nth(int i){
    if(i == 0)
        return key();
    else if(i == 1)
        return val();
    else
        throw new IndexOutOfBoundsException();
}

private IPersistentVector asVector(){
```

```
        return LazilyPersistentVector.createOwning(key(), val());
    }

    public IPersistentVector assocN(int i, Object val){
        return asVector().assocN(i, val);
    }

    public int count(){
        return 2;
    }

    public ISeq seq(){
        return asVector().seq();
    }

    public IPersistentVector cons(Object o){
        return asVector().cons(o);
    }

    public boolean containsKey(Object key){
        return asVector().containsKey(key);
    }

    public IMapEntry entryAt(Object key){
        return asVector().entryAt(key);
    }

    public Associative assoc(Object key, Object val){
        return asVector().assoc(key, val);
    }

    public Object valAt(Object key){
        return asVector().valAt(key);
    }

    public Object valAt(Object key, Object notFound){
        return asVector().valAt(key, notFound);
    }

    public Object peek(){
        return val();
    }

    public ISeq rseq() throws Exception{
        return asVector().rseq();
    }
*/
```

9.5 APersistentMap.java

(AFn [509]) (IPersistentMap [801]) (Map [1723]) (Iterable [1723]) (Serializable [1723]) (MapEquivalence [850])

— APersistentMap.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.io.Serializable;
import java.util.*;

public abstract class APersistentMap
    extends AFn
    implements IPersistentMap, Map, Iterable, Serializable,
               MapEquivalence {
    int _hash = -1;

    public String toString(){
        return RT.printString(this);
    }

    public IPersistentCollection cons(Object o){
        if(o instanceof Map.Entry)
        {
            Map.Entry e = (Map.Entry) o;

            return assoc(e.getKey(), e.getValue());
        }
        else if(o instanceof IPersistentVector)
        {
            IPersistentVector v = (IPersistentVector) o;
            if(v.count() != 2)
                throw new IllegalArgumentException(
                    "Vector arg to map conj must be a pair");
            return assoc(v.nth(0), v.nth(1));
        }

        IPersistentMap ret = this;
        for(ISeq es = RT.seq(o); es != null; es = es.next())
        {
            Map.Entry e = (Map.Entry) es.first();
            ret = ret.assoc(e.getKey(), e.getValue());
        }
    }
}
```

```
        return ret;
    }

    public boolean equals(Object obj){
        return mapEquals(this, obj);
    }

    static public boolean mapEquals(IPersistentMap m1, Object obj){
        if(m1 == obj) return true;
        if(!(obj instanceof Map))
            return false;
        Map m = (Map) obj;

        if(m.size() != m1.count() || m.hashCode() != m1.hashCode())
            return false;

        for(ISeq s = m1.seq(); s != null; s = s.next())
        {
            Map.Entry e = (Map.Entry) s.first();
            boolean found = m.containsKey(e.getKey());

            if(!found ||
               !Util.equals(e.getValue(), m.get(e.getKey())))
                return false;
        }

        return true;
    }

    public boolean equiv(Object obj){
        if(!(obj instanceof Map))
            return false;
        if(obj instanceof IPersistentMap &&
           !(obj instanceof MapEquivalence))
            return false;

        Map m = (Map) obj;

        if(m.size() != size())
            return false;

        for(ISeq s = seq(); s != null; s = s.next())
        {
            Map.Entry e = (Map.Entry) s.first();
            boolean found = m.containsKey(e.getKey());

            if(!found ||
               !Util.equiv(e.getValue(), m.get(e.getKey())))
                return false;
        }
    }
}
```

```

        return true;
    }
    public int hashCode(){
        if(_hash == -1)
        {
            this._hash = mapHash(this);
        }
        return _hash;
    }

    static public int mapHash(IPersistentMap m){
        int hash = 0;
        for(ISeq s = m.seq(); s != null; s = s.next())
        {
            Map.Entry e = (Map.Entry) s.first();
            hash += (e.getKey() == null ? 0 :
                e.getKey().hashCode()) ^
                (e.getValue() == null ? 0 : e.getValue().hashCode());
        }
        return hash;
    }

    static public class KeySeq extends ASeq{
        ISeq seq;

        static public KeySeq create(ISeq seq){
            if(seq == null)
                return null;
            return new KeySeq(seq);
        }

        private KeySeq(ISeq seq){
            this.seq = seq;
        }

        private KeySeq(IPersistentMap meta, ISeq seq){
            super(meta);
            this.seq = seq;
        }

        public Object first(){
            return ((Map.Entry) seq.first()).getKey();
        }

        public ISeq next(){
            return create(seq.next());
        }

        public KeySeq withMeta(IPersistentMap meta){

```

```
        return new KeySeq(meta, seq);
    }
}

static public class ValSeq extends ASeq{
    ISeq seq;

    static public ValSeq create(ISeq seq){
        if(seq == null)
            return null;
        return new ValSeq(seq);
    }

    private ValSeq(ISeq seq){
        this.seq = seq;
    }

    private ValSeq(IPersistentMap meta, ISeq seq){
        super(meta);
        this.seq = seq;
    }

    public Object first(){
        return ((Map.Entry) seq.first()).getValue();
    }

    public ISeq next(){
        return create(seq.next());
    }

    public ValSeq withMeta(IPersistentMap meta){
        return new ValSeq(meta, seq);
    }
}

public Object invoke(Object arg1) throws Exception{
    return valAt(arg1);
}

public Object invoke(Object arg1, Object notFound) throws Exception{
    return valAt(arg1, notFound);
}

// java.util.Map implementation

public void clear(){
    throw new UnsupportedOperationException();
}
```

```
public boolean containsValue(Object value){
    return values().contains(value);
}

public Set entrySet(){
    return new AbstractSet(){

        public Iterator iterator(){
            return APersistentMap.this.iterator();
        }

        public int size(){
            return count();
        }

        public int hashCode(){
            return APersistentMap.this.hashCode();
        }

        public boolean contains(Object o){
            if(o instanceof Entry)
            {
                Entry e = (Entry) o;
                Entry found = entryAt(e.getKey());
                if(found != null &&
                    Util.equals(found.getValue(), e.getValue()))
                    return true;
            }
            return false;
        }
    };
}

public Object get(Object key){
    return valAt(key);
}

public boolean isEmpty(){
    return count() == 0;
}

public Set keySet(){
    return new AbstractSet{

        public Iterator iterator(){
            final Iterator mi = APersistentMap.this.iterator();

            return new Iterator(){

```

```
public boolean hasNext(){
    return mi.hasNext();
}

public Object next(){
    Entry e = (Entry) mi.next();
    return e.getKey();
}

public void remove(){
    throw new UnsupportedOperationException();
}
};

};

public int size(){
    return count();
}

public boolean contains(Object o){
    return APersistentMap.this.containsKey(o);
}
};

public Object put(Object key, Object value){
    throw new UnsupportedOperationException();
}

public void putAll(Map t){
    throw new UnsupportedOperationException();
}

public Object remove(Object key){
    throw new UnsupportedOperationException();
}

public int size(){
    return count();
}

public Collection values(){
    return new AbstractCollection(){

        public Iterator iterator(){
            final Iterator mi = APersistentMap.this.iterator();

            return new Iterator(){


```

```
public boolean hasNext(){
    return mi.hasNext();
}

public Object next(){
    Entry e = (Entry) mi.next();
    return e.getValue();
}

public void remove(){
    throw new UnsupportedOperationException();
}
};

}

public int size(){
    return count();
}
};

}

/*
// java.util.Collection implementation

public Object[] toArray(){
    return RT.seqToArray(seq());
}

public boolean add(Object o){
    throw new UnsupportedOperationException();
}

public boolean remove(Object o){
    throw new UnsupportedOperationException();
}

public boolean addAll(Collection c){
    throw new UnsupportedOperationException();
}

public void clear(){
    throw new UnsupportedOperationException();
}

public boolean retainAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean removeAll(Collection c){
    throw new UnsupportedOperationException();
}
```

```
}

public boolean containsAll(Collection c){
    for(Object o : c)
    {
        if(!contains(o))
            return false;
    }
    return true;
}

public Object[] toArray(Object[] a){
    if(a.length >= count())
    {
        ISeq s = seq();
        for(int i = 0; s != null; ++i, s = s.rest())
        {
            a[i] = s.first();
        }
        if(a.length > count())
            a[count()] = null;
    }
    return a;
}
else
    return toArray();
}

public int size(){
    return count();
}

public boolean isEmpty(){
    return count() == 0;
}

public boolean contains(Object o){
    if(o instanceof Map.Entry)
    {
        Map.Entry e = (Map.Entry) o;
        Map.Entry v = entryAt(e.getKey());
        return (v != null && Util.equal(v.getValue(), e.getValue()));
    }
    return false;
}
*/
}
```

—————

9.6 APersistentSet.java

(AFn [509]) (IPersistentSet [802]) (Collection [1723]) (Set [1723]) (Serializable [1723])

— APersistentSet.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 3, 2008 */

package clojure.lang;

import java.io.Serializable;
import java.util.Collection;
import java.util.Iterator;
import java.util.Set;

public abstract class APersistentSet
    extends AFn
    implements IPersistentSet, Collection, Set, Serializable {
    int _hash = -1;
    final IPersistentMap impl;

    protected APersistentSet(IPersistentMap impl){
        this.impl = impl;
    }

    public String toString(){
        return RT.printString(this);
    }

    public boolean contains(Object key){
        return impl.containsKey(key);
    }

    public Object get(Object key){
        return impl.valAt(key);
    }

    public int count(){
        return impl.count();
    }

    public ISeq seq(){
        return RT.keys(impl);
    }

    public Object invoke(Object arg1) throws Exception{
```

```

        return get(arg1);
    }

public boolean equals(Object obj){
    if(this == obj) return true;
    if(!(obj instanceof Set))
        return false;
    Set m = (Set) obj;

    if(m.size() != count() || m.hashCode() != hashCode())
        return false;

    for(Object aM : m)
    {
        if(!contains(aM))
            return false;
    }
//    for(ISeq s = seq(); s != null; s = s.rest())
//    {
//        if(!m.contains(s.first()))
//            return false;
//    }

    return true;
}

public boolean equiv(Object o){
    return equals(o);
}

public int hashCode(){
    if(_hash == -1)
    {
        //int hash = count();
        int hash = 0;
        for(ISeq s = seq(); s != null; s = s.next())
        {
            Object e = s.first();
//            hash = Util.hashCombine(hash, Util.hash(e));
            hash += Util.hash(e);
        }
        this._hash = hash;
    }
    return _hash;
}

public Object[] toArray(){
    return RT.seqToArray(seq());
}

```

```
public boolean add(Object o){
    throw new UnsupportedOperationException();
}

public boolean remove(Object o){
    throw new UnsupportedOperationException();
}

public boolean addAll(Collection c){
    throw new UnsupportedOperationException();
}

public void clear(){
    throw new UnsupportedOperationException();
}

public boolean retainAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean removeAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean containsAll(Collection c){
    for(Object o : c)
    {
        if(!contains(o))
            return false;
    }
    return true;
}

public Object[] toArray(Object[] a){
    if(a.length >= count())
    {
        ISeq s = seq();
        for(int i = 0; s != null; ++i, s = s.next())
        {
            a[i] = s.first();
        }
        if(a.length > count())
            a[count()] = null;
        return a;
    }
    else
        return toArray();
}

public int size(){
```

```

        return count();
    }

    public boolean isEmpty(){
        return count() == 0;
    }

    public Iterator iterator(){
        return new SeqIterator(seq());
    }

}

```

—————

9.7 APersistentVector.java

(AFn [509]) (IPersistentVector [802]) (Iterable [1723]) (List [1723]) (RandomAccess [1723]) (Comparable [1723]) (Serializable [1723])

— APersistentVector.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Dec 18, 2007 */

package clojure.lang;

import java.io.Serializable;
import java.util.*;

public abstract class APersistentVector
    extends AFn
    implements IPersistentVector, Iterable, List, RandomAccess,
              Comparable, Serializable {
    int _hash = -1;

    public String toString(){
        return RT.printString(this);
    }

    public ISeq seq(){
        if(count() > 0)
            return new Seq(this, 0);
        return null;
    }

    public ISeq rseq(){

```

```

        if(count() > 0)
            return new RSeq(this, count() - 1);
        return null;
    }

    static boolean doEquals(IPersistentVector v, Object obj){
        if(v == obj) return true;
        if(obj instanceof List || obj instanceof IPersistentVector)
        {
            Collection ma = (Collection) obj;
            if(ma.size() != v.count() || ma.hashCode() != v.hashCode())
                return false;
            for(Iterator i1 = ((List) v).iterator(), i2 = ma.iterator();
                i1.hasNext();)
            {
                if(!Util.equals(i1.next(), i2.next()))
                    return false;
            }
            return true;
        }
        // if(obj instanceof IPersistentVector)
        // {
        //     IPersistentVector ma = (IPersistentVector) obj;
        //     if(ma.count() != v.count() || ma.hashCode() != v.hashCode())
        //         return false;
        //     for(int i = 0; i < v.count(); i++)
        //     {
        //         if(!Util.equal(v.nth(i), ma.nth(i)))
        //             return false;
        //     }
        // }
        else
        {
            if(!(obj instanceof Sequential))
                return false;
            ISeq ms = RT.seq(obj);
            for(int i = 0; i < v.count(); i++, ms = ms.next())
            {
                if(ms == null || !Util.equals(v.nth(i), ms.first()))
                    return false;
            }
            if(ms != null)
                return false;
        }
    }

    return true;
}

static boolean doEquiv(IPersistentVector v, Object obj){

```

```

if(obj instanceof List || obj instanceof IPersistentVector)
{
    Collection ma = (Collection) obj;
    if(ma.size() != v.count())
        return false;
    for(Iterator i1 = ((List) v).iterator(), i2 = ma.iterator();
        i1.hasNext();)
    {
        if(!Util.equiv(i1.next(), i2.next()))
            return false;
    }
    return true;
}
// if(obj instanceof IPersistentVector)
// {
//     IPersistentVector ma = (IPersistentVector) obj;
//     if(ma.count() != v.count() || ma.hashCode() != v.hashCode())
//         return false;
//     for(int i = 0; i < v.count(); i++)
//     {
//         if(!Util.equal(v.nth(i), ma.nth(i)))
//             return false;
//     }
// }
else
{
    if(!(obj instanceof Sequential))
        return false;
    ISeq ms = RT.seq(obj);
    for(int i = 0; i < v.count(); i++, ms = ms.next())
    {
        if(ms == null || !Util.equiv(v.nth(i), ms.first()))
            return false;
    }
    if(ms != null)
        return false;
}

return true;
}

public boolean equals(Object obj){
    return doEquals(this, obj);
}

public boolean equiv(Object obj){
    return doEquiv(this, obj);
}

```

```

public int hashCode(){
    if(_hash == -1)
    {
        int hash = 1;
        Iterator i = iterator();
        while(i.hasNext())
        {
            Object obj = i.next();
            hash = 31 * hash + (obj == null ? 0 : obj.hashCode());
        }
    }
    // int hash = 0;
    // for(int i = 0; i < count(); i++)
    // {
    //     hash = Util.hashCombine(hash, Util.hash(nth(i)));
    // }
    this._hash = hash;
}
return _hash;
}

public Object get(int index){
    return nth(index);
}

public Object nth(int i, Object notFound){
    if(i >= 0 && i < count())
        return nth(i);
    return notFound;
}

public Object remove(int i){
    throw new UnsupportedOperationException();
}

public int indexOf(Object o){
    for(int i = 0; i < count(); i++)
        if(Util.equiv(nth(i), o))
            return i;
    return -1;
}

public int lastIndexOf(Object o){
    for(int i = count() - 1; i >= 0; i--)
        if(Util.equiv(nth(i), o))
            return i;
    return -1;
}

public ListIterator listIterator(){
    return listIterator(0);
}

```

```
}

public ListIterator listIterator(final int index){
    return new ListIterator(){
        int nexti = index;

        public boolean hasNext(){
            return nexti < count();
        }

        public Object next(){
            return nth(nexti++);
        }

        public boolean hasPrevious(){
            return nexti > 0;
        }

        public Object previous(){
            return nth(--nexti);
        }

        public int nextIndex(){
            return nexti;
        }

        public int previousIndex(){
            return nexti - 1;
        }

        public void remove(){
            throw new UnsupportedOperationException();
        }

        public void set(Object o){
            throw new UnsupportedOperationException();
        }

        public void add(Object o){
            throw new UnsupportedOperationException();
        }
    };
}

public List subList(int fromIndex, int toIndex){
    return (List) RT.subvec(this, fromIndex, toIndex);
}

public Object set(int i, Object o){
```

```
        throw new UnsupportedOperationException();
    }

    public void add(int i, Object o){
        throw new UnsupportedOperationException();
    }

    public boolean addAll(int i, Collection c){
        throw new UnsupportedOperationException();
    }

    public Object invoke(Object arg1) throws Exception{
        if(Util.isInteger(arg1))
            return nth((Number) arg1).intValue();
        throw new IllegalArgumentException("Key must be integer");
    }

    public Iterator iterator(){
        //todo - something more efficient
        return new Iterator(){
            int i = 0;

            public boolean hasNext(){
                return i < count();
            }

            public Object next(){
                return nth(i++);
            }

            public void remove(){
                throw new UnsupportedOperationException();
            }
        };
    }

    public Object peek(){
        if(count() > 0)
            return nth(count() - 1);
        return null;
    }

    public boolean containsKey(Object key){
        if(!(Util.isInteger(key)))
            return false;
        int i = ((Number) key).intValue();
        return i >= 0 && i < count();
    }
```

```
public IMapEntry entryAt(Object key){
    if(Util.isInteger(key))
    {
        int i = ((Number) key).intValue();
        if(i >= 0 && i < count())
            return new MapEntry(key, nth(i));
    }
    return null;
}

public IPersistentVector assoc(Object key, Object val){
    if(Util.isInteger(key))
    {
        int i = ((Number) key).intValue();
        return assocN(i, val);
    }
    throw new IllegalArgumentException("Key must be integer");
}

public Object valAt(Object key, Object notFound){
    if(Util.isInteger(key))
    {
        int i = ((Number) key).intValue();
        if(i >= 0 && i < count())
            return nth(i);
    }
    return notFound;
}

public Object valAt(Object key){
    return valAt(key, null);
}

// java.util.Collection implementation

public Object[] toArray(){
    return RT.seqToArray(seq());
}

public boolean add(Object o){
    throw new UnsupportedOperationException();
}

public boolean remove(Object o){
    throw new UnsupportedOperationException();
}

public boolean addAll(Collection c){
    throw new UnsupportedOperationException();
}
```

```
public void clear(){
    throw new UnsupportedOperationException();
}

public boolean retainAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean removeAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean containsAll(Collection c){
    for(Object o : c)
    {
        if(!contains(o))
            return false;
    }
    return true;
}

public Object[] toArray(Object[] a){
    if(a.length >= count())
    {
        ISeq s = seq();
        for(int i = 0; s != null; ++i, s = s.next())
        {
            a[i] = s.first();
        }
        if(a.length > count())
            a[count()] = null;
        return a;
    }
    else
        return toArray();
}

public int size(){
    return count();
}

public boolean isEmpty(){
    return count() == 0;
}

public boolean contains(Object o){
    for(ISeq s = seq(); s != null; s = s.next())
    {
        if(Util.equiv(s.first(), o))

```

```

        return true;
    }
    return false;
}

public int length(){
    return count();
}

public int compareTo(Object o){
    IPersistentVector v = (IPersistentVector) o;
    if(count() < v.count())
        return -1;
    else if(count() > v.count())
        return 1;
    for(int i = 0; i < count(); i++)
    {
        int c = Util.compare(nth(i),v.nth(i));
        if(c != 0)
            return c;
    }
    return 0;
}

static class Seq extends ASeq implements IndexedSeq, IReduce{
//todo - something more efficient
final IPersistentVector v;
final int i;

public Seq(IPersistentVector v, int i){
    this.v = v;
    this.i = i;
}

Seq(IPersistentMap meta, IPersistentVector v, int i){
    super(meta);
    this.v = v;
    this.i = i;
}

public Object first(){
    return v.nth(i);
}

public ISeq next(){
    if(i + 1 < v.count())
        return new APersistentVector.Seq(v, i + 1);
    return null;
}
}

```

```
public int index(){
    return i;
}

public int count(){
    return v.count() - i;
}

public APersistentVector.Seq withMeta(IPersistentMap meta){
    return new APersistentVector.Seq(meta, v, i);
}

public Object reduce(IFn f) throws Exception{
    Object ret = v.nth(i);
    for(int x = i + 1; x < v.count(); x++)
        ret = f.invoke(ret, v.nth(x));
    return ret;
}

public Object reduce(IFn f, Object start) throws Exception{
    Object ret = f.invoke(start, v.nth(i));
    for(int x = i + 1; x < v.count(); x++)
        ret = f.invoke(ret, v.nth(x));
    return ret;
}
}

public static class RSeq extends ASeq implements IndexedSeq, Counted{
    final IPersistentVector v;
    final int i;

    public RSeq(IPersistentVector vector, int i){
        this.v = vector;
        this.i = i;
    }

    RSeq(IPersistentMap meta, IPersistentVector v, int i){
        super(meta);
        this.v = v;
        this.i = i;
    }

    public Object first(){
        return v.nth(i);
    }

    public ISeq next(){
        if(i > 0)
            return new APersistentVector.RSeq(v, i - 1);
    }
}
```

```
        return null;
    }

    public int index(){
        return i;
    }

    public int count(){
        return i + 1;
    }

    public APersistentVector.RSeq withMeta(IPersistentMap meta){
        return new APersistentVector.RSeq(meta, v, i);
    }
}

static class SubVector extends APersistentVector implements IObj{
    final IPersistentVector v;
    final int start;
    final int end;
    final IPersistentMap _meta;

    public SubVector(IPersistentMap meta,
                    IPersistentVector v,
                    int start,
                    int end){
        this._meta = meta;

        if(v instanceof APersistentVector.SubVector)
        {
            APersistentVector.SubVector sv =
                (APersistentVector.SubVector) v;
            start += sv.start;
            end += sv.start;
            v = sv.v;
        }
        this.v = v;
        this.start = start;
        this.end = end;
    }

    public Object nth(int i){
        if(start + i >= end)
            throw new IndexOutOfBoundsException();
        return v.nth(start + i);
    }

    public IPersistentVector assocN(int i, Object val){
```

```

        if(start + i > end)
            throw new IndexOutOfBoundsException();
        else if(start + i == end)
            return cons(val);
        return
            new SubVector(_meta, v.assocN(start + i, val), start, end);
    }

    public int count(){
        return end - start;
    }

    public IPersistentVector cons(Object o){
        return new SubVector(_meta, v.assocN(end, o), start, end + 1);
    }

    public IPersistentCollection empty(){
        return PersistentVector.EMPTY.withMeta(meta());
    }

    public IPersistentStack pop(){
        if(end - 1 == start)
        {
            return PersistentVector.EMPTY;
        }
        return new SubVector(_meta, v, start, end - 1);
    }

    public SubVector withMeta(IPersistentMap meta){
        if(meta == _meta)
            return this;
        return new SubVector(meta, v, start, end);
    }

    public IPersistentMap meta(){
        return _meta;
    }
}
}

```

—————

9.8 AReference.java

(IReference [804])
 — AReference.java —

/*

```
\getchunk{Clojure Copyright}
*/
/* rich Dec 31, 2008 */

package clojure.lang;

public class AReference implements IReference {
    private IPersistentMap _meta;

    public AReference() {
        this(null);
    }

    public AReference(IPersistentMap meta) {
        _meta = meta;
    }

    synchronized public IPersistentMap meta() {
        return _meta;
    }

    synchronized public IPersistentMap alterMeta(IFn alter, ISeq args)
        throws Exception {
        _meta = (IPersistentMap) alter.applyTo(new Cons(_meta, args));
        return _meta;
    }

    synchronized public IPersistentMap resetMeta(IPersistentMap m) {
        _meta = m;
        return m;
    }
}
```

9.9 ARef.java

(AReference [552]) (IRef [805])
 — ARef.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Jan 1, 2009 */

package clojure.lang;
```

```
import java.util.Map;

public abstract class ARef extends AReference implements IRef{
    protected volatile IFn validator = null;
    private volatile IPersistentMap watches = PersistentHashMap.EMPTY;

    public ARef(){
        super();
    }

    public ARef(IPersistentMap meta){
        super(meta);
    }

    void validate(IFn vf, Object val){
        try
        {
            if(vf != null && !RT.booleanCast(vf.invoke(val)))
                throw new IllegalStateException("Invalid reference state");
        }
        catch(RuntimeException re)
        {
            throw re;
        }
        catch(Exception e)
        {
            throw new IllegalStateException("Invalid reference state", e);
        }
    }

    void validate(Object val){
        validate.validator, val);
    }

    public void setValidator(IFn vf){
        try
        {
            validate(vf, deref());
        }
        catch(Exception e)
        {
            throw new RuntimeException(e);
        }
        validator = vf;
    }

    public IFn getValidator(){
        return validator;
    }
```

```
public IPersistentMap getWatches(){
    return watches;
}

synchronized public IRef addWatch(Object key, IFn callback){
    watches = watches.assoc(key, callback);
    return this;
}

synchronized public IRef removeWatch(Object key){
    try
    {
        watches = watches.without(key);
    }
    catch(Exception e)
    {
        throw new RuntimeException(e);
    }

    return this;
}

public void notifyWatches(Object oldval, Object newval){
    IPersistentMap ws = watches;
    if(ws.count() > 0)
    {
        for(ISeq s = ws.seq(); s != null; s = s.next())
        {
            Map.Entry e = (Map.Entry) s.first();
            IFn fn = (IFn) e.getValue();
            try
            {
                if(fn != null)
                    fn.invoke(e.getKey(), this, oldval, newval);
            }
            catch(Exception e1)
            {
                throw new RuntimeException(e1);
            }
        }
    }
}
```

9.10 ArityException.java

(IllegalArgumentException [1723])
 — ArityException.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

/**
 * @since 1.3
 */
public class ArityException extends IllegalArgumentException {

    final public int actual;

    final public String name;

    public ArityException(int actual, String name) {
        this(actual, name, null);
    }

    public ArityException(int actual, String name, Throwable cause) {
        super("Wrong number of args (" + actual +
              ") passed to: " + name, cause);
        this.actual = actual;
        this.name = name;
    }

}
```

9.11 ArrayChunk.java

(IChunk [772]) (Serializable [1723])
 — ArrayChunk.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich May 24, 2009 */

package clojure.lang;

import java.io.Serializable;
```

```
public final class ArrayChunk implements IChunk, Serializable {

    final Object[] array;
    final int off;
    final int end;

    public ArrayChunk(Object[] array){
        this(array, 0, array.length);
    }

    public ArrayChunk(Object[] array, int off){
        this(array, off, array.length);
    }

    public ArrayChunk(Object[] array, int off, int end){
        this.array = array;
        this.off = off;
        this.end = end;
    }

    public Object nth(int i){
        return array[off + i];
    }

    public Object nth(int i, Object notFound){
        if(i >= 0 && i < count())
            return nth(i);
        return notFound;
    }

    public int count(){
        return end - off;
    }

    public IChunk dropFirst(){
        if(off==end)
            throw new IllegalStateException("dropFirst of empty chunk");
        return new ArrayChunk(array, off + 1, end);
    }

    public Object reduce(IFn f, Object start) throws Exception{
        Object ret = f.invoke(start, array[off]);
        for(int x = off + 1; x < end; x++)
            ret = f.invoke(ret, array[x]);
        return ret;
    }
}
```

9.12 ArraySeq.java

```
(ASeq [571]) (IndexedSeq [799]) (IReduce [804])
— ArraySeq.java —

/*
\getchunk{Clojure Copyright}
*/
/* rich Jun 19, 2006 */

package clojure.lang;

import java.lang.reflect.Array;

public class ArraySeq extends ASeq implements IndexedSeq, IReduce{
    public final Object array;
    final int i;
    final Object[] oa;
    final Class ct;
    //ISeq _rest;

    static public ArraySeq create(){
        return null;
    }

    static public ArraySeq create(Object... array){
        if(array == null || array.length == 0)
            return null;
        return new ArraySeq(array, 0);
    }

    static ISeq createFromObject(Object array){
        if(array == null || Array.getLength(array) == 0)
            return null;
        Class aclass = array.getClass();
        if(aclass == int[].class)
            return new ArraySeq_int(null, (int[]) array, 0);
        if(aclass == float[].class)
            return new ArraySeq_float(null, (float[]) array, 0);
        if(aclass == double[].class)
            return new ArraySeq_double(null, (double[]) array, 0);
        if(aclass == long[].class)
            return new ArraySeq_long(null, (long[]) array, 0);
        if(aclass == byte[].class)
            return new ArraySeq_byte(null, (byte[]) array, 0);
        if(aclass == char[].class)
            return new ArraySeq_char(null, (char[]) array, 0);
        if(aclass == boolean[].class)
            return new ArraySeq_boolean(null, (boolean[]) array, 0);
    }
}
```

```
        return new ArraySeq(array, 0);
    }

    ArraySeq(Object array, int i){
        this.array = array;
        this.ct = array.getClass().getComponentType();
        this.i = i;
        this.oa = (Object[]) (array instanceof Object[] ? array : null);
//        this._rest = this;
    }

    ArraySeq(IPersistentMap meta, Object array, int i){
        super(meta);
        this.array = array;
        this.ct = array.getClass().getComponentType();
        this.i = i;
        this.oa = (Object[]) (array instanceof Object[] ? array : null);
    }

    public Object first(){
        if(oa != null)
            return oa[i];
        return Reflector.prepRet(ct, Array.get(array, i));
    }

    public ISeq next(){
        if(oa != null)
        {
            if(i + 1 < oa.length)
                return new ArraySeq(array, i + 1);
        }
        else
        {
            if(i + 1 < Array.getLength(array))
                return new ArraySeq(array, i + 1);
        }
        return null;
    }

    public int count(){
        if(oa != null)
            return oa.length - i;
        return Array.getLength(array) - i;
    }

    public int index(){
        return i;
    }

    public ArraySeq withMeta(IPersistentMap meta){
```

```

        return new ArraySeq(meta, array, i);
    }

    public Object reduce(IFn f) throws Exception{
        if(oa != null)
        {
            Object ret = oa[i];
            for(int x = i + 1; x < oa.length; x++)
                ret = f.invoke(ret, oa[x]);
            return ret;
        }

        Object ret = Reflector.prepRet(ct, Array.get(array, i));
        for(int x = i + 1; x < Array.getLength(array); x++)
            ret = f.invoke(ret, Reflector.prepRet(ct, Array.get(array, x)));
        return ret;
    }

    public Object reduce(IFn f, Object start) throws Exception{
        if(oa != null)
        {
            Object ret = f.invoke(start, oa[i]);
            for(int x = i + 1; x < oa.length; x++)
                ret = f.invoke(ret, oa[x]);
            return ret;
        }
        Object ret =
            f.invoke(start, Reflector.prepRet(ct, Array.get(array, i)));
        for(int x = i + 1; x < Array.getLength(array); x++)
            ret = f.invoke(ret, Reflector.prepRet(ct, Array.get(array, x)));
        return ret;
    }

    public int indexOf(Object o) {
        if (oa != null) {
            for (int j = i; j < oa.length; j++)
                if (Util.equals(o, oa[j])) return j - i;
        } else {
            int n = Array.getLength(array);
            for (int j = i; j < n; j++)
                if (Util.equals(o,
                    Reflector.prepRet(ct, Array.get(array, j))))
                    return j - i;
        }
        return -1;
    }

    public int lastIndexOf(Object o) {
        if (oa != null) {
            if (o == null) {

```

```

        for (int j = oa.length - 1 ; j >= i; j--)
            if (oa[j] == null) return j - i;
    } else {
        for (int j = oa.length - 1 ; j >= i; j--)
            if (o.equals(oa[j])) return j - i;
    }
} else {
    if (o == null) {
        for (int j = Array.getLength(array) - 1 ; j >= i; j--)
            if (Reflector.prepRet(ct,
                Array.get(array, j)) == null) return j - i;
    } else {
        for (int j = Array.getLength(array) - 1 ; j >= i; j--)
            if (o.equals(Reflector.prepRet(ct,
                Array.get(array, j)))) return j - i;
    }
}
return -1;
}

//// specialized primitive versions ////


static public class ArraySeq_int
extends ASeq implements IndexedSeq, IReduce{
    public final int[] array;
    final int i;

    ArraySeq_int(IPersistentMap meta, int[] array, int i){
        super(meta);
        this.array = array;
        this.i = i;
    }

    public Object first(){
        return array[i];
    }

    public ISeq next(){
        if(i + 1 < array.length)
            return new ArraySeq_int(meta(), array, i + 1);
        return null;
    }

    public int count(){
        return array.length - i;
    }

    public int index(){
        return i;
    }
}

```

```

public ArraySeq_int withMeta(IPersistentMap meta){
    return new ArraySeq_int(meta, array, i);
}

public Object reduce(IFn f) throws Exception{
    Object ret = array[i];
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, array[x]);
    return ret;
}

public Object reduce(IFn f, Object start) throws Exception{
    Object ret = f.invoke(start, array[i]);
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, array[x]);
    return ret;
}

public int indexOf(Object o) {
    if (o instanceof Number) {
        int k = ((Number) o).intValue();
        for (int j = i; j < array.length; j++)
            if (k == array[j]) return j - i;
    }
    return -1;
}

public int lastIndexOf(Object o) {
    if (o instanceof Number) {
        int k = ((Number) o).intValue();
        for (int j = array.length - 1; j >= i; j--)
            if (k == array[j]) return j - i;
    }
    return -1;
}

static public class ArraySeq_float
extends ASeq implements IndexedSeq, IReduce{
    public final float[] array;
    final int i;

    ArraySeq_float(IPersistentMap meta, float[] array, int i){
        super(meta);
        this.array = array;
        this.i = i;
    }
}

```

```
}

public Object first(){
    return Numbers.num(array[i]);
}

public ISeq next(){
    if(i + 1 < array.length)
        return new ArraySeq_float(meta(), array, i + 1);
    return null;
}

public int count(){
    return array.length - i;
}

public int index(){
    return i;
}

public ArraySeq_float withMeta(IPersistentMap meta){
    return new ArraySeq_float(meta, array, i);
}

public Object reduce(IFn f) throws Exception{
    Object ret = Numbers.num(array[i]);
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, Numbers.num(array[x]));
    return ret;
}

public Object reduce(IFn f, Object start) throws Exception{
    Object ret = f.invoke(start, Numbers.num(array[i]));
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, Numbers.num(array[x]));
    return ret;
}

public int indexOf(Object o) {
    if (o instanceof Number) {
        float f = ((Number) o).floatValue();
        for (int j = i; j < array.length; j++)
            if (f == array[j]) return j - i;
    }
    return -1;
}

public int lastIndexOf(Object o) {
    if (o instanceof Number) {
        float f = ((Number) o).floatValue();
```

```

        for (int j = array.length - 1; j >= i; j--)
            if (f == array[j]) return j - i;
    }
    return -1;
}
}

static public class ArraySeq_double
extends ASeq implements IndexedSeq, IReduce{
    public final double[] array;
    final int i;

    ArraySeq_double(IPersistentMap meta, double[] array, int i){
        super(meta);
        this.array = array;
        this.i = i;
    }

    public Object first(){
        return array[i];
    }

    public ISeq next(){
        if(i + 1 < array.length)
            return new ArraySeq_double(meta(), array, i + 1);
        return null;
    }

    public int count(){
        return array.length - i;
    }

    public int index(){
        return i;
    }

    public ArraySeq_double withMeta(IPersistentMap meta){
        return new ArraySeq_double(meta, array, i);
    }

    public Object reduce(IFn f) throws Exception{
        Object ret = array[i];
        for(int x = i + 1; x < array.length; x++)
            ret = f.invoke(ret, array[x]);
        return ret;
    }

    public Object reduce(IFn f, Object start) throws Exception{
        Object ret = f.invoke(start, array[i]);
        for(int x = i + 1; x < array.length; x++)

```

```
        ret = f.invoke(ret, array[x]);
        return ret;
    }

    public int indexOf(Object o) {
        if (o instanceof Number) {
            double d = ((Number) o).doubleValue();
            for (int j = i; j < array.length; j++)
                if (d == array[j]) return j - i;
        }
        return -1;
    }

    public int lastIndexOf(Object o) {
        if (o instanceof Number) {
            double d = ((Number) o).doubleValue();
            for (int j = array.length - 1; j >= i; j--)
                if (d == array[j]) return j - i;
        }
        return -1;
    }

    static public class ArraySeq_long
        extends ASeq implements IndexedSeq, IReduce{
        public final long[] array;
        final int i;

        ArraySeq_long(IPersistentMap meta, long[] array, int i){
            super(meta);
            this.array = array;
            this.i = i;
        }

        public Object first(){
            return Numbers.num(array[i]);
        }

        public ISeq next(){
            if(i + 1 < array.length)
                return new ArraySeq_long(meta(), array, i + 1);
            return null;
        }

        public int count(){
            return array.length - i;
        }
    }
}
```

```

public int index(){
    return i;
}

public ArraySeq_long withMeta(IPersistentMap meta){
    return new ArraySeq_long(meta, array, i);
}

public Object reduce(IFn f) throws Exception{
    Object ret = Numbers.num(array[i]);
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, Numbers.num(array[x]));
    return ret;
}

public Object reduce(IFn f, Object start) throws Exception{
    Object ret = f.invoke(start, Numbers.num(array[i]));
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, Numbers.num(array[x]));
    return ret;
}

public int indexOf(Object o) {
    if (o instanceof Number) {
        long l = ((Number) o).longValue();
        for (int j = i; j < array.length; j++)
            if (l == array[j]) return j - i;
    }
    return -1;
}

public int lastIndexOf(Object o) {
    if (o instanceof Number) {
        long l = ((Number) o).longValue();
        for (int j = array.length - 1; j >= i; j--)
            if (l == array[j]) return j - i;
    }
    return -1;
}

static public class ArraySeq_byte
extends ASeq implements IndexedSeq, IReduce{
    public final byte[] array;
    final int i;

    ArraySeq_byte(IPersistentMap meta, byte[] array, int i){
        super(meta);
    }
}

```

```
    this.array = array;
    this.i = i;
}

public Object first(){
    return array[i];
}

public ISeq next(){
    if(i + 1 < array.length)
        return new ArraySeq_byte(meta(), array, i + 1);
    return null;
}

public int count(){
    return array.length - i;
}

public int index(){
    return i;
}

public ArraySeq_byte withMeta(IPersistentMap meta){
    return new ArraySeq_byte(meta, array, i);
}

public Object reduce(IFn f) throws Exception{
    Object ret = array[i];
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, array[x]);
    return ret;
}

public Object reduce(IFn f, Object start) throws Exception{
    Object ret = f.invoke(start, array[i]);
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, array[x]);
    return ret;
}

public int indexOf(Object o) {
    if (o instanceof Byte) {
        byte b = ((Byte) o).byteValue();
        for (int j = i; j < array.length; j++)
            if (b == array[j]) return j - i;
    }
    if (o == null) {
        return -1;
    }
    for (int j = i; j < array.length; j++)
```

```

        if (o.equals(array[j])) return j - i;
    return -1;
}

public int lastIndexOf(Object o) {
    if (o instanceof Byte) {
        byte b = ((Byte) o).byteValue();
        for (int j = array.length - 1; j >= i; j--)
            if (b == array[j]) return j - i;
    }
    if (o == null) {
        return -1;
    }
    for (int j = array.length - 1; j >= i; j--)
        if (o.equals(array[j])) return j - i;
    return -1;
}
}

static public class ArraySeq_char
extends ASeq implements IndexedSeq, IReduce{
    public final char[] array;
    final int i;

    ArraySeq_char(IPersistentMap meta, char[] array, int i){
        super(meta);
        this.array = array;
        this.i = i;
    }

    public Object first(){
        return array[i];
    }

    public ISeq next(){
        if(i + 1 < array.length)
            return new ArraySeq_char(meta(), array, i + 1);
        return null;
    }

    public int count(){
        return array.length - i;
    }

    public int index(){
        return i;
    }

    public ArraySeq_char withMeta(IPersistentMap meta){
        return new ArraySeq_char(meta, array, i);
    }
}

```

```

    }

    public Object reduce(IFn f) throws Exception{
        Object ret = array[i];
        for(int x = i + 1; x < array.length; x++)
            ret = f.invoke(ret, array[x]);
        return ret;
    }

    public Object reduce(IFn f, Object start) throws Exception{
        Object ret = f.invoke(start, array[i]);
        for(int x = i + 1; x < array.length; x++)
            ret = f.invoke(ret, array[x]);
        return ret;
    }

    public int indexOf(Object o) {
        if (o instanceof Character) {
            char c = ((Character) o).charValue();
            for (int j = i; j < array.length; j++)
                if (c == array[j]) return j - i;
        }
        if (o == null) {
            return -1;
        }
        for (int j = i; j < array.length; j++)
            if (o.equals(array[j])) return j - i;
        return -1;
    }

    public int lastIndexOf(Object o) {
        if (o instanceof Character) {
            char c = ((Character) o).charValue();
            for (int j = array.length - 1; j >= i; j--)
                if (c == array[j]) return j - i;
        }
        if (o == null) {
            return -1;
        }
        for (int j = array.length - 1; j >= i; j--)
            if (o.equals(array[j])) return j - i;
        return -1;
    }
}

static public class ArraySeq_boolean
extends ASeq implements IndexedSeq, IReduce{
    public final boolean[] array;
    final int i;
}

```

```
ArraySeq_boolean(IPersistentMap meta, boolean[] array, int i){
    super(meta);
    this.array = array;
    this.i = i;
}

public Object first(){
    return array[i];
}

public ISeq next(){
    if(i + 1 < array.length)
        return new ArraySeq_boolean(meta(), array, i + 1);
    return null;
}

public int count(){
    return array.length - i;
}

public int index(){
    return i;
}

public ArraySeq_boolean withMeta(IPersistentMap meta){
    return new ArraySeq_boolean(meta, array, i);
}

public Object reduce(IFn f) throws Exception{
    Object ret = array[i];
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, array[x]);
    return ret;
}

public Object reduce(IFn f, Object start) throws Exception{
    Object ret = f.invoke(start, array[i]);
    for(int x = i + 1; x < array.length; x++)
        ret = f.invoke(ret, array[x]);
    return ret;
}

public int indexOf(Object o) {
    if (o instanceof Boolean) {
        boolean b = ((Boolean) o).booleanValue();
        for (int j = i; j < array.length; j++)
            if (b == array[j]) return j - i;
    }
    if (o == null) {
        return -1;
    }
}
```

```

        }
        for (int j = i; j < array.length; j++)
            if (o.equals(array[j])) return j - i;
        return -1;
    }

    public int lastIndexOf(Object o) {
        if (o instanceof Boolean) {
            boolean b = ((Boolean) o).booleanValue();
            for (int j = array.length - 1; j >= i; j--)
                if (b == array[j]) return j - i;
        }
        if (o == null) {
            return -1;
        }
        for (int j = array.length - 1; j >= i; j--)
            if (o.equals(array[j])) return j - i;
        return -1;
    }
}

```

9.13 ASeq.java

(Obj [947]) (ISeq [805]) (List [1723]) (Serializable [1723])
— ASeq.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.io.Serializable;
import java.util.*;

public abstract class ASeq
    extends Obj implements ISeq, List, Serializable {
transient int _hash = -1;

public String toString(){
    return RT.printString(this);
}

public IPersistentCollection empty(){
    return PersistentList.EMPTY;
}

```

```

    }

protected ASeq(IPersistentMap meta){
    super(meta);
}

protected ASeq(){
}

public boolean equiv(Object obj){

    if(!(obj instanceof Sequential || obj instanceof List))
        return false;
    ISeq ms = RT.seq(obj);
    for(ISeq s = seq(); s != null; s = s.next(), ms = ms.next())
    {
        if(ms == null || !Util.equiv(s.first(), ms.first()))
            return false;
    }
    return ms == null;
}

public boolean equals(Object obj){
    if(this == obj) return true;
    if(!(obj instanceof Sequential || obj instanceof List))
        return false;
    ISeq ms = RT.seq(obj);
    for(ISeq s = seq(); s != null; s = s.next(), ms = ms.next())
    {
        if(ms == null || !Util.equals(s.first(), ms.first()))
            return false;
    }
    return ms == null;
}

public int hashCode(){
    if(_hash == -1)
    {
        int hash = 1;
        for(ISeq s = seq(); s != null; s = s.next())
        {
            hash = 31 * hash + (s.first() == null
                ? 0 : s.first().hashCode());
        }
        this._hash = hash;
    }
    return _hash;
}

```

```
}

//public Object reduce(IFn f) throws Exception{
//    Object ret = first();
//    for(ISeq s = rest(); s != null; s = s.rest())
//        ret = f.invoke(ret, s.first());
//    return ret;
//}
//
//public Object reduce(IFn f, Object start) throws Exception{
//    Object ret = f.invoke(start, first());
//    for(ISeq s = rest(); s != null; s = s.rest())
//        ret = f.invoke(ret, s.first());
//    return ret;
//}

//public Object peek(){
//    return first();
//}
//
//public IPersistentList pop(){
//    return rest();
//}

public int count(){
    int i = 1;
    for(ISeq s = next(); s != null; s = s.next(), i++)
        if(s instanceof Counted)
            return i + s.count();
    return i;
}

final public ISeq seq(){
    return this;
}

public ISeq cons(Object o){
    return new Cons(o, this);
}

public ISeq more(){
    ISeq s = next();
    if(s == null)
        return PersistentList.EMPTY;
    return s;
}

//final public ISeq rest(){
//    Seqable m = more();
```

```
//      if(m == null)
//          return null;
//      return m.seq();
//}

// java.util.Collection implementation

public Object[] toArray(){
    return RT.seqToArray(seq());
}

public boolean add(Object o){
    throw new UnsupportedOperationException();
}

public boolean remove(Object o){
    throw new UnsupportedOperationException();
}

public boolean addAll(Collection c){
    throw new UnsupportedOperationException();
}

public void clear(){
    throw new UnsupportedOperationException();
}

public boolean retainAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean removeAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean containsAll(Collection c){
    for(Object o : c)
    {
        if(!contains(o))
            return false;
    }
    return true;
}

public Object[] toArray(Object[] a){
    if(a.length >= count())
    {
        ISeq s = seq();
        for(int i = 0; s != null; ++i, s = s.next())
        {
```

```

        a[i] = s.first();
    }
    if(a.length > count())
        a[count()] = null;
    return a;
}
else
    return toArray();
}

public int size(){
    return count();
}

public boolean isEmpty(){
    return seq() == null;
}

public boolean contains(Object o){
    for(ISeq s = seq(); s != null; s = s.next())
    {
        if(Util.equiv(s.first(), o))
            return true;
    }
    return false;
}

public Iterator iterator(){
    return new SeqIterator(this);
}

//////////// List stuff /////////////
private List reify(){
    return Collections.unmodifiableList(new ArrayList(this));
}

public List subList(int fromIndex, int toIndex){
    return reify().subList(fromIndex, toIndex);
}

public Object set(int index, Object element){
    throw new UnsupportedOperationException();
}

public Object remove(int index){
    throw new UnsupportedOperationException();
}

```

```

public int indexOf(Object o){
    ISeq s = seq();
    for(int i = 0; s != null; s = s.next(), i++)
    {
        if(Util.equiv(s.first(), o))
            return i;
    }
    return -1;
}

public int lastIndexOf(Object o){
    return reify().lastIndexOf(o);
}

public ListIterator listIterator(){
    return reify().listIterator();
}

public ListIterator listIterator(int index){
    return reify().listIterator(index);
}

public Object get(int index){
    return RT.nth(this, index);
}

public void add(int index, Object element){
    throw new UnsupportedOperationException();
}

public boolean addAll(int index, Collection c){
    throw new UnsupportedOperationException();
}
}

```

9.14 Associative.java

(IPersistentCollection [800]) (ILookup [797])
 — **Associative.java** —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

```

```
public interface Associative extends IPersistentCollection, ILookup{
    boolean containsKey(Object key);

    IMapEntry entryAt(Object key);

    Associative assoc(Object key, Object val);

}
```

—————

9.15 Atom.java

(ARef [553])

— Atom.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Jan 1, 2009 */

package clojure.lang;

import java.util.concurrent.atomic.AtomicReference;

final public class Atom extends ARef{
final AtomicReference state;

public Atom(Object state){
    this.state = new AtomicReference(state);
}

public Atom(Object state, IPersistentMap meta){
    super(meta);
    this.state = new AtomicReference(state);
}

public Object deref(){
    return state.get();
}

public Object swap(IFn f) throws Exception{
    for(; ;)
    {
        Object v = deref();
        Object newv = f.invoke(v);
        validate(newv);
        if(state.compareAndSet(v, newv))
```

```

        {
        notifyWatches(v, newv);
        return newv;
    }
}

public Object swap(IFn f, Object arg) throws Exception{
    for(; ;)
    {
        Object v = deref();
        Object newv = f.invoke(v, arg);
        validate(newv);
        if(state.compareAndSet(v, newv))
        {
            notifyWatches(v, newv);
            return newv;
        }
    }
}

public Object swap(IFn f, Object arg1, Object arg2) throws Exception{
    for(; ;)
    {
        Object v = deref();
        Object newv = f.invoke(v, arg1, arg2);
        validate(newv);
        if(state.compareAndSet(v, newv))
        {
            notifyWatches(v, newv);
            return newv;
        }
    }
}

public Object swap(IFn f, Object x, Object y, ISeq args)
throws Exception{
    for(; ;)
    {
        Object v = deref();
        Object newv = f.applyTo(RT.listStar(v, x, y, args));
        validate(newv);
        if(state.compareAndSet(v, newv))
        {
            notifyWatches(v, newv);
            return newv;
        }
    }
}

```

```

public boolean compareAndSet(Object oldv, Object newv){
    validate(newv);
    boolean ret = state.compareAndSet(oldv, newv);
    if(ret)
        notifyWatches(oldv, newv);
    return ret;
}

public Object reset(Object newval){
    Object oldval = state.get();
    validate(newval);
    state.set(newval);
    notifyWatches(oldval, newval);
    return newval;
}
}

```

—————

9.16 ATransientMap.java

(AFn [509]) (ITransientMap [808])
 — ATransientMap.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.util.Map;

import clojure.lang.PersistentHashMap.INode;

abstract class ATransientMap extends AFn implements ITransientMap {
    abstract void ensureEditable();
    abstract ITransientMap doAssoc(Object key, Object val);
    abstract ITransientMap doWithout(Object key);
    abstract Object doValAt(Object key, Object notFound);
    abstract int doCount();
    abstract IPersistentMap doPersistent();

    public ITransientMap conj(Object o) {
        ensureEditable();
        if(o instanceof Map.Entry)
        {
            Map.Entry e = (Map.Entry) o;

            return assoc(e.getKey(), e.getValue());
        }
    }
}

```

```

        }
    else if(o instanceof IPersistentVector)
    {
        IPersistentVector v = (IPersistentVector) o;
        if(v.count() != 2)
            throw new IllegalArgumentException(
                "Vector arg to map conj must be a pair");
        return assoc(v.nth(0), v.nth(1));
    }

    ITransientMap ret = this;
    for(ISeq es = RT.seq(o); es != null; es = es.next())
    {
        Map.Entry e = (Map.Entry) es.first();
        ret = ret.assoc(e.getKey(), e.getValue());
    }
    return ret;
}

public final Object invoke(Object arg1) throws Exception{
    return valAt(arg1);
}

public final Object invoke(Object arg1, Object notFound)
throws Exception{
    return valAt(arg1, notFound);
}

public final Object valAt(Object key) {
    return valAt(key, null);
}

public final ITransientMap assoc(Object key, Object val) {
    ensureEditable();
    return doAssoc(key, val);
}

public final ITransientMap without(Object key) {
    ensureEditable();
    return doWithout(key);
}

public final IPersistentMap persistent() {
    ensureEditable();
    return doPersistent();
}

public final Object valAt(Object key, Object notFound) {
    ensureEditable();
    return doValAt(key, notFound);
}

```

```

    }

    public final int count() {
        ensureEditable();
        return doCount();
    }
}

```

—————

9.17 ATransientSet.java

(AFn [509]) (ITransientSet [809])
 — ATransientSet.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 3, 2008 */

package clojure.lang;

public abstract class ATransientSet
    extends AFn implements ITransientSet{
    ITransientMap impl;

    ATransientSet(ITransientMap impl) {
        this.impl = impl;
    }

    public int count() {
        return impl.count();
    }

    public ITransientSet conj(Object val) {
        ITransientMap m = impl.assoc(val, val);
        if (m != impl) this.impl = m;
        return this;
    }

    public boolean contains(Object key) {
        return this != impl.valAt(key, this);
    }

    public ITransientSet disjoin(Object key) throws Exception {
        ITransientMap m = impl.without(key);
        if (m != impl) this.impl = m;
        return this;
    }
}

```

```

    }

    public Object get(Object key) {
        return impl.valAt(key);
    }

    public Object invoke(Object key, Object notFound) throws Exception {
        return impl.valAt(key, notFound);
    }

    public Object invoke(Object key) throws Exception {
        return impl.valAt(key);
    }

}

```

9.18 BigInt.java

(Number [1723])
— BigInt.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* chouser Jun 23, 2010 */

package clojure.lang;

import java.math.BigInteger;

public final class BigInt extends Number{

    final public long lpart;
    final public BigInteger bipart;

    final public static BigInt ZERO = new BigInt(0,null);
    final public static BigInt ONE = new BigInt(1,null);

    //must follow Long
    public int hashCode(){
        if(bipart == null)
            return (int) (this.lpart ^ (this.lpart >>> 32));
        return bipart.hashCode();
    }
}

```

```
public boolean equals(Object obj){
    if(this == obj)
        return true;
    if(obj instanceof BigInt)
    {
        BigInt o = (BigInt) obj;
        if(bipart == null)
            return o.bipart == null && this.lpart == o.lpart;
        return o.bipart != null && this.bipart.equals(o.bipart);
    }
    return false;
}

private BigInt(long lpart, BigInteger bipart){
    this.lpart = lpart;
    this.bipart = bipart;
}

public static BigInt fromBigInteger(BigInteger val){
    if(val.bitLength() < 64)
        return new BigInt(val.longValue(), null);
    else
        return new BigInt(0, val);
}

public static BigInt fromLong(long val){
    return new BigInt(val, null);
}

public BigInteger toBigInteger(){
    if(bipart == null)
        return BigInteger.valueOf(lpart);
    else
        return bipart;
}

////// java.lang.Number:

public int intValue(){
    if(bipart == null)
        return (int) lpart;
    else
        return bipart.intValue();
}

public long longValue(){
    if(bipart == null)
        return lpart;
    else
        return bipart.longValue();
```

```
}

public float floatValue(){
    if(bipart == null)
        return lpart;
    else
        return bipart.floatValue();
}

public double doubleValue(){
    if(bipart == null)
        return lpart;
    else
        return bipart.doubleValue();
}

public byte byteValue(){
    if(bipart == null)
        return (byte) lpart;
    else
        return bipart.byteValue();
}

public short shortValue(){
    if(bipart == null)
        return (short) lpart;
    else
        return bipart.shortValue();
}

public static BigInt valueOf(long val){
    return new BigInt(val, null);
}

public String toString(){
    if(bipart == null)
        return String.valueOf(lpart);
    return bipart.toString();
}

public int bitLength(){
    return toBigInteger().bitLength();
}
```

9.19 Binding.java

— Binding.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

public class Binding<T>{
public T val;
public final Binding rest;

public Binding(T val){
    this.val = val;
    this.rest = null;
}

public Binding(T val, Binding rest){
    this.val = val;
    this.rest = rest;
}
}
```

—————

9.20 Box.java

— Box.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 27, 2006 8:40:19 PM */

package clojure.lang;

public class Box{

public Object val;

public Box(Object val){
    this.val = val;
}
}
```

9.21 ChunkBuffer.java

(Counted [768])
— **ChunkBuffer.java** —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich May 26, 2009 */

package clojure.lang;

final public class ChunkBuffer implements Counted{
    Object[] buffer;
    int end;

    public ChunkBuffer(int capacity){
        buffer = new Object[capacity];
        end = 0;
    }

    public void add(Object o){
        buffer[end++] = o;
    }

    public IChunk chunk(){
        ArrayChunk ret = new ArrayChunk(buffer, 0, end);
        buffer = null;
        return ret;
    }

    public int count(){
        return end;
    }
}
```

9.22 ChunkedCons.java

(ASeq [571]) (IChunkedSeq [773])
— **ChunkedCons.java** —

/*

```
\getchunk{Clojure Copyright}
*/
/* rich May 25, 2009 */

package clojure.lang;

final public class ChunkedCons extends ASeq implements IChunkedSeq{

    final IChunk chunk;
    final ISeq _more;

    ChunkedCons(IPersistentMap meta, IChunk chunk, ISeq more){
        super(meta);
        this.chunk = chunk;
        this._more = more;
    }

    public ChunkedCons(IChunk chunk, ISeq more){
        this(null,chunk, more);
    }

    public Obj withMeta(IPersistentMap meta){
        if(meta != _meta)
            return new ChunkedCons(meta, chunk, _more);
        return this;
    }

    public Object first(){
        return chunk.nth(0);
    }

    public ISeq next(){
        if(chunk.count() > 1)
            return new ChunkedCons(chunk.dropFirst(), _more);
        return chunkedNext();
    }

    public ISeq more(){
        if(chunk.count() > 1)
            return new ChunkedCons(chunk.dropFirst(), _more);
        if(_more == null)
            return PersistentList.EMPTY;
        return _more;
    }

    public IChunk chunkedFirst(){
        return chunk;
    }

    public ISeq chunkedNext(){
```

```

        return chunkedMore().seq();
    }

    public ISeq chunkedMore(){
        if(_more == null)
            return PersistentList.EMPTY;
        return _more;
    }
}

```

9.23 Compile.java

— Compile.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.IOException;

// Compiles libs and generates class files stored within the directory
// named by the Java System property "clojure.compile.path". Arguments
// are strings naming the libs to be compiled. The libs and compile-path
// must all be within CLASSPATH.

public class Compile{

    private static final String PATH_PROP =
        "clojure.compile.path";
    private static final String REFLECTION_WARNING_PROP =
        "clojure.compile.warn-on-reflection";
    private static final Var compile_path =
        RT.var("clojure.core", "*compile-path*");
    private static final Var compile =
        RT.var("clojure.core", "compile");
    private static final Var warn_on_reflection =
        RT.var("clojure.core", "*warn-on-reflection*");

    public static void main(String[] args) throws Exception{
        OutputStreamWriter out = (OutputStreamWriter) RT.OUT.deref();
        PrintWriter err = RT.errPrintWriter();

```

```
String path = System.getProperty(PATH_PROP);
int count = args.length;

if(path == null)
{
    err.println("ERROR: Must set system property " + PATH_PROP +
        "\nto the location for compiled .class files." +
        "\nThis directory must also be on your CLASSPATH.");
    System.exit(1);
}

boolean warnOnReflection =
    System.getProperty(REFLECTION_WARNING_PROP, "false").equals("true");

try
{
    Var.pushThreadBindings(
        RT.map(compile_path,
            path,
            warn_on_reflection,
            warnOnReflection));

    for(String lib : args)
    {
        out.write("Compiling " + lib + " to " + path + "\n");
        out.flush();
        compile.invoke(Symbol.intern(lib));
    }
}
finally
{
    Var.popThreadBindings();
    try
    {
        out.flush();
        out.close();
    }
    catch(IOException e)
    {
        e.printStackTrace(err);
    }
}
}
```

9.24 Compiler.java

(Opcodes [387])
 — Compiler.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Aug 21, 2007 */

package clojure.lang;

/**


import clojure.asm.*;
import clojure.asm.commons.Method;
import clojure.asm.commons.GeneratorAdapter;
/**/
/*

import org.objectweb.asm.*;
import org.objectweb.asm.commons.Method;
import org.objectweb.asm.commons.GeneratorAdapter;
import org.objectweb.asm.util.TraceClassVisitor;
import org.objectweb.asm.util.CheckClassAdapter;
/**/


import java.io.*;
import java.util.*;
import java.lang.reflect.Constructor;
import java.lang.reflect.Modifier;

public class Compiler implements Opcodes{

    static final Symbol DEF = Symbol.intern("def");
    static final Symbol LOOP = Symbol.intern("loop*");
    static final Symbol RECUR = Symbol.intern("recur");
    static final Symbol IF = Symbol.intern("if");
    static final Symbol LET = Symbol.intern("let*");
    static final Symbol LETFN = Symbol.intern("letfn*");
    static final Symbol DO = Symbol.intern("do");
    static final Symbol FN = Symbol.intern("fn*");
    static final Symbol QUOTE = Symbol.intern("quote");
    static final Symbol THE_VAR = Symbol.intern("var");
    static final Symbol DOT = Symbol.intern(".");
    static final Symbol ASSIGN = Symbol.intern("set!");
    //static final Symbol TRY_FINALLY = Symbol.intern("try-finally");
    static final Symbol TRY = Symbol.intern("try");
    static final Symbol CATCH = Symbol.intern("catch");
```

```

static final Symbol FINALLY = Symbol.intern("finally");
static final Symbol THROW = Symbol.intern("throw");
static final Symbol MONITOR_ENTER = Symbol.intern("monitor-enter");
static final Symbol MONITOR_EXIT = Symbol.intern("monitor-exit");
static final Symbol IMPORT = Symbol.intern("clojure.core", "import*");
//static final Symbol INSTANCE = Symbol.intern("instance?");
static final Symbol DEFTYPE = Symbol.intern("deftype*");
static final Symbol CASE = Symbol.intern("case*");

//static final Symbol THISFN = Symbol.intern("thisfn");
static final Symbol CLASS = Symbol.intern("Class");
static final Symbol NEW = Symbol.intern("new");
static final Symbol THIS = Symbol.intern("this");
static final Symbol REIFY = Symbol.intern("reify*");
//static final Symbol UNQUOTE = Symbol.intern("unquote");
//static final Symbol UNQUOTE_SPLICING =
//    Symbol.intern("unquote-splicing");
//static final Symbol SYNTAX_QUOTE =
//    Symbol.intern("clojure.core", "syntax-quote");
static final Symbol LIST = Symbol.intern("clojure.core", "list");
static final Symbol HASHMAP = Symbol.intern("clojure.core", "hash-map");
static final Symbol VECTOR = Symbol.intern("clojure.core", "vector");
static final Symbol IDENTITY = Symbol.intern("clojure.core", "identity");

static final Symbol _AMP_ = Symbol.intern("&");
static final Symbol ISEQ = Symbol.intern("clojure.lang.ISeq");

static final Keyword inlineKey = Keyword.intern(null, "inline");
static final Keyword inlineAritiesKey =
    Keyword.intern(null, "inline-arities");
static final Keyword staticKey = Keyword.intern(null, "static");
static final Keyword arglistsKey = Keyword.intern(null, "arglists");
static final Symbol INVOKE_STATIC = Symbol.intern("invokeStatic");

static final Keyword volatileKey = Keyword.intern(null, "volatile");
static final Keyword implementsKey = Keyword.intern(null, "implements");
static final String COMPILE_STUB_PREFIX = "compile__stub";

static final Keyword protocolKey = Keyword.intern(null, "protocol");
static final Keyword onKey = Keyword.intern(null, "on");
static final Keyword dynamicKey = Keyword.intern("dynamic");

static final Symbol NS = Symbol.intern("ns");
static final Symbol IN_NS = Symbol.intern("in-ns");

//static final Symbol IMPORT = Symbol.intern("import");
//static final Symbol USE = Symbol.intern("use");

//static final Symbol IFN = Symbol.intern("clojure.lang", "IFn");

```

```

static final public IPersistentMap specials = PersistentHashMap.create(
    DEF, new DefExpr.Parser(),
    LOOP, new LetExpr.Parser(),
    RECUR, new RecurExpr.Parser(),
    IF, new IfExpr.Parser(),
    CASE, new CaseExpr.Parser(),
    LET, new LetExpr.Parser(),
    LETFN, new LetFnExpr.Parser(),
    DO, new BodyExpr.Parser(),
    FN, null,
    QUOTE, new ConstantExpr.Parser(),
    THE_VAR, new TheVarExpr.Parser(),
    IMPORT, new ImportExpr.Parser(),
    DOT, new HostExpr.Parser(),
    ASSIGN, new AssignExpr.Parser(),
    DEFTYPE, new NewInstanceExpr.DeftypeParser(),
    REIFY, new NewInstanceExpr.ReifyParser(),
    // TRY_FINALLY, new TryFinallyExpr.Parser(),
    TRY, new TryExpr.Parser(),
    THROW, new ThrowExpr.Parser(),
    MONITOR_ENTER, new MonitorEnterExpr.Parser(),
    MONITOR_EXIT, new MonitorExitExpr.Parser(),
    // INSTANCE, new InstanceExpr.Parser(),
    // IDENTICAL, new IdenticalExpr.Parser(),
    //THISFN, null,
    CATCH, null,
    FINALLY, null,
    // CLASS, new ClassExpr.Parser(),
    NEW, new NewExpr.Parser(),
    // UNQUOTE, null,
    // UNQUOTE_SPLICING, null,
    // SYNTAX_QUOTE, null,
    _AMP_, null
);

private static final int MAX_POSITIONAL_ARITY = 20;
private static final Type OBJECT_TYPE;
private static final Type KEYWORD_TYPE = Type.getType(Keyword.class);
private static final Type VAR_TYPE = Type.getType(Var.class);
private static final Type SYMBOL_TYPE = Type.getType(Symbol.class);
//private static final Type NUM_TYPE = Type.getType(Num.class);
private static final Type IFN_TYPE = Type.getType(IFn.class);
private static final Type AFUNCTION_TYPE = Type.getType(AFunction.class);
private static final Type RT_TYPE = Type.getType(RT.class);
private static final Type NUMBERS_TYPE = Type.getType(Numbers.class);
final static Type CLASS_TYPE = Type.getType(Class.class);
final static Type NS_TYPE = Type.getType(Namespace.class);
final static Type UTIL_TYPE = Type.getType(Util.class);
final static Type REFLECTOR_TYPE = Type.getType(Reflector.class);
final static Type THROWABLE_TYPE = Type.getType(Throwable.class);

```

```

final static Type BOOLEAN_OBJECT_TYPE = Type.getType(Boolean.class);
final static Type IPERSISTENTMAP_TYPE =
    Type.getType(IPersistentMap.class);
final static Type IOBJ_TYPE = Type.getType(IObj.class);

private static final Type[] [] ARG_TYPES;
private static final Type[] EXCEPTION_TYPES =
    {Type.getType(Exception.class)};

static
{
    OBJECT_TYPE = Type.getType(Object.class);
    ARG_TYPES = new Type[MAX_POSITIONAL_ARITY + 2] [];
    for(int i = 0; i <= MAX_POSITIONAL_ARITY; ++i)
    {
        Type[] a = new Type[i];
        for(int j = 0; j < i; j++)
            a[j] = OBJECT_TYPE;
        ARG_TYPES[i] = a;
    }
    Type[] a = new Type[MAX_POSITIONAL_ARITY + 1];
    for(int j = 0; j < MAX_POSITIONAL_ARITY; j++)
        a[j] = OBJECT_TYPE;
    a[MAX_POSITIONAL_ARITY] = Type.getType("[Ljava/lang/Object;");
    ARG_TYPES[MAX_POSITIONAL_ARITY + 1] = a;

}

//symbol->localbinding
static final public Var LOCAL_ENV = Var.create(null).setDynamic();

//vector<localbinding>
static final public Var LOOP_LOCALS = Var.create().setDynamic();

//Label
static final public Var LOOP_LABEL = Var.create().setDynamic();

//vector<object>
static final public Var CONSTANTS = Var.create().setDynamic();

//IdentityHashMap
static final public Var CONSTANT_IDS = Var.create().setDynamic();

//vector<keyword>
static final public Var KEYWORD_CALLSITES = Var.create().setDynamic();

//vector<var>
static final public Var PROTOCOL_CALLSITES = Var.create().setDynamic();

```

```
//set<var>
static final public Var VAR_CALLSITES = Var.create().setDynamic();

//keyword->constid
static final public Var KEYWORDS = Var.create().setDynamic();

//var->constid
static final public Var VARS = Var.create().setDynamic();

//FnFrame
static final public Var METHOD = Var.create(null).setDynamic();

//null or not
static final public Var IN_CATCH_FINALLY = Var.create(null).setDynamic();

static final public Var NO_RECUR = Var.create(null).setDynamic();

//DynamicClassLoader
static final public Var LOADER = Var.create().setDynamic();

//String
static final public Var SOURCE =
    Var.intern(Namespace.findOrCreate(Symbol.intern("clojure.core")),
               Symbol.intern("*source-path*"), "NO_SOURCE_FILE").setDynamic();

//String
static final public Var SOURCE_PATH =
    Var.intern(Namespace.findOrCreate(Symbol.intern("clojure.core")),
               Symbol.intern("*file*"), "NO_SOURCE_PATH").setDynamic();

//String
static final public Var COMPILE_PATH =
    Var.intern(Namespace.findOrCreate(Symbol.intern("clojure.core")),
               Symbol.intern("*compile-path*"), null).setDynamic();
//boolean
static final public Var COMPILE_FILES =
    Var.intern(Namespace.findOrCreate(Symbol.intern("clojure.core")),
               Symbol.intern("*compile-files*"), Boolean.FALSE).setDynamic();

static final public Var INSTANCE =
    Var.intern(Namespace.findOrCreate(Symbol.intern("clojure.core")),
               Symbol.intern("instance?"));

static final public Var ADD_ANNOTATIONS =
    Var.intern(Namespace.findOrCreate(Symbol.intern("clojure.core")),
               Symbol.intern("add-annotations"));

//boolean
static final public Var UNCHECKED_MATH =
```

```
Var.intern(Namespace.findOrCreate(Symbol.intern("clojure.core")),
           Symbol.intern("*unchecked-math*"), Boolean.FALSE).setDynamic();

//Integer
static final public Var LINE = Var.create(0).setDynamic();

//Integer
static final public Var LINE_BEFORE = Var.create(0).setDynamic();
static final public Var LINE_AFTER = Var.create(0).setDynamic();

//Integer
static final public Var NEXT_LOCAL_NUM = Var.create(0).setDynamic();

//Integer
static final public Var RET_LOCAL_NUM = Var.create().setDynamic();

static final public Var COMPILE_STUB_SYM =
    Var.create(null).setDynamic();
static final public Var COMPILE_STUB_CLASS =
    Var.create(null).setDynamic();

//PathNode chain
static final public Var CLEAR_PATH = Var.create(null).setDynamic();

//tail of PathNode chain
static final public Var CLEAR_ROOT = Var.create(null).setDynamic();

//LocalBinding -> Set<LocalBindingExpr>
static final public Var CLEAR_SITES = Var.create(null).setDynamic();

public enum C{
    STATEMENT, //value ignored
    EXPRESSION, //value required
    RETURN,      //tail position relative to enclosing recur frame
    EVAL
}

interface Expr{
    Object eval() throws Exception;

    void emit(C context, ObjExpr objx, GeneratorAdapter gen);

    boolean hasJavaClass() throws Exception;

    Class getJavaClass() throws Exception;
}

public static abstract class UntypedExpr implements Expr{
```

```

public Class getJavaClass(){
    throw new IllegalArgumentException("Has no Java class");
}

public boolean hasJavaClass(){
    return false;
}
}

interface IParser{
    Expr parse(C context, Object form) throws Exception;
}

static boolean isSpecial(Object sym){
    return specials.containsKey(sym);
}

static Symbol resolveSymbol(Symbol sym){
    //already qualified or classname?
    if(sym.name.indexOf('.') > 0)
        return sym;
    if(sym.ns != null)
    {
        Namespace ns = namespaceFor(sym);
        if(ns == null || ns.name.name == sym.ns)
            return sym;
        return Symbol.intern(ns.name.name, sym.name);
    }
    Object o = currentNS().getMapping(sym);
    if(o == null)
        return Symbol.intern(currentNS().name.name, sym.name);
    else if(o instanceof Class)
        return Symbol.intern(null, ((Class) o).getName());
    else if(o instanceof Var)
    {
        Var v = (Var) o;
        return Symbol.intern(v.ns.name.name, v.sym.name);
    }
    return null;
}

static class DefExpr implements Expr{
    public final Var var;
    public final Expr init;
    public final Expr meta;
    public final boolean initProvided;
    public final boolean isDynamic;
    public final String source;
}

```

```

public final int line;
final static Method bindRootMethod =
    Method.getMethod("void bindRoot(Object)");
final static Method setTagMethod =
    Method.getMethod("void setTag(clojure.lang.Symbol)");
final static Method setMetaMethod =
    Method.getMethod("void setMeta(clojure.lang.IPersistentMap)");
final static Method setDynamicMethod =
    Method.getMethod("clojure.lang.Var setDynamic(boolean)");
final static Method symintern =
    Method.getMethod("clojure.lang.Symbol intern(String, String)");

public DefExpr(String source, int line, Var var, Expr init,
               Expr meta, boolean initProvided, boolean isDynamic){
    this.source = source;
    this.line = line;
    this.var = var;
    this.init = init;
    this.meta = meta;
    this.isDynamic = isDynamic;
    this.initProvided = initProvided;
}

private boolean includesExplicitMetadata(MapExpr expr) {
    for(int i=0; i < expr.keyvals.count(); i += 2)
    {
        Keyword k = ((KeywordExpr) expr.keyvals.nth(i)).k;
        if ((k != RT.FILE_KEY) &&
            (k != RT.DECLARED_KEY) &&
            (k != RT.LINE_KEY))
            return true;
    }
    return false;
}

public Object eval() throws Exception{
    try
    {
        if(initProvided)
        {
//            if(init instanceof FnExpr &&
//                ((FnExpr) init).closes.count()==0)
//                var.bindRoot(new FnLoaderThunk((FnExpr) init, var));
//            else
//                var.bindRoot(init.eval());
//            }
        if(meta != null)
        {
            IPersistentMap metaMap = (IPersistentMap) meta.eval();
//            includesExplicitMetadata((MapExpr) meta)
        }
    }
}

```

```

        if (initProvided || true)
            var.setMeta((IPersistentMap) meta.eval());
    }
    return var.setDynamic(isDynamic);
}
catch(Throwable e)
{
    if(!(e instanceof CompilerException))
        throw new CompilerException(source, line, e);
    else
        throw (CompilerException) e;
}
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    objx.emitVar(gen, var);
    if(isDynamic)
    {
        gen.push(isDynamic);
        gen.invokeVirtual(VAR_TYPE, setDynamicMethod);
    }
    if(meta != null)
    {
        //includesExplicitMetadata((MapExpr) meta))
        if (initProvided || true)
        {
            gen.dup();
            meta.emit(C.EXPRESSION, objx, gen);
            gen.checkCast(IPERSISTENTMAP_TYPE);
            gen.invokeVirtual(VAR_TYPE, setMetaMethod);
        }
    }
    if(initProvided)
    {
        gen.dup();
        init.emit(C.EXPRESSION, objx, gen);
        gen.invokeVirtual(VAR_TYPE, bindRootMethod);
    }

    if(context == C.STATEMENT)
        gen.pop();
}

public boolean hasJavaClass(){
    return true;
}

public Class getJavaClass(){
    return Var.class;
}

```

```

static class Parser implements IParser{
    public Expr parse(C context, Object form) throws Exception{
        //((def x) or (def x initexpr) or (def x "docstring" initexpr)
        String docstring = null;
        if(RT.count(form) == 4 &&
           (RT.third(form) instanceof String)) {
            docstring = (String) RT.third(form);
            form = RT.list(RT.first(form),
                           RT.second(form),
                           RT.fourth(form));
        }
        if(RT.count(form) > 3)
            throw new Exception("Too many arguments to def");
        else if(RT.count(form) < 2)
            throw new Exception("Too few arguments to def");
        else if(!(RT.second(form) instanceof Symbol))
            throw new Exception(
                "First argument to def must be a Symbol");
        Symbol sym = (Symbol) RT.second(form);
        Var v = lookupVar(sym, true);
        if(v == null)
            throw new Exception(
                "Can't refer to qualified var that doesn't exist");
        if(!v.ns.equals(currentNS()))
        {
            if(sym.ns == null)
                v = currentNS().intern(sym);
            // throw new Exception(
            //     "Name conflict, can't def " + sym +
            //     " because namespace: " + currentNS().name +
            //     " refers to:" + v);
            else
                throw new Exception(
                    "Can't create defs outside of current ns");
        }
        IPersistentMap mm = sym.meta();
        boolean isDynamic = RT.booleanCast(RT.get(mm,dynamicKey));
        if(!isDynamic &&
           sym.name.startsWith("*") &&
           sym.name.endsWith("*") &&
           sym.name.length() > 1)
        {
            RT.errPrintWriter().format(
                "Var %s not marked :dynamic true, setting to "+
                ":dynamic. You should fix this before next release!\n",
                sym);
            isDynamic = true;
            mm = (IPersistentMap) RT.assoc(mm,dynamicKey, RT.T);
        }
    }
}

```

```

        if(RT.booleanCast(RT.get(mm, arglistsKey)))
        {
            IPersistentMap vm = v.meta();
            //vm = (IPersistentMap) RT.assoc(vm,staticKey,RT.T);
            //drop quote
            vm = (IPersistentMap)
                RT.assoc(vm,arglistsKey,
                    RT.second(mm.valAt(arglistsKey)));
            v.setMeta(vm);
        }
        Object source_path = SOURCE_PATH.get();
        source_path = source_path ==
            null ? "NO_SOURCE_FILE" : source_path;
        mm = (IPersistentMap) RT.assoc(mm, RT.LINE_KEY, LINE.get())
            .assoc(RT.FILE_KEY, source_path);
        if (docstring != null)
            mm = (IPersistentMap) RT.assoc(mm, RT.DOC_KEY, docstring);
        Expr meta =
            analyze(context == C.EVAL ? context : C.EXPRESSION, mm);
        return new DefExpr((String) SOURCE.deref(),
            (Integer) LINE.deref(),
            v, analyze(
                context == C.EVAL ?
                    context : C.EXPRESSION,
                    RT.third(form),
                    v.sym.name),
            meta, RT.count(form) == 3, isDynamic);
    }
}
}

public static class AssignExpr implements Expr{
    public final AssignableExpr target;
    public final Expr val;

    public AssignExpr(AssignableExpr target, Expr val){
        this.target = target;
        this.val = val;
    }

    public Object eval() throws Exception{
        return target.evalAssign(val);
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        target.emitAssign(context, objx, gen, val);
    }

    public boolean hasJavaClass() throws Exception{
        return val.hasJavaClass();
    }
}

```

```

    }

    public Class getJavaClass() throws Exception{
        return val.getJavaClass();
    }

    static class Parser implements IParser{
        public Expr parse(C context, Object frm) throws Exception{
            ISeq form = (ISeq) frm;
            if(RT.length(form) != 3)
                throw new IllegalArgumentException(
                    "Malformed assignment, expecting (set! target val)");
            Expr target = analyze(C.EXPRESSION, RT.second(form));
            if(!(target instanceof AssignableExpr))
                throw new IllegalArgumentException(
                    "Invalid assignment target");
            return new AssignExpr((AssignableExpr) target,
                analyze(C.EXPRESSION,
                    RT.third(form)));
        }
    }
}

public static class VarExpr implements Expr, AssignableExpr{
    public final Var var;
    public final Object tag;
    final static Method getMethod =
        Method.getMethod("Object get()");
    final static Method setMethod =
        Method.getMethod("Object set(Object)");

    public VarExpr(Var var, Symbol tag){
        this.var = var;
        this.tag = tag != null ? tag : var.getTag();
    }

    public Object eval() throws Exception{
        return var.deref();
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        objx.emitVarValue(gen, var);
        if(context == C.STATEMENT)
        {
            gen.pop();
        }
    }

    public boolean hasJavaClass(){
        return tag != null;
    }
}

```

```

    }

    public Class getJavaClass() throws Exception{
        return HostExpr.tagToClass(tag);
    }

    public Object evalAssign(Expr val) throws Exception{
        return var.set(val.eval());
    }

    public void emitAssign(C context, ObjExpr objx, GeneratorAdapter gen,
                          Expr val){
        objx.emitVar(gen, var);
        val.emit(C.EXPRESSION, objx, gen);
        gen.invokeVirtual(VAR_TYPE, setMethod);
        if(context == C.STATEMENT)
            gen.pop();
    }
}

public static class TheVarExpr implements Expr{
    public final Var var;

    public TheVarExpr(Var var){
        this.var = var;
    }

    public Object eval() throws Exception{
        return var;
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        objx.emitVar(gen, var);
        if(context == C.STATEMENT)
            gen.pop();
    }

    public boolean hasJavaClass(){
        return true;
    }

    public Class getJavaClass() throws ClassNotFoundException{
        return Var.class;
    }

    static class Parser implements IParser{
        public Expr parse(C context, Object form) throws Exception{
            Symbol sym = (Symbol) RT.second(form);
            Var v = lookupVar(sym, false);
            if(v != null)

```

```

        return new TheVarExpr(v);
        throw new Exception("Unable to resolve var: " + sym +
            " in this context");
    }
}
}

public static class KeywordExpr extends LiteralExpr{
    public final Keyword k;

    public KeywordExpr(Keyword k){
        this.k = k;
    }

    Object val(){
        return k;
    }

    public Object eval() {
        return k;
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        objx.emitKeyword(gen, k);
        if(context == C.STATEMENT)
            gen.pop();
    }

    public boolean hasJavaClass(){
        return true;
    }

    public Class getJavaClass() throws ClassNotFoundException{
        return Keyword.class;
    }
}

public static class ImportExpr implements Expr{
    public final String c;
    final static Method forNameMethod =
        Method.getMethod("Class forName(String)");
    final static Method importClassMethod =
        Method.getMethod("Class importClass(Class)");
    final static Method derefMethod =
        Method.getMethod("Object deref()");

    public ImportExpr(String c){
        this.c = c;
    }
}

```

```

public Object eval() throws Exception{
    Namespace ns = (Namespace) RT.CURRENT_NS.deref();
    ns.importClass(RT.classForName(c));
    return null;
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    gen.getStatic(RT_TYPE,"CURRENT_NS",VAR_TYPE);
    gen.invokeVirtual(VAR_TYPE, derefMethod);
    gen.checkCast(NS_TYPE);
    gen.push(c);
    gen.invokeStatic(CLASS_TYPE, forNameMethod);
    gen.invokeVirtual(NS_TYPE, importClassMethod);
    if(context == C.STATEMENT)
        gen.pop();
}

public boolean hasJavaClass(){
    return false;
}

public Class getJavaClass() throws ClassNotFoundException{
    throw
        new IllegalArgumentException("ImportExpr has no Java class");
}

static class Parser implements IParser{
    public Expr parse(C context, Object form) throws Exception{
        return new ImportExpr((String) RT.second(form));
    }
}
}

public static abstract class LiteralExpr implements Expr{
    abstract Object val();

    public Object eval(){
        return val();
    }
}

static interface AssignableExpr{
    Object evalAssign(Expr val) throws Exception;

    void emitAssign(C context,
                    ObjExpr objx,
                    GeneratorAdapter gen,
                    Expr val);
}

```

```
static public interface MaybePrimitiveExpr extends Expr{
    public boolean canEmitPrimitive();
    public void emitUnboxed(C context,
                           ObjExpr objx,
                           GeneratorAdapter gen);
}

static public abstract class HostExpr
implements Expr, MaybePrimitiveExpr{
    final static Type BOOLEAN_TYPE = Type.getType(Boolean.class);
    final static Type CHAR_TYPE = Type.getType(Character.class);
    final static Type INTEGER_TYPE = Type.getType(Integer.class);
    final static Type LONG_TYPE = Type.getType(Long.class);
    final static Type FLOAT_TYPE = Type.getType(Float.class);
    final static Type DOUBLE_TYPE = Type.getType(Double.class);
    final static Type SHORT_TYPE = Type.getType(Short.class);
    final static Type BYTE_TYPE = Type.getType(Byte.class);
    final static Type NUMBER_TYPE = Type.getType(Number.class);

    final static Method charValueMethod =
        Method.getMethod("char charValue()");
    final static Method booleanValueMethod =
        Method.getMethod("boolean booleanValue()");

    final static Method charValueOfMethod =
        Method.getMethod("Character valueOf(char)");
    final static Method intValueOfMethod =
        Method.getMethod("Integer valueOf(int)");
    final static Method longValueOfMethod =
        Method.getMethod("Long valueOf(long)");
    final static Method floatValueOfMethod =
        Method.getMethod("Float valueOf(float)");
    final static Method doubleValueOfMethod =
        Method.getMethod("Double valueOf(double)");
    final static Method shortValueOfMethod =
        Method.getMethod("Short valueOf(short)");
    final static Method byteValueOfMethod =
        Method.getMethod("Byte valueOf(byte)");

    final static Method intValueMethod =
        Method.getMethod("int intValue()");
    final static Method longValueMethod =
        Method.getMethod("long longValue()");
    final static Method floatValueMethod =
        Method.getMethod("float floatValue()");
    final static Method doubleValueMethod =
        Method.getMethod("double doubleValue()");
    final static Method byteValueMethod =
        Method.getMethod("byte byteValue()");
}
```

```

final static Method shortValueMethod =
    Method.getMethod("short shortValue()");

final static Method fromIntMethod =
    Method.getMethod("clojure.lang.Num from(int)");
final static Method fromLongMethod =
    Method.getMethod("clojure.lang.Num from(long)");
final static Method fromDoubleMethod =
    Method.getMethod("clojure.lang.Num from(double)");

/*
public static void emitBoxReturn(ObjExpr objx,
                                 GeneratorAdapter gen,
                                 Class returnType){
    if(returnType.isPrimitive())
    {
        if(returnType == boolean.class)
        {
            Label falseLabel = gen.newLabel();
            Label endLabel = gen.newLabel();
            gen.ifZCmp(GeneratorAdapter.EQ, falseLabel);
            gen.getStatic(BOOLEAN_OBJECT_TYPE, "TRUE",
                         BOOLEAN_OBJECT_TYPE);
            gen.goTo(endLabel);
            gen.mark(falseLabel);
            gen.getStatic(BOOLEAN_OBJECT_TYPE, "FALSE",
                         BOOLEAN_OBJECT_TYPE);
        }
        // NIL_EXPR.emit(C.EXPRESSION, fn, gen);
        gen.mark(endLabel);
    }
    else if(returnType == void.class)
    {
        NIL_EXPR.emit(C.EXPRESSION, objx, gen);
    }
    else if(returnType == char.class)
    {
        gen.invokeStatic(CHAR_TYPE, charValueOfMethod);
    }
    else
    {
        if(returnType == int.class)
        {
            gen.visitInsn(I2L);
            gen.invokeStatic(NUMBERS_TYPE,
                            Method.getMethod("Number num(long)"));
        }
        else if(returnType == float.class)
        {
            gen.visitInsn(F2D);
        }
    }
}

```

```

                gen.invokeStatic(DOUBLE_TYPE,
                                  doubleValueOfMethod);
            }
        else if(returnType == double.class)
            gen.invokeStatic(DOUBLE_TYPE,
                            doubleValueOfMethod);
        else if(returnType == long.class)
            gen.invokeStatic(NUMBERS_TYPE,
                            Method.getMethod("Number num(long)"));
        else if(returnType == byte.class)
            gen.invokeStatic(BYTE_TYPE,
                            byteValueOfMethod);
        else if(returnType == short.class)
            gen.invokeStatic(SHORT_TYPE,
                            shortValueOfMethod);
    }
}

<*/
public static void emitUnboxArg(ObjExpr objx,
                                GeneratorAdapter gen,
                                Class paramType){
if(paramType.isPrimitive())
{
    if(paramType == boolean.class)
    {
        gen.checkCast(BOOLEAN_TYPE);
        gen.invokeVirtual(BOOLEAN_TYPE, booleanValueMethod);
        // Label falseLabel = gen.newLabel();
        // Label endLabel = gen.newLabel();
        // gen.ifNull(falseLabel);
        // gen.push(1);
        // gen.goTo(endLabel);
        // gen.mark(falseLabel);
        // gen.push(0);
        // gen.mark(endLabel);
    }
    else if(paramType == char.class)
    {
        gen.checkCast(CHAR_TYPE);
        gen.invokeVirtual(CHAR_TYPE, charValueMethod);
    }
    else
    {
        Method m = null;
        gen.checkCast(NUMBER_TYPE);
        if(RT.booleanCast(UNCHECKED_MATH.deref()))
        {
            if(paramType == int.class)

```

```

        m = Method.getMethod(
            "int uncheckedIntCast(Object)");
        else if(paramType == float.class)
            m = Method.getMethod(
                "float uncheckedFloatCast(Object)");
        else if(paramType == double.class)
            m = Method.getMethod(
                "double uncheckedDoubleCast(Object)");
        else if(paramType == long.class)
            m = Method.getMethod(
                "long uncheckedLongCast(Object)");
        else if(paramType == byte.class)
            m = Method.getMethod(
                "byte uncheckedByteCast(Object)");
        else if(paramType == short.class)
            m = Method.getMethod(
                "short uncheckedShortCast(Object"));
    }
    else
    {
        if(paramType == int.class)
            m = Method.getMethod("int intCast(Object)");
        else if(paramType == float.class)
            m = Method.getMethod("float floatCast(Object)");
        else if(paramType == double.class)
            m = Method.getMethod("double doubleCast(Object)");
        else if(paramType == long.class)
            m = Method.getMethod("long longCast(Object)");
        else if(paramType == byte.class)
            m = Method.getMethod("byte byteCast(Object)");
        else if(paramType == short.class)
            m = Method.getMethod("short shortCast(Object"));
    }
    gen.invokeStatic(RT_TYPE, m);
}
}
else
{
    gen.checkCast(Type.getType(paramType));
}
}

static class Parser implements IParser{
    public Expr parse(C context, Object frm) throws Exception{
        ISeq form = (ISeq) frm;
        //(. x fieldname-sym) or
        //(. x 0-ary-method)
        // (. x methodname-sym args+)
        // (. x (methodname-sym args?))
        if(RT.length(form) < 3)

```

```

        throw new IllegalArgumentException(
    "Malformed member expression, expecting (. target member ...)");
    //determine static or instance
    //static target must be symbol, either
    // fully.qualified.Classname or Classname that has
    // been imported
    int line = (Integer) LINE.deref();
    String source = (String) SOURCE.deref();
    Class c = maybeClass(RT.second(form), false);
    //at this point c will be non-null if static
    Expr instance = null;
    if(c == null)
        instance =
            analyze(context == C.EVAL ? context
                : C.EXPRESSION, RT.second(form));
    boolean maybeField = RT.length(form) == 3 &&
        (RT.third(form) instanceof Symbol ||
         RT.third(form) instanceof Keyword);
    if(maybeField && !(RT.third(form) instanceof Keyword))
    {
        Symbol sym = (Symbol) RT.third(form);
        if(c != null)
            maybeField =
                Reflector.getMethods(c, 0, munge(sym.name), true)
                    .size() == 0;
        else if(instance != null &&
            instance.hasJavaClass() &&
            instance.getJavaClass() != null)
            maybeField =
                Reflector.getMethods(instance.getJavaClass(), 0,
                    munge(sym.name), false).size() == 0;
    }
    if(maybeField) //field
    {
        Symbol sym = (RT.third(form) instanceof Keyword)?
            ((Keyword)RT.third(form)).sym
            :(Symbol) RT.third(form);
        Symbol tag = tagOf(form);
        if(c != null) {
            return
                new StaticFieldExpr(line, c, munge(sym.name), tag);
        } else
            return
                new InstanceFieldExpr(line, instance,
                    munge(sym.name), tag);
    }
    else
    {
        ISeq call = (ISeq)
            ((RT.third(form) instanceof ISeq)

```

```

        ? RT.third(form)
        : RT.next(RT.next(form)));
if(!(RT.first(call) instanceof Symbol))
    throw new IllegalArgumentException(
        "Malformed member expression");
Symbol sym = (Symbol) RT.first(call);
Symbol tag = tagOf(form);
PersistentVector args = PersistentVector.EMPTY;
for(ISeq s = RT.next(call); s != null; s = s.next())
    args =
        args.cons(analyze(context == C.EVAL
            ? context
            : C.EXPRESSION, s.first()));
if(c != null)
    return
        new StaticMethodExpr(source, line, tag, c,
            munge(sym.name), args);
else
    return
        new InstanceMethodExpr(source, line, tag,
            instance,
            munge(sym.name), args);
}
}
}

private static Class maybeClass(Object form, boolean stringOk)
throws Exception{
    if(form instanceof Class)
        return (Class) form;
    Class c = null;
    if(form instanceof Symbol)
    {
        Symbol sym = (Symbol) form;
        if(sym.ns == null) //if ns-qualified can't be classname
        {
            if(Util.equals(sym, COMPILE_STUB_SYM.get()))
                return (Class) COMPILE_STUB_CLASS.get();
            if(sym.name.indexOf('.') > 0 ||
                sym.name.charAt(0) == '[')
                c = RT.className(sym.name);
            else
            {
                Object o = currentNS().getMapping(sym);
                if(o instanceof Class)
                    c = (Class) o;
                else
                {
                    try{
                        c = RT.className(sym.name);
                    }
                    catch(Throwable t)
                    {
                        if(stringOk)
                            c = RT.classForName(sym.name);
                        else
                            throw t;
                    }
                }
            }
        }
    }
}
```

```

        }
        catch(Exception e){
            //aargh
        }
    }
}

else if(stringOk && form instanceof String)
    c = RT.className((String) form);
return c;
}

/*
private static String maybeClassName(Object form, boolean stringOk){
    String className = null;
    if(form instanceof Symbol)
    {
        Symbol sym = (Symbol) form;
        if(sym.ns == null) //if ns-qualified can't be classname
        {
            if(sym.name.indexOf('.') > 0 ||
                sym.name.charAt(0) == '[')
                className = sym.name;
            else
            {
                IPersistentMap imports =
                    (IPersistentMap)
                    ((Var) RT.NS_IMPORTS.get()).get();
                className = (String) imports.valAt(sym);
            }
        }
    }
    else if(stringOk && form instanceof String)
        className = (String) form;
    return className;
}
*/
static Class tagToClass(Object tag) throws Exception{
    Class c = maybeClass(tag, true);
    if(tag instanceof Symbol)
    {
        Symbol sym = (Symbol) tag;
        if(sym.ns == null) //if ns-qualified can't be classname
        {
            if(sym.name.equals("objects"))
                c = Object[].class;
            else if(sym.name.equals("ints"))
                c = int[].class;
            else if(sym.name.equals("longs"))

```

```

        c = long[].class;
    else if(sym.name.equals("floats"))
        c = float[].class;
    else if(sym.name.equals("doubles"))
        c = double[].class;
    else if(sym.name.equals("chars"))
        c = char[].class;
    else if(sym.name.equals("shorts"))
        c = short[].class;
    else if(sym.name.equals("bytes"))
        c = byte[].class;
    else if(sym.name.equals("booleans"))
        c = boolean[].class;
    }
}
if(c != null)
    return c;
throw new IllegalArgumentException(
    "Unable to resolve classname: " + tag);
}

static abstract class FieldExpr extends HostExpr{

static class InstanceFieldExpr extends FieldExpr
    implements AssignableExpr{
    public final Expr target;
    public final Class targetClass;
    public final java.lang.reflect.Field field;
    public final String fieldName;
    public final int line;
    public final Symbol tag;
    final static Method invokeNoArgInstanceMember =
        Method.getMethod(
            "Object invokeNoArgInstanceMember(Object,String)");
    final static Method setInstanceFieldMethod =
        Method.getMethod(
            "Object setInstanceField(Object,String,Object)");
    }

    public InstanceFieldExpr(int line,
                           Expr target,
                           String fieldName,
                           Symbol tag) throws Exception{
        this.target = target;
        this.targetClass =
            target.hasJavaClass() ? target.getJavaClass() : null;
        this.field =
            targetClass != null

```

```

        ? Reflector.getField(targetClass, fieldName, false)
        : null;
    this.fieldName = fieldName;
    this.line = line;
    this.tag = tag;
    if(field == null &&
       RT.booleanCast(RT.WARN_ON_REFLECTION.deref()))
    {
        RT.errPrintWriter()
        .format("Reflection warning, %s:%d - "+
               "reference to field %s can't be resolved.\n",
               SOURCE_PATH.deref(), line, fieldName);
    }
}

public Object eval() throws Exception{
    return
        Reflector.invokeNoArgInstanceMember(target.eval(), fieldName);
}

public boolean canEmitPrimitive(){
    return targetClass != null && field != null &&
           Util.isPrimitive(field.getType());
}

public void emitUnboxed(C context,
                        ObjExpr objx,
                        GeneratorAdapter gen){
    gen.visitLineNumber(line, gen.mark());
    if(targetClass != null && field != null)
    {
        target.emit(C.EXPRESSION, objx, gen);
        gen.checkCast(getType(targetClass));
        gen.getField(getType(targetClass), fieldName,
                     Type.getType(field.getType()));
    }
    else
        throw new UnsupportedOperationException(
            "Unboxed emit of unknown member");
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    gen.visitLineNumber(line, gen.mark());
    if(targetClass != null && field != null)
    {
        target.emit(C.EXPRESSION, objx, gen);
        gen.checkCast(getType(targetClass));
        gen.getField(getType(targetClass), fieldName,
                     Type.getType(field.getType()));
        //if(context != C.STATEMENT)
    }
}

```

```

        HostExpr.emitBoxReturn(objx, gen, field.getType());
        if(context == C.STATEMENT)
        {
            gen.pop();
        }
    }
    else
    {
        target.emit(C.EXPRESSION, objx, gen);
        gen.push(fieldName);
        gen.invokeStatic(REFLECTOR_TYPE, invokeNoArgInstanceMember);
        if(context == C.STATEMENT)
            gen.pop();
    }
}

public boolean hasJavaClass() throws Exception{
    return field != null || tag != null;
}

public Class getJavaClass() throws Exception{
    return tag != null ? HostExpr.tagToClass(tag) : field.getType();
}

public Object evalAssign(Expr val) throws Exception{
    return
        Reflector.setInstanceField(target.eval(),
                                    fieldName,
                                    val.eval());
}

public void emitAssign(C context,
                      ObjExpr objx,
                      GeneratorAdapter gen,
                      Expr val){
    gen.visitLineNumber(line, gen.mark());
    if(targetClass != null && field != null)
    {
        target.emit(C.EXPRESSION, objx, gen);
        gen.checkCast(Type.getType(targetClass));
        val.emit(C.EXPRESSION, objx, gen);
        gen.dupX1();
        HostExpr.emitUnboxArg(objx, gen, field.getType());
        gen.putField(Type.getType(targetClass), fieldName,
                    Type.getType(field.getType()));
    }
    else
    {
        target.emit(C.EXPRESSION, objx, gen);
        gen.push(fieldName);
    }
}

```

```

        val.emit(C.EXPRESSION, objx, gen);
        gen.invokeStatic(REFLECTOR_TYPE, setInstanceFieldMethod);
    }
    if(context == C.STATEMENT)
        gen.pop();
}
}

static class StaticFieldExpr extends FieldExpr implements AssignableExpr{
    //final String className;
    public final String fieldName;
    public final Class c;
    public final java.lang.reflect.Field field;
    public final Symbol tag;
    //    final static Method getStaticFieldMethod =
    //        Method.getMethod("Object getStaticField(String,String)");
    //    final static Method setStaticFieldMethod =
    //        Method.getMethod("Object setStaticField(String,String, Object)");
    final int line;

    public StaticFieldExpr(int line,
                          Class c,
                          String fieldName,
                          Symbol tag) throws Exception{
        //this.className = className;
        this.fieldName = fieldName;
        this.line = line;
        //c = Class.forName(className);
        this.c = c;
        field = c.getField(fieldName);
        this.tag = tag;
    }

    public Object eval() throws Exception{
        return Reflector.getStaticField(c, fieldName);
    }

    public boolean canEmitPrimitive(){
        return Util.isPrimitive(field.getType());
    }

    public void emitUnboxed(C context,
                           ObjExpr objx,
                           GeneratorAdapter gen){
        gen.visitLineNumber(line, gen.mark());
        gen.getStatic(Type.getType(c), fieldName,
                     Type.getType(field.getType()));
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){

```

```

        gen.visitLineNumber(line, gen.mark());

        gen.getStatic(Type.getType(c), fieldName,
                      Type.getType(field.getType()));
        //if(context != C.STATEMENT)
        HostExpr.emitBoxReturn(objx, gen, field.getType());
        if(context == C.STATEMENT)
        {
            gen.pop();
        }
        //gen.push(className);
        //gen.push(fieldName);
        //gen.invokeStatic(REFLECTOR_TYPE, getStaticFieldMethod);
    }

    public boolean hasJavaClass(){
        return true;
    }

    public Class getJavaClass() throws Exception{
        //Class c = Class.forName(className);
        //java.lang.reflect.Field field = c.getField(fieldName);
        return tag != null ? HostExpr.tagToClass(tag) : field.getType();
    }

    public Object evalAssign(Expr val) throws Exception{
        return Reflector.setStaticField(c, fieldName, val.eval());
    }

    public void emitAssign(C context, ObjExpr objx, GeneratorAdapter gen,
                          Expr val){
        gen.visitLineNumber(line, gen.mark());
        val.emit(C.EXPRESSION, objx, gen);
        gen.dup();
        HostExpr.emitUnboxArg(objx, gen, field.getType());
        gen.putStatic(Type.getType(c),
                      fieldName, Type.getType(field.getType()));
        if(context == C.STATEMENT)
            gen.pop();
    }

}

static Class maybePrimitiveType(Expr e){
    try
    {
        if(e instanceof MaybePrimitiveExpr &&
           e.hasClass() &&
           ((MaybePrimitiveExpr)e).canEmitPrimitive())

```

```

    {
        Class c = e.getJavaClass();
        if(Util.isPrimitive(c))
            return c;
        }
    }
catch(Exception ex)
{
    throw new RuntimeException(ex);
}
return null;
}

static abstract class MethodExpr extends HostExpr{
    static void emitArgsAsArray(IPersistentVector args,
                                ObjExpr objx,
                                GeneratorAdapter gen){
        gen.push(args.count());
        gen newArray(OBJECT_TYPE);
        for(int i = 0; i < args.count(); i++)
        {
            gen.dup();
            gen.push(i);
            ((Expr) args.nth(i)).emit(C.EXPRESSION, objx, gen);
            gen.arrayStore(OBJECT_TYPE);
        }
    }

    public static void emitTypedArgs(ObjExpr objx,
                                    GeneratorAdapter gen,
                                    Class[] parameterTypes,
                                    IPersistentVector args){
        for(int i = 0; i < parameterTypes.length; i++)
        {
            Expr e = (Expr) args.nth(i);
            try
            {
                final Class primc = maybePrimitiveType(e);
                if(primc == parameterTypes[i])
                {
                    final MaybePrimitiveExpr pe =
                        (MaybePrimitiveExpr) e;
                    pe.emitUnboxed(C.EXPRESSION, objx, gen);
                }
                else if(primc == int.class &&
                        parameterTypes[i] == long.class)
                {
                    final MaybePrimitiveExpr pe =
                        (MaybePrimitiveExpr) e;
                    pe.emitUnboxed(C.EXPRESSION, objx, gen);
                }
            }
        }
    }
}

```

```

        gen.visitInsn(I2L);
    }
    else if(primc == long.class &&
            parameterTypes[i] == int.class)
    {
        final MaybePrimitiveExpr pe =
            (MaybePrimitiveExpr) e;
        pe.emitUnboxed(C.EXPRESSION, objx, gen);
        if(RT.booleanCast(UNCHECKED_MATH.deref()))
            gen.invokeStatic(RT_TYPE,
                Method.getMethod("int uncheckedIntCast(long)"));
        else
            gen.invokeStatic(RT_TYPE,
                Method.getMethod("int intCast(long)"));
    }
    else if(primc == float.class &&
            parameterTypes[i] == double.class)
    {
        final MaybePrimitiveExpr pe = (MaybePrimitiveExpr) e;
        pe.emitUnboxed(C.EXPRESSION, objx, gen);
        gen.visitInsn(F2D);
    }
    else if(primc == double.class &&
            parameterTypes[i] == float.class)
    {
        final MaybePrimitiveExpr pe = (MaybePrimitiveExpr) e;
        pe.emitUnboxed(C.EXPRESSION, objx, gen);
        gen.visitInsn(D2F);
    }
    else
    {
        e.emit(C.EXPRESSION, objx, gen);
        HostExpr.emitUnboxArg(objx, gen, parameterTypes[i]);
    }
}
catch(Exception e1)
{
    e1.printStackTrace(RT.errPrintWriter());
}

}

}

static class InstanceMethodExpr extends MethodExpr{
    public final Expr target;
    public final String methodName;
    public final IPersistentVector args;
    public final String source;
    public final int line;
}

```

```
public final Symbol tag;
public final java.lang.reflect.Method method;

final static Method invokeInstanceMethodMethod =
    Method.getMethod(
        "Object invokeInstanceMethod(Object, String, Object[])");

public InstanceMethodExpr(String source,
                           int line,
                           Symbol tag,
                           Expr target,
                           String methodName,
                           IPersistentVector args)
    throws Exception{
this.source = source;
this.line = line;
this.args = args;
this.methodName = methodName;
this.target = target;
this.tag = tag;
if(target.hasJavaClass() && target.getJavaClass() != null)
{
    List methods =
        Reflector.getMethods(target.getJavaClass(),
                             args.count(), methodName, false);
    if(methods.isEmpty())
        method = null;
    //throw new IllegalArgumentException(
    //           "No matching method found");
    else
    {
        int methodidx = 0;
        if(methods.size() > 1)
        {
            ArrayList<Class[]> params = new ArrayList();
            ArrayList<Class> rets = new ArrayList();
            for(int i = 0; i < methods.size(); i++)
            {
                java.lang.reflect.Method m =
                    (java.lang.reflect.Method) methods.get(i);
                params.add(m.getParameterTypes());
                rets.add(m.getReturnType());
            }
            methodidx =
                getMatchingParams(methodName, params, args, rets);
        }
        java.lang.reflect.Method m =
            (java.lang.reflect.Method)
                (methodidx >= 0
```

```

                ? methods.get(methodidx)
                : null);
        if(m != null &&
           !Modifier.isPublic(m.getDeclaringClass()
                               .getModifiers()))
        {
            //public method of non-public class, try to find
            // it in hierarchy
            m = Reflector.getAsMethodOfPublicBase(
                m.getDeclaringClass(), m);
        }
        method = m;
    }
}
else
    method = null;

if(method == null &&
   RT.booleanCast(RT.WARN_ON_REFLECTION.deref()))
{
    RT.errPrintWriter()
        .format(
    "Reflection warning, %s:%d - call to %s can't be resolved.\n",
        SOURCE_PATH.deref(), line, methodName);
}
}

public Object eval() throws Exception{
try
{
    Object targetval = target.eval();
    Object[] argvals = new Object[args.count()];
    for(int i = 0; i < args.count(); i++)
        argvals[i] = ((Expr) args.nth(i)).eval();
    if(method != null)
    {
        LinkedList ms = new LinkedList();
        ms.add(method);
        return Reflector.invokeMatchingMethod(methodName, ms,
                                              targetval,
                                              argvals);
    }
    return Reflector.invokeInstanceMethod(targetval,
                                           methodName,
                                           argvals);
}
catch(Throwable e)
{
    if(!(e instanceof CompilerException))
        throw new CompilerException(source, line, e);
}
}

```

```

        else
            throw (CompilerException) e;
    }
}

public boolean canEmitPrimitive(){
    return method != null &&
           Util.isPrimitive(method.getReturnType());
}

public void emitUnboxed(C context,
                        ObjExpr objx,
                        GeneratorAdapter gen){
    gen.visitLineNumber(line, gen.mark());
    if(method != null)
    {
        Type type = Type.getType(method.getDeclaringClass());
        target.emit(C.EXPRESSION, objx, gen);
        //if(!method.getDeclaringClass().isInterface())
        gen.checkCast(type);
        MethodExpr.emitTypedArgs(objx, gen,
                                 method.getParameterTypes(), args);
        if(context == C.RETURN)
        {
            ObjMethod method = (ObjMethod) METHOD.deref();
            method.emitClearLocals(gen);
        }
        Method m = new Method(methodName,
                              Type.getReturnType(method),
                              Type.getArgumentTypes(method));
        if(method.getDeclaringClass().isInterface())
            gen.invokeInterface(type, m);
        else
            gen.invokeVirtual(type, m);
    }
    else
        throw new UnsupportedOperationException(
            "Unboxed emit of unknown member");
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    gen.visitLineNumber(line, gen.mark());
    if(method != null)
    {
        Type type = Type.getType(method.getDeclaringClass());
        target.emit(C.EXPRESSION, objx, gen);
        //if(!method.getDeclaringClass().isInterface())
        gen.checkCast(type);
        MethodExpr.emitTypedArgs(objx, gen,
                                 method.getParameterTypes(), args);
    }
}

```

```

        if(context == C.RETURN)
        {
            ObjMethod method = (ObjMethod) METHOD.deref();
            method.emitClearLocals(gen);
        }
        Method m = new Method(methodName,
                               Type.getReturnType(method),
                               Type.getArgumentTypes(method));
        if(method.getDeclaringClass().isInterface())
            gen.invokeInterface(type, m);
        else
            gen.invokeVirtual(type, m);
        //if(context != C.STATEMENT ||
        //    method.getReturnType() == Void.TYPE)
        HostExpr.emitBoxReturn(objx, gen, method.getReturnType());
    }
    else
    {
        target.emit(C.EXPRESSION, objx, gen);
        gen.push(methodName);
        emitArgsAsArray(args, objx, gen);
        if(context == C.RETURN)
        {
            ObjMethod method = (ObjMethod) METHOD.deref();
            method.emitClearLocals(gen);
        }
        gen.invokeStatic(REFLECTOR_TYPE, invokeInstanceMethodMethod);
    }
    if(context == C.STATEMENT)
        gen.pop();
}

public boolean hasJavaClass(){
    return method != null || tag != null;
}

public Class getJavaClass() throws Exception{
    return tag != null
        ? HostExpr.tagToClass(tag)
        : method.getReturnType();
}
}

static class StaticMethodExpr extends MethodExpr{
    //final String className;
    public final Class c;
    public final String methodName;
    public final IPersistentVector args;
    public final String source;
}

```

```

public final int line;
public final java.lang.reflect.Method method;
public final Symbol tag;
final static Method forNameMethod =
    Method.getMethod("Class forName(String)");
final static Method invokeStaticMethodMethod =
    Method.getMethod(
        "Object invokeStaticMethod(Class, String, Object[])");

public StaticMethodExpr(String source, int line, Symbol tag,
                       Class c, String methodName,
                       IPersistentVector args)
    throws Exception{
this.c = c;
this.methodName = methodName;
this.args = args;
this.source = source;
this.line = line;
this.tag = tag;

List methods =
    Reflector.getMethods(c, args.count(), methodName, true);
if(methods.isEmpty())
    throw new IllegalArgumentException(
        "No matching method: " + methodName);

int methodidx = 0;
if(methods.size() > 1)
{
    ArrayList<Class[]> params = new ArrayList();
    ArrayList<Class> rets = new ArrayList();
    for(int i = 0; i < methods.size(); i++)
    {
        java.lang.reflect.Method m =
            (java.lang.reflect.Method) methods.get(i);
        params.add(m.getParameterTypes());
        rets.add(m.getReturnType());
    }
    methodidx =
        getMatchingParams(methodName, params, args, rets);
}
method = (java.lang.reflect.Method)
    (methodidx >= 0 ? methods.get(methodidx) : null);
if(method == null &&
    RT.booleanCast(RT.WARN_ON_REFLECTION.deref()))
{
    RT.errPrintWriter()
        .format(
    "Reflection warning, %s:%d - call to %s can't be resolved.\n",

```

```

        SOURCE_PATH.deref(), line, methodName);
    }
}

public Object eval() throws Exception{
    try
    {
        Object[] argvals = new Object[args.count()];
        for(int i = 0; i < args.count(); i++)
            argvals[i] = ((Expr) args.nth(i)).eval();
        if(method != null)
        {
            LinkedList ms = new LinkedList();
            ms.add(method);
            return Reflector.invokeMatchingMethod(methodName, ms,
                null, argvals);
        }
        return Reflector.invokeStaticMethod(c, methodName, argvals);
    }
    catch(Throwable e)
    {
        if(!(e instanceof CompilerException))
            throw new CompilerException(source, line, e);
        else
            throw (CompilerException) e;
    }
}

public boolean canEmitPrimitive(){
    return method != null &&
        Util.isPrimitive(method.getReturnType());
}

public void emitUnboxed(C context,
    ObjExpr objx,
    GeneratorAdapter gen){
    gen.visitLineNumber(line, gen.mark());
    if(method != null)
    {
        MethodExpr.emitTypedArgs(objx, gen,
            method.getParameterTypes(),
            args);
        //Type type =
        // Type.getObjectType(className.replace('.', '/'));
        if(context == C.RETURN)
        {
            ObjMethod method = (ObjMethod) METHOD.deref();
            method.emitClearLocals(gen);
        }
        Type type = Type.getType(c);
    }
}

```

```

Method m = new Method(methodName,
                      Type.getType(method),
                      Type.getArgumentTypes(method));
gen.invokeStatic(type, m);
}
else
    throw new UnsupportedOperationException(
        "Unboxed emit of unknown member");
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    gen.visitLineNumber(line, gen.mark());
    if(method != null)
    {
        MethodExpr.emitTypedArgs(objx, gen,
                                 method.getParameterTypes(),
                                 args);
        //Type type =
        //  Type.getObjectType(className.replace('.', '/'));
        if(context == C.RETURN)
        {
            ObjMethod method = (ObjMethod) METHOD.deref();
            method.emitClearLocals(gen);
        }
        Type type = Type.getType(c);
        Method m = new Method(methodName,
                              Type.getType(method),
                              Type.getArgumentTypes(method));
        gen.invokeStatic(type, m);
        //if(context != C.STATEMENT ||
        //  method.getType() == Void.TYPE)
        Class retClass = method.getType();
        if(context == C.STATEMENT)
        {
            if(retClass == long.class || retClass == double.class)
                gen.pop2();
            else if(retClass != void.class)
                gen.pop();
        }
        else
        {
            HostExpr.emitBoxReturn(objx, gen,
                                   method.getType());
        }
    }
    else
    {
        gen.push(c.getName());
        gen.invokeStatic(CLASS_TYPE, forNameMethod);
        gen.push(methodName);
    }
}

```

```

        emitArgsAsArray(args, objx, gen);
        if(context == C.RETURN)
        {
            ObjMethod method = (ObjMethod) METHOD.deref();
            method.emitClearLocals(gen);
        }
        gen.invokeStatic(REFLECTOR_TYPE, invokeStaticMethodMethod);
        if(context == C.STATEMENT)
            gen.pop();
    }
}

public boolean hasJavaClass(){
    return method != null || tag != null;
}

public Class getJavaClass() throws Exception{
    return tag != null
        ? HostExpr.tagToClass(tag)
        : method.getReturnType();
}
}

static class UnresolvedVarExpr implements Expr
{
    public final Symbol symbol;

    public UnresolvedVarExpr(Symbol symbol){
        this.symbol = symbol;
    }

    public boolean hasJavaClass(){
        return false;
    }

    public Class getJavaClass() throws Exception{
        throw new IllegalArgumentException(
            "UnresolvedVarExpr has no Java class");
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    }

    public Object eval() throws Exception{
        throw new IllegalArgumentException(
            "UnresolvedVarExpr cannot be evalled");
    }
}

static class NumberExpr extends LiteralExpr
    implements MaybePrimitiveExpr{

```

```
final Number n;
public final int id;

public NumberExpr(Number n){
    this.n = n;
    this.id = registerConstant(n);
}

Object val(){
    return n;
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    if(context != C.STATEMENT)
    {
        objx.emitConstant(gen, id);
    }
    //      emitUnboxed(context,objx,gen);
    //      HostExpr.emitBoxReturn(objx,gen,getJavaClass());
}
}

public boolean hasJavaClass() throws Exception{
    return true;
}

public Class getJavaClass(){
    if(n instanceof Integer)
        return long.class;
    else if(n instanceof Double)
        return double.class;
    else if(n instanceof Long)
        return long.class;
    else
        throw new IllegalStateException(
            "Unsupported Number type: " + n.getClass().getName());
}

public boolean canEmitPrimitive(){
    return true;
}

public void emitUnboxed(C context,
                       ObjExpr objx,
                       GeneratorAdapter gen){
    if(n instanceof Integer)
        gen.push(n.longValue());
    else if(n instanceof Double)
        gen.push(n.doubleValue());
    else if(n instanceof Long)
        gen.push(n.longValue());
```

```

    }

    static public Expr parse(Number form){
        if(form instanceof Integer
            || form instanceof Double
            || form instanceof Long)
            return new NumberExpr(form);
        else
            return new ConstantExpr(form);
    }
}

static class ConstantExpr extends LiteralExpr{
    //stuff quoted vals in classloader at compile time, pull out
    //at runtime this won't work for static compilation...
    public final Object v;
    public final int id;

    public ConstantExpr(Object v){
        this.v = v;
        this.id = registerConstant(v);
        //    this.id = RT.nextID();
        //    DynamicClassLoader loader = (DynamicClassLoader) LOADER.get();
        //    loader.registerQuotedVal(id, v);
    }

    Object val(){
        return v;
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        objx.emitConstant(gen, id);
        if(context == C.STATEMENT)
        {
            gen.pop();
            //            gen.loadThis();
            //            gen.invokeVirtual(OBJECT_TYPE, getClassMethod());
            //            gen.invokeVirtual(CLASS_TYPE, getClassLoaderMethod());
            //            gen.checkCast(DYNAMIC_CLASSLOADER_TYPE);
            //            gen.push(id);
            //            gen.invokeVirtual(DYNAMIC_CLASSLOADER_TYPE,
            //                            getQuotedValMethod());
        }
    }

    public boolean hasJavaClass(){
        return Modifier.isPublic(v.getClass().getModifiers());
        //return false;
    }
}

```

```

public Class getJavaClass() throws Exception{
    return v.getClass();
    //throw new IllegalArgumentException("Has no Java class");
}

static class Parser implements IParser{
    public Expr parse(C context, Object form){
        Object v = RT.second(form);

        if(v == null)
            return NIL_EXPR;
        //        Class fclass = v.getClass();
        //        if(fclass == Keyword.class)
        //            return registerKeyword((Keyword) v);
        //        else if(v instanceof Num)
        //            return new NumExpr((Num) v);
        //        else if(fclass == String.class)
        //            return new StringExpr((String) v);
        //        else if(fclass == Character.class)
        //            return new CharExpr((Character) v);
        //        else if(v instanceof IPersistentCollection &&
        //                ((IPersistentCollection) v).count() == 0)
        //            return new EmptyExpr(v);
        //        else
        //            return new ConstantExpr(v);
    }
}

static class NilExpr extends LiteralExpr{
    Object val(){
        return null;
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        gen.visitInsn(Opcodes.ACONST_NULL);
        if(context == C.STATEMENT)
            gen.pop();
    }

    public boolean hasJavaClass(){
        return true;
    }

    public Class getJavaClass() throws Exception{
        return null;
    }
}

final static NilExpr NIL_EXPR = new NilExpr();

```

```

static class BooleanExpr extends LiteralExpr{
    public final boolean val;

    public BooleanExpr(boolean val){
        this.val = val;
    }

    Object val(){
        return val ? RT.T : RT.F;
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        if(val)
            gen.getStatic(BOOLEAN_OBJECT_TYPE, "TRUE",
                         BOOLEAN_OBJECT_TYPE);
        else
            gen.getStatic(BOOLEAN_OBJECT_TYPE, "FALSE",
                         BOOLEAN_OBJECT_TYPE);
        if(context == C.STATEMENT)
        {
            gen.pop();
        }
    }

    public boolean hasJavaClass(){
        return true;
    }

    public Class getJavaClass() throws Exception{
        return Boolean.class;
    }
}

final static BooleanExpr TRUE_EXPR = new BooleanExpr(true);
final static BooleanExpr FALSE_EXPR = new BooleanExpr(false);

static class StringExpr extends LiteralExpr{
    public final String str;

    public StringExpr(String str){
        this.str = str;
    }

    Object val(){
        return str;
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){

```

```
        if(context != C.STATEMENT)
            gen.push(str);
    }

    public boolean hasJavaClass(){
        return true;
    }

    public Class getJavaClass() throws Exception{
        return String.class;
    }
}

static class MonitorEnterExpr extends UntypedExpr{
    final Expr target;

    public MonitorEnterExpr(Expr target){
        this.target = target;
    }

    public Object eval() throws Exception{
        throw
            new UnsupportedOperationException("Can't eval monitor-enter");
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        target.emit(C.EXPRESSION, objx, gen);
        gen.monitorEnter();
        NIL_EXPR.emit(context, objx, gen);
    }

    static class Parser implements IParser{
        public Expr parse(C context, Object form) throws Exception{
            return
                new MonitorEnterExpr(analyze(C.EXPRESSION, RT.second(form)));
        }
    }
}

static class MonitorExitExpr extends UntypedExpr{
    final Expr target;

    public MonitorExitExpr(Expr target){
        this.target = target;
    }

    public Object eval() throws Exception{
        throw
            new UnsupportedOperationException("Can't eval monitor-exit");
    }
}
```

```

    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        target.emit(C.EXPRESSION, objx, gen);
        gen.monitorExit();
        NIL_EXPR.emit(context, objx, gen);
    }

    static class Parser implements IParser{
        public Expr parse(C context, Object form) throws Exception{
            return
                new MonitorExitExpr(analyze(C.EXPRESSION, RT.second(form)));
        }
    }

    public static class TryExpr implements Expr{
        public final Expr tryExpr;
        public final Expr finallyExpr;
        public final PersistentVector catchExprs;
        public final int retLocal;
        public final int finallyLocal;

        public static class CatchClause{
            //final String className;
            public final Class c;
            public final LocalBinding lb;
            public final Expr handler;
            Label label;
            Label endLabel;

            public CatchClause(Class c, LocalBinding lb, Expr handler){
                this.c = c;
                this.lb = lb;
                this.handler = handler;
            }
        }

        public TryExpr(Expr tryExpr,
                      PersistentVector catchExprs,
                      Expr finallyExpr,
                      int retLocal,
                      int finallyLocal){
            this.tryExpr = tryExpr;
            this.catchExprs = catchExprs;
            this.finallyExpr = finallyExpr;
            this.retLocal = retLocal;
            this.finallyLocal = finallyLocal;
        }
    }
}

```

```

    }

    public Object eval() throws Exception{
        throw new UnsupportedOperationException("Can't eval try");
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        Label startTry = gen.newLabel();
        Label endTry = gen.newLabel();
        Label end = gen.newLabel();
        Label ret = gen.newLabel();
        Label finallyLabel = gen.newLabel();
        for(int i = 0; i < catchExprs.count(); i++)
        {
            CatchClause clause = (CatchClause) catchExprs.nth(i);
            clause.label = gen.newLabel();
            clause.endLabel = gen.newLabel();
        }

        gen.mark(startTry);
        tryExpr.emit(context, objx, gen);
        if(context != C.STATEMENT)
            gen.visitVarInsn(OBJECT_TYPE.getOpcode(OpCodes.ISTORE),
                            retLocal);
        gen.mark(endTry);
        if(finallyExpr != null)
            finallyExpr.emit(C.STATEMENT, objx, gen);
        gen.goTo(ret);

        for(int i = 0; i < catchExprs.count(); i++)
        {
            CatchClause clause = (CatchClause) catchExprs.nth(i);
            gen.mark(clause.label);
            //exception should be on stack
            //put in clause local
            gen.visitVarInsn(OBJECT_TYPE.getOpcode(OpCodes.ISTORE),
                            clause.lb.idx);
            clause.handler.emit(context, objx, gen);
            if(context != C.STATEMENT)
                gen.visitVarInsn(OBJECT_TYPE.getOpcode(OpCodes.ISTORE),
                                retLocal);
            gen.mark(clause.endLabel);

            if(finallyExpr != null)
                finallyExpr.emit(C.STATEMENT, objx, gen);
            gen.goTo(ret);
        }
        if(finallyExpr != null)
        {
            gen.mark(finallyLabel);
        }
    }
}

```

```

//exception should be on stack
gen.visitVarInsn(OBJECT_TYPE.getOpcode(Opcodes.ISTORE),
    finallyLocal);
finallyExpr.emit(C.STATEMENT, objx, gen);
gen.visitVarInsn(OBJECT_TYPE.getOpcode(Opcodes.ILOAD),
    finallyLocal);
gen.throwException();
}
gen.mark(ret);
if(context != C.STATEMENT)
    gen.visitVarInsn(OBJECT_TYPE.getOpcode(Opcodes.ILOAD),
        retLocal);
gen.mark(end);
for(int i = 0; i < catchExprs.count(); i++)
{
    CatchClause clause = (CatchClause) catchExprs.nth(i);
    gen.visitTryCatchBlock(startTry, endTry, clause.label,
        clause.c.getName()
            .replace('.', '/'));
}
if(finallyExpr != null)
{
    gen.visitTryCatchBlock(startTry, endTry,
        finallyLabel, null);
    for(int i = 0; i < catchExprs.count(); i++)
    {
        CatchClause clause = (CatchClause) catchExprs.nth(i);
        gen.visitTryCatchBlock(clause.label,
            clause.endLabel,
            finallyLabel, null);
    }
}
for(int i = 0; i < catchExprs.count(); i++)
{
    CatchClause clause = (CatchClause) catchExprs.nth(i);
    gen.visitLocalVariable(clause.lb.name,
        "Ljava/lang/Object;", null,
        clause.label, clause.endLabel,
        clause.lb.idx);
}
}

public boolean hasJavaClass() throws Exception{
    return tryExpr.hasJavaClass();
}

public Class getJavaClass() throws Exception{
    return tryExpr.getJavaClass();
}

```

```

static class Parser implements IParser{

    public Expr parse(C context, Object frm) throws Exception{
        ISeq form = (ISeq) frm;
        //
        if(context == C.EVAL || context == C.EXPRESSION)
        if(context != C.RETURN)
            return analyze(context,
                           RT.list(
                               RT.list(FN,
                                       PersistentVector.EMPTY, form)));
        //((try try-expr* catch-expr* finally-expr?))
        //catch-expr: (catch class sym expr*)
        //finally-expr: (finally expr*)

        PersistentVector body = PersistentVector.EMPTY;
        PersistentVector catches = PersistentVector.EMPTY;
        Expr bodyExpr = null;
        Expr finallyExpr = null;
        boolean caught = false;

        int retLocal = getAndIncLocalNum();
        int finallyLocal = getAndIncLocalNum();
        for(ISeq fs = form.next(); fs != null; fs = fs.next())
        {
            Object f = fs.first();
            Object op =
                (f instanceof ISeq) ? ((ISeq) f).first() : null;
            if(!Util.equals(op, CATCH) &&
               !Util.equals(op, FINALLY))
            {
                if(caught)
                    throw new Exception(
                        "Only catch or finally clause can follow catch in try expression");
                body = body.cons(f);
            }
            else
            {
                if(bodyExpr == null)
                    try {
                        Var.pushThreadBindings(RT.map(NO_RECUR, true));
                        bodyExpr =
                            (new BodyExpr.Parser())
                                .parse(context, RT.seq(body));
                    } finally {
                        Var.popThreadBindings();
                    }
                if(Util.equals(op, CATCH))
                {
                    Class c =

```

```

        HostExpr.maybeClass(RT.second(f), false);
        if(c == null)
            throw new IllegalArgumentException(
                "Unable to resolve classname: " +
                RT.second(f));
        if(!(RT.third(f) instanceof Symbol))
            throw new IllegalArgumentException(
                "Bad binding form, expected symbol, got: " +
                + RT.third(f));
        Symbol sym = (Symbol) RT.third(f);
        if(sym.getNamespace() != null)
            throw new Exception(
                "Can't bind qualified name:" + sym);

IPersistentMap dynamicBindings =
    RT.map(LOCAL_ENV, LOCAL_ENV.deref(),
           NEXT_LOCAL_NUM, NEXT_LOCAL_NUM.deref(),
           IN_CATCH_FINALLY, RT.T);
try
{
    Var.pushThreadBindings(dynamicBindings);
    LocalBinding lb =
        registerLocal(sym,
                      (Symbol)
                      (RT.second(f) instanceof Symbol
                       ? RT.second(f)
                       : null),
                      null, false);
    Expr handler =
        (new BodyExpr.Parser())
        .parse(context,
               RT.next(RT.next(RT.next(f))));
    catches =
        catches.cons(
            new CatchClause(c, lb, handler));
}
finally
{
    Var.popThreadBindings();
}
caught = true;
}
else //finally
{
    if(fs.next() != null)
        throw new Exception(
            "finally clause must be last in try expression");
    try
    {
        Var.pushThreadBindings(

```

```

        RT.map(IN_CATCH_FINALLY, RT.T));
        finallyExpr =
            (new BodyExpr.Parser())
                .parse(C.STATEMENT, RT.next(f));
        }
        finally
        {
            Var.popThreadBindings();
        }
    }
}

if(bodyExpr == null) {
    try
    {
        Var.pushThreadBindings(
            RT.map(NO_RECUR, true));
        bodyExpr =
            (new BodyExpr.Parser())
                .parse(context, RT.seq(body));
    }
    finally
    {
        Var.popThreadBindings();
    }
}

return new TryExpr(bodyExpr, catches, finallyExpr, retLocal,
                    finallyLocal);
}
}
}

//static class TryFinallyExpr implements Expr{
//    final Expr tryExpr;
//    final Expr finallyExpr;
//
//    public TryFinallyExpr(Expr tryExpr, Expr finallyExpr){
//        this.tryExpr = tryExpr;
//        this.finallyExpr = finallyExpr;
//    }
//
//    public Object eval() throws Exception{
//        throw new UnsupportedOperationException("Can't eval try");
//    }
//
//    public void emit(C context, FnExpr fn, GeneratorAdapter gen){
//        Label startTry = gen.newLabel();
//        Label endTry = gen.newLabel();

```

```

//      Label end = gen.newLabel();
//      Label finallyLabel = gen.newLabel();
//      gen.visitTryCatchBlock(startTry, endTry, finallyLabel, null);
//      gen.mark(startTry);
//      tryExpr.emit(context, fn, gen);
//      gen.mark(endTry);
//      finallyExpr.emit(C.STATEMENT, fn, gen);
//      gen.goTo(end);
//      gen.mark(finallyLabel);
//      //exception should be on stack
//      finallyExpr.emit(C.STATEMENT, fn, gen);
//      gen.throwException();
//      gen.mark(end);
//  }
//
//  public boolean hasJavaClass() throws Exception{
//      return tryExpr.hasJavaClass();
//  }
//
//  public Class getJavaClass() throws Exception{
//      return tryExpr.getJavaClass();
//  }
//
//  static class Parser implements IParser{
//      public Expr parse(C context, Object frm) throws Exception{
//          ISeq form = (ISeq) frm;
//          //((try-finally try-expr finally-expr)
//          if(form.count() != 3)
//              throw new IllegalArgumentException(
// "Wrong number of arguments, expecting: "+
// "(try-finally try-expr finally-expr) ");
//          if(context == C.EVAL || context == C.EXPRESSION)
//              return analyze(context,
//                  RT.list(
//                      RT.list(FN,
//                          PersistentVector.EMPTY,
//                          form)));
//          return
//              new TryFinallyExpr(analyze(context, RT.second(form)),
//                  analyze(C.STATEMENT, RT.third(form)));
//      }
//  }
//}

static class ThrowExpr extends UntypedExpr{
    public final Expr excExpr;

    public ThrowExpr(Expr excExpr){

```

```

        this.excExpr = excExpr;
    }

    public Object eval() throws Exception{
        throw new Exception("Can't eval throw");
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        excExpr.emit(C.EXPRESSION, objx, gen);
        gen.checkCast(THROWABLE_TYPE);
        gen.throwException();
    }

    static class Parser implements IParser{
        public Expr parse(C context, Object form) throws Exception{
            if(context == C.EVAL)
                return
                    analyze(context,
                        RT.list(
                            RT.list(FN, PersistentVector.EMPTY, form)));
            return new ThrowExpr(analyze(C.EXPRESSION, RT.second(form)));
        }
    }
}

static public boolean subsumes(Class[] c1, Class[] c2){
    //presumes matching lengths
    Boolean better = false;
    for(int i = 0; i < c1.length; i++)
    {
        if(c1[i] != c2[i])
            // || c2[i].isPrimitive() && c1[i] == Object.class)
            {
                if(!c1[i].isPrimitive() && c2[i].isPrimitive()
                    //|| Number.class.isAssignableFrom(c1[i]) &&
                    // c2[i].isPrimitive()
                    ||
                    c2[i].isAssignableFrom(c1[i]))
                    better = true;
                else
                    return false;
            }
    }
    return better;
}

static int getMatchingParams(String methodName,
                             ArrayList<Class[]> paramlists,

```

```

IPersistentVector argexprs,
List<Class> rets)
throws Exception{
//presumes matching lengths
int matchIdx = -1;
boolean tied = false;
boolean foundExact = false;
for(int i = 0; i < paramlists.size(); i++)
{
    boolean match = true;
    ISeq aseq = argexprs.seq();
    int exact = 0;
    for(int p = 0; match &&
                    p < argexprs.count() &&
                    aseq != null;
                    ++p, aseq = aseq.next())
    {
        Expr arg = (Expr) aseq.first();
        Class aclass =
            arg.hasJavaClass() ? arg.getJavaClass() : Object.class;
        Class pclass = paramlists.get(i)[p];
        if(arg.hasJavaClass() && aclass == pclass)
            exact++;
        else
            match = Reflector.paramArgTypeMatch(pclass, aclass);
    }
    if(exact == argexprs.count())
    {
        if(!foundExact ||
           matchIdx == -1 ||
           rets.get(matchIdx).isAssignableFrom(rets.get(i)))
            matchIdx = i;
        foundExact = true;
    }
    else if(match && !foundExact)
    {
        if(matchIdx == -1)
            matchIdx = i;
        else
        {
            if(subsumes(paramlists.get(i), paramlists.get(matchIdx)))
            {
                matchIdx = i;
                tied = false;
            }
            else if(Arrays.equals(paramlists.get(matchIdx),
                                paramlists.get(i)))
            {
                if(rets.get(matchIdx).isAssignableFrom(rets.get(i)))
                    matchIdx = i;
            }
        }
    }
}

```

```

        }
        else if(!(subsumes(paramlists.get(matchIdx),
                           paramlists.get(i))))
            tied = true;
    }
}
if(tied)
    throw new IllegalArgumentException(
        "More than one matching method found: " + methodName);

return matchIdx;
}

public static class NewExpr implements Expr{
    public final IPersistentVector args;
    public final Constructor ctor;
    public final Class c;
    final static Method invokeConstructorMethod =
        Method.getMethod("Object invokeConstructor(Class, Object[])");
//    final static Method forNameMethod =
//        Method.getMethod("Class classForName(String)");
    final static Method forNameMethod =
        Method.getMethod("Class forName(String)");

    public NewExpr(Class c, IPersistentVector args, int line)
        throws Exception{
        this.args = args;
        this.c = c;
        Constructor[] allctors = c.getConstructors();
        ArrayList ctors = new ArrayList();
        ArrayList<Class[]> params = new ArrayList();
        ArrayList<Class> rets = new ArrayList();
        for(int i = 0; i < allctors.length; i++)
        {
            Constructor ctor = allctors[i];
            if(ctor.getParameterTypes().length == args.count())
            {
                ctors.add(ctor);
                params.add(ctor.getParameterTypes());
                rets.add(c);
            }
        }
        if(ctors.isEmpty())
            throw new IllegalArgumentException(
                "No matching ctor found for " + c);

        int ctoridx = 0;
        if(ctors.size() > 1)

```

```

    {
      ctoridx = getMatchingParams(c.getName(), params, args, rets);
    }

    this.ctor =
      ctoridx >= 0 ? (Constructor) ctors.get(ctoridx) : null;
    if(ctor == null &&
      RT.booleanCast(RT.WARN_ON_REFLECTION.deref()))
    {
      RT.errPrintWriter()
        .format(
          "Reflection warning, %s:%d - call to %s ctor can't be resolved.\n",
          SOURCE_PATH.deref(), line, c.getName());
    }
  }

  public Object eval() throws Exception{
    Object[] argvals = new Object[args.count()];
    for(int i = 0; i < args.count(); i++)
      argvals[i] = ((Expr) args.nth(i)).eval();
    if(this.ctor != null)
    {
      return
        ctor.newInstance(
          Reflector.boxArgs(ctor.getParameterTypes(), argvals));
    }
    return Reflector.invokeConstructor(c, argvals);
  }

  public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    if(this.ctor != null)
    {
      Type type = getType(c);
      gen.newInstance(type);
      gen.dup();
      MethodExpr.emitTypedArgs(objx, gen, ctor.
        getParameterTypes(), args);
      if(context == C.RETURN)
      {
        ObjMethod method = (ObjMethod) METHOD.deref();
        method.emitClearLocals(gen);
      }
      gen.invokeConstructor(type,
        new Method("<init>",
          Type.getConstructorDescriptor(ctor)));
    }
    else
    {
      gen.push(destubClassName(c.getName()));
      gen.invokeStatic(CLASS_TYPE, forNameMethod);
    }
  }
}

```

```

        MethodExpr.emitArgsAsArray(args, objx, gen);
        if(context == C.RETURN)
        {
            ObjMethod method = (ObjMethod) METHOD.deref();
            method.emitClearLocals(gen);
        }
        gen.invokeStatic(REFLECTOR_TYPE, invokeConstructorMethod);
    }
    if(context == C.STATEMENT)
        gen.pop();
}

public boolean hasJavaClass(){
    return true;
}

public Class getJavaClass() throws Exception{
    return c;
}

static class Parser implements IParser{
    public Expr parse(C context, Object frm) throws Exception{
        int line = (Integer) LINE.deref();
        ISeq form = (ISeq) frm;
        //((new Classname args...))
        if(form.count() < 2)
            throw new Exception(
"wrong number of arguments, expecting: (new Classname args...)");
        Class c = HostExpr.maybeClass(RT.second(form), false);
        if(c == null)
            throw new IllegalArgumentException(
"Unable to resolve classname: " + RT.second(form));
        PersistentVector args = PersistentVector.EMPTY;
        for(ISeq s = RT.next(RT.next(form)); s != null; s = s.next())
            args =
                args.cons(analyze(context == C.EVAL
                    ? context
                    : C.EXPRESSION, s.first())));
        return new NewExpr(c, args, line);
    }
}

public static class MetaExpr implements Expr{
    public final Expr expr;
    public final Expr meta;
    final static Type IOBJ_TYPE = Type.getType(IObj.class);
    final static Method withMetaMethod =
        Method.getMethod(

```

```

    "clojure.lang.IObj withMeta(clojure.lang.IPersistentMap)");}

public MetaExpr(Expr expr, Expr meta){
    this.expr = expr;
    this.meta = meta;
}

public Object eval() throws Exception{
    return ((IObj) expr.eval()).withMeta((IPersistentMap) meta.eval());
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    expr.emit(C.EXPRESSION, objx, gen);
    gen.checkCast(IOBJ_TYPE);
    meta.emit(C.EXPRESSION, objx, gen);
    gen.checkCast(IPERSISTENTMAP_TYPE);
    gen.invokeInterface(IOBJ_TYPE, withMetaMethod);
    if(context == C.STATEMENT)
    {
        gen.pop();
    }
}

public boolean hasJavaClass() throws Exception{
    return expr.hasJavaClass();
}

public Class getJavaClass() throws Exception{
    return expr.getJavaClass();
}

public static class IfExpr implements Expr, MaybePrimitiveExpr{
    public final Expr testExpr;
    public final Expr thenExpr;
    public final Expr elseExpr;
    public final int line;

    public IfExpr(int line, Expr testExpr, Expr thenExpr, Expr elseExpr){
        this.testExpr = testExpr;
        this.thenExpr = thenExpr;
        this.elseExpr = elseExpr;
        this.line = line;
    }

    public Object eval() throws Exception{
        Object t = testExpr.eval();
        if(t != null && t != Boolean.FALSE)

```

```

        return thenExpr.eval();
        return elseExpr.eval();
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        doEmit(context, objx, gen, false);
    }

    public void emitUnboxed(C context,
                           ObjExpr objx,
                           GeneratorAdapter gen){
        doEmit(context, objx, gen, true);
    }

    public void doEmit(C context,
                       ObjExpr objx,
                       GeneratorAdapter gen,
                       boolean emitUnboxed){
        Label nullLabel = gen.newLabel();
        Label falseLabel = gen.newLabel();
        Label endLabel = gen.newLabel();

        gen.visitLineNumber(line, gen.mark());

        try
        {
            if(maybePrimitiveType(testExpr) == boolean.class)
            {
                ((MaybePrimitiveExpr) testExpr)
                    .emitUnboxed(C.EXPRESSION, objx, gen);
                gen.ifZCmp(gen.EQ, falseLabel);
            }
            else
            {
                testExpr.emit(C.EXPRESSION, objx, gen);
                gen.dup();
                gen.ifNull(nullLabel);
                gen.getStatic(BOOLEAN_OBJECT_TYPE, "FALSE",
                             BOOLEAN_OBJECT_TYPE);
                gen.visitJumpInsn(IF_ACMPEQ, falseLabel);
            }
        }
        catch(Exception e)
        {
            throw new RuntimeException(e);
        }
        if(emitUnboxed)
            ((MaybePrimitiveExpr)thenExpr).emitUnboxed(context, objx, gen);
        else
            thenExpr.emit(context, objx, gen);
    }
}

```

```

        gen.goTo(endLabel);
        gen.mark(nullLabel);
        gen.pop();
        gen.mark(falseLabel);
        if(emitUnboxed)
            ((MaybePrimitiveExpr)elseExpr).emitUnboxed(context, objx, gen);
        else
            elseExpr.emit(context, objx, gen);
        gen.mark(endLabel);
    }

    public boolean hasJavaClass() throws Exception{
        return thenExpr.hasJavaClass()
            && elseExpr.hasJavaClass()
            &&
            (thenExpr.getJavaClass() == elseExpr.getJavaClass()
                || (thenExpr.getJavaClass() == null
                    && !elseExpr.getJavaClass().isPrimitive()
                    || (elseExpr.getJavaClass() == null
                        && !thenExpr.getJavaClass().isPrimitive())));
    }

    public boolean canEmitPrimitive(){
        try
        {
            return thenExpr instanceof MaybePrimitiveExpr
                && elseExpr instanceof MaybePrimitiveExpr
                && thenExpr.getJavaClass() == elseExpr.getJavaClass()
                && ((MaybePrimitiveExpr)thenExpr).canEmitPrimitive()
                && ((MaybePrimitiveExpr)elseExpr).canEmitPrimitive();
        }
        catch(Exception e)
        {
            return false;
        }
    }

    public Class getJavaClass() throws Exception{
        Class thenClass = thenExpr.getJavaClass();
        if(thenClass != null)
            return thenClass;
        return elseExpr.getJavaClass();
    }

    static class Parser implements IParser{
        public Expr parse(C context, Object frm) throws Exception{
            ISeq form = (ISeq) frm;
            //((if test then) or (if test then else)
            if(form.count() > 4)
                throw new Exception("Too many arguments to if");
        }
    }
}

```

```

        else if(form.count() < 3)
            throw new Exception("Too few arguments to if");
        PathNode branch =
            new PathNode(PATHTYPE.BRANCH, (PathNode) CLEAR_PATH.get());
        Expr testexpr =
            analyze(context == C.EVAL
            ? context
            : C.EXPRESSION, RT.second(form));
        Expr thenexpr, elseexpr;
        try {
            Var.pushThreadBindings(
                RT.map(CLEAR_PATH,
                    new PathNode(PATHTYPE.PATH,branch)));
            thenexpr = analyze(context, RT.third(form));
        }
        finally{
            Var.popThreadBindings();
        }
        try {
            Var.pushThreadBindings(
                RT.map(CLEAR_PATH,
                    new PathNode(PATHTYPE.PATH,branch)));
            elseexpr = analyze(context, RT.fourth(form));
        }
        finally{
            Var.popThreadBindings();
        }
        return new IfExpr((Integer) LINE.deref(),
            testexpr,
            thenexpr,
            elseexpr);
    }
}
}

static final public IPersistentMap CHAR_MAP =
    PersistentHashMap.create('-', "_",
    //                                '_',
    ':', "_COLON_",
    '+', "_PLUS_",
    '>', "_GT_",
    '<', "_LT_",
    '=', "_EQ_",
    '^', "_TILDE_",
    '!', "_BANG_",
    '@', "_CIRCA_",
    '#', "_SHARP_",
    '$', "_DOLLARSIGN_",
    '%', "_PERCENT_",
    '^', "_CARET_",

```

```

'&', "_AMPERSAND_",
'*', "_STAR_",
'|', "_BAR_",
'{', "_LBRACE_",
'}', "_RBRACE_",
'[', "_LBRACK_",
']', "_RBRACK_",
('/', "_SLASH_",
'\\', "_BSLASH_",
'? ', "_QMARK_");

static public String munge(String name){
    StringBuilder sb = new StringBuilder();
    for(char c : name.toCharArray())
    {
        String sub = (String) CHAR_MAP.getValueAt(c);
        if(sub != null)
            sb.append(sub);
        else
            sb.append(c);
    }
    return sb.toString();
}

public static class EmptyExpr implements Expr{
    public final Object coll;
    final static Type HASHMAP_TYPE =
        Type.getType(PersistentHashMap.class);
    final static Type HASHSET_TYPE =
        Type.getType(PersistentHashSet.class);
    final static Type VECTOR_TYPE =
        Type.getType(PersistentVector.class);
    final static Type LIST_TYPE =
        Type.getType(PersistentList.class);
    final static Type EMPTY_LIST_TYPE =
        Type.getType(PersistentList.EmptyList.class);

    public EmptyExpr(Object coll){
        this.coll = coll;
    }

    public Object eval() throws Exception{
        return coll;
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        if(coll instanceof IPersistentList)
            gen.getStatic(LIST_TYPE, "EMPTY", EMPTY_LIST_TYPE);
        else if(coll instanceof IPersistentVector)

```

```

        gen.getStatic(VECTOR_TYPE, "EMPTY", VECTOR_TYPE);
    else if(coll instanceof IPersistentMap)
        gen.getStatic(HASHMAP_TYPE, "EMPTY", HASHMAP_TYPE);
    else if(coll instanceof IPersistentSet)
        gen.getStatic(HASHSET_TYPE, "EMPTY", HASHSET_TYPE);
    else
        throw new UnsupportedOperationException(
            "Unknown Collection type");
    if(context == C.STATEMENT)
    {
        gen.pop();
    }
}

public boolean hasJavaClass() throws Exception{
    return true;
}

public Class getJavaClass() throws Exception{
    if(coll instanceof IPersistentList)
        return IPersistentList.class;
    else if(coll instanceof IPersistentVector)
        return IPersistentVector.class;
    else if(coll instanceof IPersistentMap)
        return IPersistentMap.class;
    else if(coll instanceof IPersistentSet)
        return IPersistentSet.class;
    else
        throw new UnsupportedOperationException(
            "Unknown Collection type");
}
}

public static class ListExpr implements Expr{
    public final IPersistentVector args;
    final static Method arrayToListMethod =
        Method.getMethod("clojure.lang.ISeq arrayToList(Object[])");

    public ListExpr(IPersistentVector args){
        this.args = args;
    }

    public Object eval() throws Exception{
        IPersistentVector ret = PersistentVector.EMPTY;
        for(int i = 0; i < args.count(); i++)
            ret =
                (IPersistentVector) ret.cons(((Expr) args.nth(i)).eval());
        return ret.seq();
    }
}

```

```
public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    MethodExpr.emitArgsAsArray(args, objx, gen);
    gen.invokeStatic(RT_TYPE, arrayToListMethod);
    if(context == C.STATEMENT)
        gen.pop();
}

public boolean hasJavaClass() throws Exception{
    return true;
}

public Class getJavaClass() throws Exception{
    return IPersistentList.class;
}

}

public static class MapExpr implements Expr{
    public final IPersistentVector keyvals;
    final static Method mapMethod =
        Method.getMethod("clojure.lang.IPersistentMap map(Object[])");

    public MapExpr(IPersistentVector keyvals){
        this.keyvals = keyvals;
    }

    public Object eval() throws Exception{
        Object[] ret = new Object[keyvals.count()];
        for(int i = 0; i < keyvals.count(); i++)
            ret[i] = ((Expr) keyvals.nth(i)).eval();
        return RT.map(ret);
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        MethodExpr.emitArgsAsArray(keyvals, objx, gen);
        gen.invokeStatic(RT_TYPE, mapMethod);
        if(context == C.STATEMENT)
            gen.pop();
    }

    public boolean hasJavaClass() throws Exception{
        return true;
    }

    public Class getJavaClass() throws Exception{
        return IPersistentMap.class;
    }
}
```

```

static public Expr parse(C context, IPersistentMap form)
throws Exception{
    IPersistentVector keyvals = PersistentVector.EMPTY;
    boolean constant = true;
    for(ISeq s = RT.seq(form); s != null; s = s.next())
    {
        IMapEntry e = (IMapEntry) s.first();
        Expr k =
            analyze(context == C.EVAL
                ? context
                : C.EXPRESSION, e.key());
        Expr v =
            analyze(context == C.EVAL
                ? context
                : C.EXPRESSION, e.val());
        keyvals = (IPersistentVector) keyvals.cons(k);
        keyvals = (IPersistentVector) keyvals.cons(v);
        if(!(k instanceof LiteralExpr && v instanceof LiteralExpr))
            constant = false;
    }

    Expr ret = new MapExpr(keyvals);
    if(form instanceof IObj && ((IObj) form).meta() != null)
        return new MetaExpr(ret, MapExpr
            .parse(context == C.EVAL
                ? context
                : C.EXPRESSION, ((IObj) form).meta()));
    else if(constant)
    {
        IPersistentMap m = PersistentHashMap.EMPTY;
        for(int i=0;i<keyvals.length();i+= 2)
        {
            m = m.assoc(((LiteralExpr)keyvals.nth(i)).val(),
                        ((LiteralExpr)keyvals.nth(i+1)).val());
        }
        // System.err.println("Constant: " + m);
        return new ConstantExpr(m);
    }
    else
        return ret;
}
}

public static class SetExpr implements Expr{
    public final IPersistentVector keys;
    final static Method setMethod =
        Method.getMethod("clojure.lang.IPersistentSet set(Object[])");
}

```

```

public SetExpr(IPersistentVector keys){
    this.keys = keys;
}

public Object eval() throws Exception{
    Object[] ret = new Object[keys.count()];
    for(int i = 0; i < keys.count(); i++)
        ret[i] = ((Expr) keys.nth(i)).eval();
    return RT.set(ret);
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    MethodExpr.emitArgsAsArray(keys, objx, gen);
    gen.invokeStatic(RT_TYPE, setMethod);
    if(context == C.STATEMENT)
        gen.pop();
}

public boolean hasJavaClass() throws Exception{
    return true;
}

public Class getJavaClass() throws Exception{
    return IPersistentSet.class;
}

static public Expr parse(C context, IPersistentSet form)
throws Exception{
    IPersistentVector keys = PersistentVector.EMPTY;
    boolean constant = true;

    for(ISeq s = RT.seq(form); s != null; s = s.next())
    {
        Object e = s.first();
        Expr expr = analyze(context == C.EVAL
            ? context
            : C.EXPRESSION, e);
        keys = (IPersistentVector) keys.cons(expr);
        if(!(expr instanceof LiteralExpr))
            constant = false;
    }
    Expr ret = new SetExpr(keys);
    if(form instanceof IObj && ((IObj) form).meta() != null)
        return new MetaExpr(ret, MapExpr
            .parse(context == C.EVAL
            ? context
            : C.EXPRESSION, ((IObj) form).meta()));
    else if(constant)
    {
}

```

```

IPersistentSet set = PersistentHashSet.EMPTY;
for(int i=0;i<keys.count();i++)
{
    LiteralExpr ve = (LiteralExpr)keys.nth(i);
    set = (IPersistentSet)set.cons(ve.val());
}
//      System.err.println("Constant: " + set);
return new ConstantExpr(set);
}
else
    return ret;
}
}

public static class VectorExpr implements Expr{
    public final IPersistentVector args;
    final static Method vectorMethod =
        Method.getMethod(
            "clojure.lang.IPersistentVector vector(Object[])");

    public VectorExpr(IPersistentVector args){
        this.args = args;
    }

    public Object eval() throws Exception{
        IPersistentVector ret = PersistentVector.EMPTY;
        for(int i = 0; i < args.count(); i++)
            ret =
                (IPersistentVector) ret.cons(((Expr) args.nth(i)).eval());
        return ret;
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        MethodExpr.emitArgsAsArray(args, objx, gen);
        gen.invokeStatic(RT_TYPE, vectorMethod);
        if(context == C.STATEMENT)
            gen.pop();
    }

    public boolean hasJavaClass() throws Exception{
        return true;
    }

    public Class getJavaClass() throws Exception{
        return IPersistentVector.class;
    }

    static public Expr parse(C context, IPersistentVector form)
        throws Exception{

```

```

        boolean constant = true;

        IPersistentVector args = PersistentVector.EMPTY;
        for(int i = 0; i < form.count(); i++)
        {
            Expr v = analyze(context == C.EVAL
                ? context
                : C.EXPRESSION, form.nth(i));
            args = (IPersistentVector) args.cons(v);
            if(!(v instanceof LiteralExpr))
                constant = false;
        }
        Expr ret = new VectorExpr(args);
        if(form instanceof IObj && ((IObj) form).meta() != null)
            return new MetaExpr(ret, MapExpr
                .parse(context == C.EVAL
                    ? context
                    : C.EXPRESSION, ((IObj) form).meta()));
        else if (constant)
        {
            PersistentVector rv = PersistentVector.EMPTY;
            for(int i = 0;i<args.count();i++)
            {
                LiteralExpr ve = (LiteralExpr)args.nth(i);
                rv = rv.cons(ve.val());
            }
            // System.err.println("Constant: " + rv);
            return new ConstantExpr(rv);
        }
        else
            return ret;
    }

    static class KeywordInvokeExpr implements Expr{
        public final KeywordExpr kw;
        public final Object tag;
        public final Expr target;
        public final int line;
        public final int siteIndex;
        public final String source;
        static Type ILOOKUP_TYPE = Type.getType(ILookup.class);

        public KeywordInvokeExpr(String source, int line, Symbol tag,
                               KeywordExpr kw, Expr target){
            this.source = source;
            this.kw = kw;
            this.target = target;
            this.line = line;
        }
    }
}

```

```

        this.tag = tag;
        this.siteIndex = registerKeywordCallsite(kw.k);
    }

    public Object eval() throws Exception{
        try
        {
            return kw.k.invoke(target.eval());
        }
        catch(Throwable e)
        {
            if(!(e instanceof CompilerException))
                throw new CompilerException(source, line, e);
            else
                throw (CompilerException) e;
        }
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        Label endLabel = gen.newLabel();
        Label faultLabel = gen.newLabel();

        gen.visitLineNumber(line, gen.mark());
        gen.getStatic(objx.objtype,
                     objx.thunkNameStatic(siteIndex),
                     ObjExpr.ILOOKUP_THUNK_TYPE);
        gen.dup(); //thunk, thunk
        target.emit(C.EXPRESSION, objx, gen); //thunk,thunk,target
        gen.dupX2(); //target,thunk,thunk,target
        gen.invokeInterface(ObjExpr.ILOOKUP_THUNK_TYPE,
                           Method.getMethod("Object get(Object)"));
                           //target,thunk,result
        gen.dupX2(); //result,target,thunk,result
        gen.visitJumpInsn(IF_ACMPEQ, faultLabel); //result,target
        gen.pop(); //result
        gen.goTo(endLabel);

        gen.mark(faultLabel); //result,target
        gen.swap(); //target,result
        gen.pop(); //target
        gen.dup(); //target,target
        gen.getStatic(objx.objtype,
                     objx.siteNameStatic(siteIndex),
                     ObjExpr.KEYWORD_LOOKUPSITE_TYPE);
                     //target,target,site
        gen.swap(); //target,site,target
        gen.invokeInterface(ObjExpr.ILOOKUP_SITE_TYPE,
                           Method.getMethod("closure.lang.ILookupThunk fault(Object)"));
                           //target,new-thunk
        gen.dup(); //target,new-thunk,new-thunk
    }
}

```

```

        gen.putStatic(objx.objtype,
                      objx.thunkNameStatic(siteIndex),
                      ObjExpr.ILOOKUP_THUNK_TYPE); //target,new-thunk
        gen.swap(); //new-thunk,target
        gen.invokeInterface(ObjExpr.ILOOKUP_THUNK_TYPE,
                           Method.getMethod("Object get(Object)"));
                           //result

        gen.mark(endLabel);
        if(context == C.STATEMENT)
            gen.pop();
    }

    public boolean hasJavaClass() throws Exception{
        return tag != null;
    }

    public Class getJavaClass() throws Exception{
        return HostExpr.tagToClass(tag);
    }

}

//static class KeywordSiteInvokeExpr implements Expr{
//    public final Expr site;
//    public final Object tag;
//    public final Expr target;
//    public final int line;
//    public final String source;
//
//    public KeywordSiteInvokeExpr(String source, int line,
//                                Symbol tag, Expr site, Expr target){
//        this.source = source;
//        this.site = site;
//        this.target = target;
//        this.line = line;
//        this.tag = tag;
//    }
//
//    public Object eval() throws Exception{
//        try
//        {
//            KeywordCallSite s = (KeywordCallSite) site.eval();
//            return s.thunk.invoke(s,target.eval());
//        }
//        catch(Throwable e)
//        {
//            if(!(e instanceof CompilerException))
//                throw new CompilerException(source, line, e);
//            else
//                throw (CompilerException) e;
//        }
//    }
//}
```

```

//           }
//       }
//
//       public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
//           gen.visitLineNumber(line, gen.mark());
//           site.emit(C.EXPRESSION, objx, gen);
//           gen.dup();
//           gen.getField(Type.getType(KeywordCallSite.class),
//                       "thunk", IFN_TYPE);
//           gen.swap();
//           target.emit(C.EXPRESSION, objx, gen);
//
//           gen.invokeInterface(IFN_TYPE,
//                               new Method("invoke", OBJECT_TYPE, ARG_TYPES[2]));
//           if(context == C.STATEMENT)
//               gen.pop();
//       }
//
//       public boolean hasJavaClass() throws Exception{
//           return tag != null;
//       }
//
//       public Class getJavaClass() throws Exception{
//           return HostExpr.tagToClass(tag);
//       }
//
//   }
}

public static class InstanceOfExpr
    implements Expr, MaybePrimitiveExpr{
    Expr expr;
    Class c;

    public InstanceOfExpr(Class c, Expr expr){
        this.expr = expr;
        this.c = c;
    }

    public Object eval() throws Exception{
        if(c.isInstance(expr.eval()))
            return RT.T;
        return RT.F;
    }

    public boolean canEmitPrimitive(){
        return true;
    }

    public void emitUnboxed(C context,
                           ObjExpr objx,

```

```

        GeneratorAdapter gen){
    expr.emit(C.EXPRESSION, objx, gen);
    gen.newInstance0f(getType(c));
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    emitUnboxed(context,objx,gen);
    HostExpr.emitBoxReturn(objx,gen,Boolean.TYPE);
    if(context == C.STATEMENT)
        gen.pop();
}

public boolean hasJavaClass() throws Exception{
    return true;
}

public Class getJavaClass() throws Exception{
    return Boolean.TYPE;
}

}

static class StaticInvokeExpr implements Expr, MaybePrimitiveExpr{
    public final Type target;
    public final Class retClass;
    public final Class[] paramclasses;
    public final Type[] paramtypes;
    public final IPersistentVector args;
    public final boolean variadic;
    public final Symbol tag;

    StaticInvokeExpr(Type target,
                     Class retClass,
                     Class[] paramclasses,
                     Type[] paramtypes,
                     boolean variadic,
                     IPersistentVector args,Symbol tag){
        this.target = target;
        this.retClass = retClass;
        this.paramclasses = paramclasses;
        this.paramtypes = paramtypes;
        this.args = args;
        this.variadic = variadic;
        this.tag = tag;
    }

    public Object eval() throws Exception{
        throw new UnsupportedOperationException(
            "Can't eval StaticInvokeExpr");
    }
}

```

```

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    emitUnboxed(context, objx, gen);
    if(context != C.STATEMENT)
        HostExpr.emitBoxReturn(objx,gen,retClass);
    if(context == C.STATEMENT)
    {
        if(retClass == long.class || retClass == double.class)
            gen.pop2();
        else
            gen.pop();
    }
}

public boolean hasJavaClass() throws Exception{
    return true;
}

public Class getJavaClass() throws Exception{
    return tag != null ? HostExpr.tagToClass(tag) : retClass;
}

public boolean canEmitPrimitive(){
    return retClass.isPrimitive();
}

public void emitUnboxed(C context,
                       ObjExpr objx,
                       GeneratorAdapter gen){
    Method ms =
        new Method("invokeStatic", getReturnType(), paramtypes);
    if(variadic)
    {
        for(int i = 0; i < paramclasses.length - 1; i++)
        {
            Expr e = (Expr) args.nth(i);
            try
            {
                if(maybePrimitiveType(e) == paramclasses[i])
                {
                    ((MaybePrimitiveExpr) e)
                        .emitUnboxed(C.EXPRESSION, objx, gen);
                }
                else
                {
                    e.emit(C.EXPRESSION, objx, gen);
                    HostExpr.emitUnboxArg(objx, gen,
                                         paramclasses[i]);
                }
            }
        }
    }
}

```

```

        catch(Exception ex)
        {
            throw new RuntimeException(ex);
        }
    }

IPersistentVector restArgs =
    RT.subvec(args,paramclasses.length - 1,args.count());
MethodExpr.emitArgsAsArray(restArgs,objx,gen);
gen.invokeStatic(Type.getType(ArraySeq.class),
    Method.getMethod(
        "clojure.lang.ArraySeq create(Object[])"));
}
else
    MethodExpr.emitTypedArgs(objx, gen, paramclasses, args);

gen.invokeStatic(target, ms);
}

private Type getReturnType(){
    return Type.getType(retClass);
}

public static Expr parse(Var v, ISeq args, Symbol tag)
throws Exception{
    IPersistentCollection paramlists =
        (IPersistentCollection) RT.get(v.meta(), arglistsKey);
    if(paramlists == null)
        throw new IllegalStateException(
            "Can't call static fn with no arglists: " + v);
    IPersistentVector paramlist = null;
    int argcnt = RT.count(args);
    boolean variadic = false;
    for(ISeq aseq =
        RT.seq(paramlists); aseq != null; aseq = aseq.next())
    {
        if(!(aseq.first() instanceof IPersistentVector))
            throw new IllegalStateException(
                "Expected vector arglist, had: " + aseq.first());
        IPersistentVector alist = (IPersistentVector) aseq.first();
        if(alist.count() > 1
            && alist.nth(alist.count() - 2).equals(_AMP_))
        {
            if(argcnt >= alist.count() - 2)
            {
                paramlist = alist;
                variadic = true;
            }
        }
        else if(alist.count() == argcnt)
        {

```

```

        paramlist = alist;
        variadic = false;
        break;
    }
}

if(paramlist == null)
    throw new IllegalArgumentException(
        "Invalid arity - can't call: " + v + " with " +
        argcount + " args");

Class retClass = tagClass(tagOf(paramlist));

ArrayList<Class> paramClasses = new ArrayList();
ArrayList<Type> paramTypes = new ArrayList();

if(variadic)
{
    for(int i = 0; i < paramlist.count()-2;i++)
    {
        Class pc = tagClass(tagOf(paramlist.nth(i)));
        paramClasses.add(pc);
        paramTypes.add(Type.getType(pc));
    }
    paramClasses.add(ISeq.class);
    paramTypes.add(Type.getType(ISeq.class));
}
else
{
    for(int i = 0; i < argcount;i++)
    {
        Class pc = tagClass(tagOf(paramlist.nth(i)));
        paramClasses.add(pc);
        paramTypes.add(Type.getType(pc));
    }
}

String cname =
    v.ns.name.name.replace('.', '/').replace('-', '_') +
    $" + munge(v.sym.name);
Type target = Type.getObjectType(cname);

PersistentVector argv = PersistentVector.EMPTY;
for(ISeq s = RT.seq(args); s != null; s = s.next())
    argv = argv.cons(analyze(C.EXPRESSION, s.first()));

return
new StaticInvokeExpr(target,
                     retClass,
                     paramClasses

```



```

        protocolOn.getName() +
        " found for function: " +
        fvar.sym + " of protocol: " +
        pvar.sym +
    " (The protocol method may have been defined before and removed.));
    }
    String mname = munge(mmapVal.sym.toString());
    List methods =
        Reflector.getMethods(protocolOn, args.count() - 1,
                             mname, false);
    if(methods.size() != 1)
        throw new IllegalArgumentException(
            "No single method: " + mname +
            " of interface: " +
            protocolOn.getName() +
            " found for function: " + fvar.sym +
            " of protocol: " + pvar.sym);
    this.onMethod =
        (java.lang.reflect.Method) methods.get(0);
    }
}
this.tag = tag != null
    ? tag
    : (fexpr instanceof VarExpr
        ? ((VarExpr) fexpr).tag
        : null);
}

public Object eval() throws Exception{
    try
    {
        IFn fn = (IFn) fexpr.eval();
        PersistentVector argvs = PersistentVector.EMPTY;
        for(int i = 0; i < args.count(); i++)
            argvs = argvs.cons(((Expr) args.nth(i)).eval());
        return fn.applyTo(RT.seq(argvs));
    }
    catch(Throwable e)
    {
        if(!(e instanceof CompilerException))
            throw new CompilerException(source, line, e);
        else
            throw (CompilerException) e;
    }
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    gen.visitLineNumber(line, gen.mark());
    if(isProtocol)

```

```

    {
        emitProto(context,objx,gen);
    }

    else
    {
        fexpr.emit(C.EXPRESSION, objx, gen);
        gen.checkCast(IFN_TYPE);
        emitArgsAndCall(0, context,objx,gen);
    }
    if(context == C.STATEMENT)
        gen.pop();
}

public void emitProto(C context, ObjExpr objx, GeneratorAdapter gen){
    Label onLabel = gen.newLabel();
    Label callLabel = gen.newLabel();
    Label endLabel = gen.newLabel();

    Var v = ((VarExpr)fexpr).var;

    Expr e = (Expr) args.nth(0);
    e.emit(C.EXPRESSION, objx, gen);
    gen.dup(); //target, target
    gen.invokeStatic(UTIL_TYPE,
                    Method.getMethod("Class classOf(Object)"));
                    //target,class
    gen.loadThis();
    gen.getField(objx.objtype,
                objx.cachedClassName(siteIndex),
                CLASS_TYPE); //target,class,cached-class
    gen.visitJumpInsn(IF_ACMPEQ, callLabel); //target
    if(protocolOn != null)
    {
        gen.dup(); //target, target
        gen.instanceOf(Type.getType(protocolOn));
        gen.ifZCmp(GeneratorAdapter.NE, onLabel);
    }

    gen.dup(); //target, target
    gen.invokeStatic(UTIL_TYPE,
                    Method.getMethod("Class classOf(Object)"));
                    //target,class
    gen.loadThis();
    gen.swap();
    gen.putField(objx.objtype,
                objx.cachedClassName(siteIndex),
                CLASS_TYPE); //target

    gen.mark(callLabel); //target
}

```

```

objx.emitVar(gen, v);
gen.invokeVirtual(VAR_TYPE,
                  Method.getMethod("Object getRawRoot()"));
                  //target, proto-fn
gen.swap();
emitArgsAndCall(1, context,objx,gen);
gen.goTo(endLabel);

gen.mark(onLabel); //target
if(protocolOn != null)
{
    MethodExpr.emitTypedArgs(objx, gen,
                             onMethod.getParameterTypes(),
                             RT.subvec(args,1,args.count()));
    if(context == C.RETURN)
    {
        ObjMethod method = (ObjMethod) METHOD.deref();
        method.emitClearLocals(gen);
    }
    Method m = new Method(onMethod.getName(),
                          Type.getReturnType(onMethod),
                          Type.getArgumentTypes(onMethod));
    gen.invokeInterface(Type.getType(protocolOn), m);
    HostExpr.emitBoxReturn(objx, gen, onMethod.getReturnType());
}
gen.mark(endLabel);
}

void emitArgsAndCall(int firstArgToEmit,
                     C context,
                     ObjExpr objx,
                     GeneratorAdapter gen){
for(int i = firstArgToEmit;
    i < Math.min(MAX_POSITIONAL_ARITY, args.count());
    i++)
{
    Expr e = (Expr) args.nth(i);
    e.emit(C.EXPRESSION, objx, gen);
}
if(args.count() > MAX_POSITIONAL_ARITY)
{
    PersistentVector restArgs = PersistentVector.EMPTY;
    for(int i = MAX_POSITIONAL_ARITY; i < args.count(); i++)
    {
        restArgs = restArgs.cons(args.nth(i));
    }
    MethodExpr.emitArgsAsArray(restArgs, objx, gen);
}

if(context == C.RETURN)

```

```

    {
        ObjMethod method = (ObjMethod) METHOD.deref();
        method.emitClearLocals(gen);
    }

    gen.invokeInterface(
        IFN_TYPE,
        new Method("invoke",
            OBJECT_TYPE,
            ARG_TYPES[Math.min(MAX_POSITIONAL_ARITY + 1,
                args.count()))]);
}

public boolean hasJavaClass() throws Exception{
    return tag != null;
}

public Class getJavaClass() throws Exception{
    return HostExpr.tagToClass(tag);
}

static public Expr parse(C context, ISeq form) throws Exception{
    if(context != C.EVAL)
        context = C.EXPRESSION;
    Expr fexpr = analyze(context, form.first());
    if(fexpr instanceof VarExpr &&
        ((VarExpr)fexpr).var.equals(INSTANCE))
    {
        if(RT.second(form) instanceof Symbol)
        {
            Class c = HostExpr.maybeClass(RT.second(form),false);
            if(c != null)
                return
                    new InstanceOfExpr(c,
                        analyze(context, RT.third(form)));
        }
    }

    // if(fexpr instanceof VarExpr && context != C.EVAL)
    // {
    //     Var v = ((VarExpr)fexpr).var;
    //     if(RT.booleanCast(RT.get(RT.meta(v),staticKey)))
    //     {
    //         return StaticInvokeExpr.parse(v, RT.next(form),
    //             tagOf(form));
    //     }
    // }

    if(fexpr instanceof VarExpr && context != C.EVAL)
    {

```

```

Var v = ((VarExpr)fexpr).var;
Object arglists = RT.get(RT.meta(v), arglistsKey);
int arity = RT.count(form.next());
for(ISeq s = RT.seq(arglists); s != null; s = s.next())
{
    IPersistentVector args = (IPersistentVector) s.first();
    if(args.count() == arity)
    {
        String primc = FnMethod.primInterface(args);
        if(primc != null)
            return
                analyze(context,
                    RT.listStar(Symbol.intern(".invokePrim"),
                        ((Symbol) form.first())
                            .withMeta(
                                RT.map(
                                    RT.TAG_KEY, Symbol.intern(primc))),
                        form.next()));
        break;
    }
}
}

if(fexpr instanceof KeywordExpr &&
    RT.count(form) == 2 &&
    KEYWORD_CALLSITES.isBound())
{
    fexpr =
    new ConstantExpr(
        new KeywordCallSite(((KeywordExpr)fexpr).k));
    Expr target = analyze(context, RT.second(form));
    return
        new KeywordInvokeExpr((String) SOURCE.deref(),
            (Integer) LINE.deref(), tagOf(form),
            (KeywordExpr) fexpr, target);
}
PersistentVector args = PersistentVector.EMPTY;
for(ISeq s = RT.seq(form.next()); s != null; s = s.next())
{
    args = args.cons(analyze(context, s.first()));
}
// if(args.count() > MAX_POSITIONAL_ARITY)
//     throw new IllegalArgumentException(
//         String.format(
//             "No more than %d args supported", MAX_POSITIONAL_ARITY));
// return new InvokeExpr((String) SOURCE.deref(),
//     (Integer) LINE.deref(),
//     tagOf(form),
//     fexpr,

```

```

                args);
        }
    }

    static class SourceDebugExtensionAttribute extends Attribute{
        public SourceDebugExtensionAttribute(){
            super("SourceDebugExtension");
        }

        void writeSMAP(ClassWriter cw, String smap){
            ByteVector bv = write(cw, null, -1, -1, -1);
            bv.putUTF8(smap);
        }
    }

    static public class FnExpr extends ObjExpr{
        final static Type aFnType = Type.getType(AFunction.class);
        final static Type restFnType = Type.getType(RestFn.class);
        //if there is a variadic overload (there can only be one)
        // it is stored here
        FnMethod variadicMethod = null;
        IPersistentCollection methods;
        //    String superName = null;

        public FnExpr(Object tag){
            super(tag);
        }

        public boolean hasJavaClass() throws Exception{
            return true;
        }

        public Class getJavaClass() throws Exception{
            return AFunction.class;
        }

        protected void emitMethods(ClassVisitor cv){
            //override of invoke/doInvoke for each method
            for(ISeq s = RT.seq(methods); s != null; s = s.next())
            {
                ObjMethod method = (ObjMethod) s.first();
                method.emit(this, cv);
            }

            if(isVariadic())
            {
                GeneratorAdapter gen =
                    new GeneratorAdapter(
                        ACC_PUBLIC,
                        Method.getMethod("int getRequiredArity()"),

```

```

        null,
        null,
        cv);
    gen.visitCode();
    gen.push(variadicMethod.reqParms.count());
    gen.returnValue();
    gen.endMethod();
}
}

static Expr parse(C context, ISeq form, String name)
throws Exception{
    ISeq origForm = form;
    FnExpr fn = new FnExpr(tagOf(form));
    fn.src = form;
    ObjMethod enclosingMethod = (ObjMethod) METHOD.deref();
    if(((IMeta) form.first()).meta() != null)
    {
        fn.onceOnly =
            RT.booleanCast(RT.get(RT.meta(form.first())),
                           Keyword.intern(null, "once"));
    }
    //fn.superName = (String) RT.get(RT.meta(form.first())),
    //                           Keyword.intern(null,
    //                                         "super-name"));
    //}
    //fn.thisName = name;
    String basename = enclosingMethod != null ?
        (enclosingMethod.objx.name + "$")
        : //clojure.fns."
        (munge(currentNS().name.name) + "$");
    if(RT.second(form) instanceof Symbol)
        name = ((Symbol) RT.second(form)).name;
    String simpleName = name != null ?
        (munge(name).replace(".", "_DOT_")
         + (enclosingMethod != null
            ? "__" + RT.nextID()
            : ""))
        : ("fn"
         + "__" + RT.nextID());
    fn.name = basename + simpleName;
    fn.internalName = fn.name.replace('.', '/');
    fn.objtype = Type.getObjectType(fn.internalName);
    ArrayList<String> prims = new ArrayList();
    try
    {
        Var.pushThreadBindings(
            RT.map(CONSTANTS, PersistentVector.EMPTY,
                   CONSTANT_IDS, new IdentityHashMap(),
                   KEYWORDS, PersistentHashMap.EMPTY,
                   VARS, PersistentHashMap.EMPTY),

```

```

KEYWORD_CALLSITES, PersistentVector.EMPTY,
PROTOCOL_CALLSITES, PersistentVector.EMPTY,
VAR_CALLSITES, emptyVarCallSites(),
NO_RECUR, null
));

//arglist might be preceded by symbol naming this fn
if(RT.second(form) instanceof Symbol)
{
    Symbol nm = (Symbol) RT.second(form);
    fn.thisName = nm.name;
    //RT.booleanCast(RT.get(nm.meta(), staticKey));
    fn.isStatic = false;
    form = RT.cons(FN, RT.next(RT.next(form)));
}

//now (fn [args] body...) or
//      (fn ([args] body...) ([args2] body2...) ...)
//turn former into latter
if(RT.second(form) instanceof IPersistentVector)
    form = RT.list(FN, RT.next(form));
fn.line = (Integer) LINE.deref();
FnMethod[] methodArray =
    new FnMethod[MAX_POSITIONAL_ARITY + 1];
FnMethod variadicMethod = null;
for(ISeq s = RT.next(form); s != null; s = RT.next(s))
{
    FnMethod f =
        FnMethod.parse(fn, (ISeq) RT.first(s), fn.isStatic);
    if(f.isVariadic())
    {
        if(variadicMethod == null)
            variadicMethod = f;
        else
            throw new Exception(
                "Can't have more than 1 variadic overload");
    }
    else if(methodArray[f.reqParms.count()] == null)
        methodArray[f.reqParms.count()] = f;
    else
        throw new Exception(
            "Can't have 2 overloads with same arity");
    if(f.prim != null)
        prims.add(f.prim);
}
if(variadicMethod != null)
{
    for(int i = variadicMethod.reqParms.count() + 1;
        i <= MAX_POSITIONAL_ARITY;
        i++)

```

```

        if(methodArray[i] != null)
            throw new Exception(
                "Can't have fixed arity function with more params "+
                "than variadic function");
    }

    if(fn.isStatic && fn.closes.count() > 0)
        throw new IllegalArgumentException(
            "static fns can't be closures");
    IPersistentCollection methods = null;
    for(int i = 0; i < methodArray.length; i++)
        if(methodArray[i] != null)
            methods = RT.conj(methods, methodArray[i]);
    if(variadicMethod != null)
        methods = RT.conj(methods, variadicMethod);

    fn.methods = methods;
    fn.variadicMethod = variadicMethod;
    fn.keywords = (IPersistentMap) KEYWORDS.deref();
    fn.vars = (IPersistentMap) VARS.deref();
    fn.constants = (PersistentVector) CONSTANTS.deref();
    fn.keywordCallsites =
        (IPersistentVector) KEYWORD_CALLSITES.deref();
    fn.protocolCallsites =
        (IPersistentVector) PROTOCOL_CALLSITES.deref();
    fn.varCallsites = (IPersistentSet) VAR_CALLSITES.deref();

    fn.constantsID = RT.nextID();
    // DynamicClassLoader loader =
    //     (DynamicClassLoader) LOADER.get();
    // loader.registerConstants(fn.constantsID,
    //                         fn.constants.toArray());
    //
}
finally
{
    Var.popThreadBindings();
}
fn.compile(fn.isVariadic()
    ? "clojure/lang/RestFn"
    : "clojure/lang/AFunction",
    (prims.size() == 0)?
        null
        :prims.toArray(new String[prims.size()]),
    fn.onceOnly);
fn.getCompiledClass();

if(origForm instanceof IObj && ((IObj) origForm).meta() != null)
    return new MetaExpr(fn, MapExpr
        .parse(context == C.EVAL
            ? context

```

```

        : C.EXPRESSION, ((IObj) origForm).meta()));
    else
        return fn;
}

public final ObjMethod variadicMethod(){
    return variadicMethod;
}

boolean isVariadic(){
    return variadicMethod != null;
}

public final IPersistentCollection methods(){
    return methods;
}
}

static public class ObjExpr implements Expr{
    static final String CONST_PREFIX = "const__";
    String name;
    //String simpleName;
    String internalName;
    String thisName;
    Type objtype;
    public final Object tag;
    //localbinding->itself
    IPersistentMap closes = PersistentHashMap.EMPTY;
    //localbndingexprs
    IPersistentVector closesExprs = PersistentVector.EMPTY;
    //symbols
    IPersistentSet volatiles = PersistentHashSet.EMPTY;

    //symbol->lb
    IPersistentMap fields = null;

    //Keyword->KeywordExpr
    IPersistentMap keywords = PersistentHashMap.EMPTY;
    IPersistentMap vars = PersistentHashMap.EMPTY;
    Class compiledClass;
    int line;
    PersistentVector constants;
    int constantsID;
    int altCtorDrops = 0;

    IPersistentVector keywordCallsites;
    IPersistentVector protocolCallsites;
    IPersistentSet varCallsites;
    boolean onceOnly = false;
}

```

```
Object src;

final static Method voidctor = Method.getMethod("void <init>()");
protected IPersistentMap classMeta;
protected boolean isStatic;

public final String name(){
    return name;
}

//    public final String simpleName(){
//        return simpleName;
//    }

public final String internalName(){
    return internalName;
}

public final String thisName(){
    return thisName;
}

public final Type objtype(){
    return objtype;
}

public final IPersistentMap closes(){
    return closes;
}

public final IPersistentMap keywords(){
    return keywords;
}

public final IPersistentMap vars(){
    return vars;
}

public final Class compiledClass(){
    return compiledClass;
}

public final int line(){
    return line;
}

public final PersistentVector constants(){
    return constants;
}
```

```

public final int constantsID(){
    return constantsID;
}

final static Method kwintern =
    Method.getMethod("clojure.lang.Keyword intern(String, String)");
final static Method symintern =
    Method.getMethod("clojure.lang.Symbol intern(String)");
final static Method varintern =
    Method.getMethod(
"clojure.lang.Var intern(clojure.lang.Symbol, clojure.lang.Symbol)");

final static Type DYNAMIC_CLASSLOADER_TYPE =
    Type.getType(DynamicClassLoader.class);
final static Method getClassMethod =
    Method.getMethod("Class getClass()");
final static Method getClassLoaderMethod =
    Method.getMethod("ClassLoader getClassLoader()");
final static Method getConstantsMethod =
    Method.getMethod("Object[] getConstants(int)");
final static Method readStringMethod =
    Method.getMethod("Object readString(String)");

final static Type ILOOKUP_SITE_TYPE =
    Type.getType(ILookupSite.class);
final static Type ILOOKUP_THUNK_TYPE =
    Type.getType(ILookupThunk.class);
final static Type KEYWORD_LOOKUPSITE_TYPE =
    Type.getType(KeywordLookupSite.class);

private DynamicClassLoader loader;
private byte[] bytecode;

public ObjExpr(Object tag){
    this.tag = tag;
}

static String trimGenID(String name){
    int i = name.lastIndexOf("__");
    return i== -1 ? name : name.substring(0,i);
}
}

Type[] ctorTypes(){
    IPersistentVector tv =
        isDeftype()
        ? PersistentVector.EMPTY
        : RT.vector(IPERSISTENTMAP_TYPE);
    for(ISeq s = RT.keys(closes); s != null; s = s.next())

```

```

{
    LocalBinding lb = (LocalBinding) s.first();
    if(lb.getPrimitiveType() != null)
        tv = tv.cons(Type.getType(lb.getPrimitiveType()));
    else
        tv = tv.cons(OBJECT_TYPE);
}
Type[] ret = new Type[tv.count()];
for(int i = 0; i < tv.count(); i++)
    ret[i] = (Type) tv.nth(i);
return ret;
}

void compile(String superName,
             String[] interfaceNames,
             boolean oneTimeUse)
throws Exception{
    //create bytecode for a class
    //with name current_ns.defname[$letname]+
    //anonymous fns get names fn__id
    //derived from AFn/RestFn
    ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAXS);
//    ClassWriter cw = new ClassWriter(0);
    ClassVisitor cv = cw;
//    ClassVisitor cv =
//        new TraceClassVisitor(new CheckClassAdapter(cw),
//                             new PrintWriter(System.out));
//    ClassVisitor cv =
//        new TraceClassVisitor(cw, new PrintWriter(System.out));
    cv.visit(V1_5, ACC_PUBLIC + ACC_SUPER + ACC_FINAL,
             internalName, null,superName,interfaceNames);
//            superName != null ? superName :
//            (isVariadic()
//            ? "clojure/lang/RestFn"
//            : "clojure/lang/AFunction"), null);
    String source = (String) SOURCE.deref();
    int lineBefore = (Integer) LINE_BEFORE.deref();
    int lineAfter = (Integer) LINE_AFTER.deref() + 1;

    if(source != null && SOURCE_PATH.deref() != null)
    {
        //cv.visitSource(source, null);
        String smap = "SMAP\n" +
                      ((source.lastIndexOf('.') > 0) ?
                           source.substring(0, source.lastIndexOf('.')))
                           :source)
                           // : simpleName
                           + ".java\n" +
                           "Clojure\n" +
                           "*S Clojure\n" +

```

```

    "*F\n" +
    "+ 1 " + source + "\n" +
    (String) SOURCE_PATH.deref() + "\n" +
    "*L\n" +
    String.format("%d#1,%d:%d\n", lineBefore,
                  lineAfter - lineBefore, lineBefore) +
    "*E";
    cv.visitSource(source, smap);
}
addAnnotation(cv, classMeta);
//static fields for constants
for(int i = 0; i < constants.count(); i++)
{
    cv.visitField(ACC_PUBLIC + ACC_FINAL
                  + ACC_STATIC,
                  constantName(i),
                  constantType(i).getDescriptor(),
                  null, null);
}

//static fields for lookup sites
for(int i = 0; i < keywordCallsites.count(); i++)
{
    cv.visitField(ACC_FINAL
                  + ACC_STATIC,
                  siteNameStatic(i),
                  KEYWORD_LOOKUPSITE_TYPE.getDescriptor(),
                  null, null);
    cv.visitField(ACC_STATIC,
                  thunkNameStatic(i),
                  ILOOKUP_THUNK_TYPE.getDescriptor(),
                  null, null);
}

//        for(int i=0;i<varCallsites.count();i++)
//        {
//            cv.visitField(ACC_PRIVATE + ACC_STATIC + ACC_FINAL
//                          , varCallsiteName(i),
//                          IFN_TYPE.getDescriptor(), null, null);
//        }

//static init for constants, keywords and vars
GeneratorAdapter clinitgen =
    new GeneratorAdapter(ACC_PUBLIC + ACC_STATIC,
                         Method.getMethod("void <clinit> ()"),
                         null,
                         null,
                         cv);
clinitgen.visitCode();
clinitgen.visitLineNumber(line, clinitgen.mark());

```

```

if(constants.count() > 0)
{
    emitConstants(clinitgen);
}

if(keywordCallsites.count() > 0)
    emitKeywordCallsites(clinitgen);

/*
for(int i=0;i<varCallsites.count();i++)
{
    Label skipLabel = clinitgen.newLabel();
    Label endLabel = clinitgen.newLabel();
    Var var = (Var) varCallsites.nth(i);
    clinitgen.push(var.ns.name.toString());
    clinitgen.push(var.sym.toString());
    clinitgen.invokeStatic(RT_TYPE,
        Method.getMethod("clojure.lang.Var var(String, String)"));
    clinitgen.dup();
    clinitgen.invokeVirtual(VAR_TYPE,
        Method.getMethod("boolean hasRoot()"));
    clinitgen.ifZCmp(GeneratorAdapter.EQ, skipLabel);

    clinitgen.invokeVirtual(VAR_TYPE,
        Method.getMethod("Object getRoot()"));
    clinitgen.dup();
    clinitgen.instanceOf(AFUNCTION_TYPE);
    clinitgen.ifZCmp(GeneratorAdapter.EQ, skipLabel);
    clinitgen.checkCast(IFN_TYPE);
    clinitgen.putStatic(objtype, varCallsiteName(i), IFN_TYPE);
    clinitgen.goTo(endLabel);

    clinitgen.mark(skipLabel);
    clinitgen.pop();

    clinitgen.mark(endLabel);
}
*/
clinitgen.returnValue();

clinitgen.endMethod();
if(!isDeftype())
{
    cv.visitField(ACC_FINAL, "__meta",
        IPERSISTENTMAP_TYPE.getDescriptor(), null, null);
}
//instance fields for closed-overs
for(ISeq s = RT.keys(closes); s != null; s = s.next())
{

```

```

LocalBinding lb = (LocalBinding) s.first();
if(isDeftype())
{
    int access = isVolatile(lb) ? ACC_VOLATILE :
        isMutable(lb) ? 0 :
        (ACC_PUBLIC + ACC_FINAL);
    FieldVisitor fv;
    if(lb.getPrimitiveType() != null)
        fv = cv.visitField(access,
            lb.name,
            Type.getType(lb.getPrimitiveType())
                .getDescriptor(),
            null, null);
    else
        //todo - when closed-overs are fields, use more
        // specific types here and in ctor and emitLocal?
        fv = cv.visitField(access,
            lb.name,
            OBJECT_TYPE.getDescriptor(), null, null);
    addAnnotation(fv, RT.meta(lb.sym));
}
else
{
    //todo - only enable this non-private+writability
    // for letfns where we need it
    if(lb.getPrimitiveType() != null)
        cv.visitField(0 + (isVolatile(lb)
            ? ACC_VOLATILE
            : 0),
            lb.name,
            Type.getType(lb.getPrimitiveType())
                .getDescriptor(),
            null, null);
    else
        cv.visitField(0 //+ (oneTimeUse ? 0 : ACC_FINAL)
            , lb.name,
            OBJECT_TYPE.getDescriptor(), null, null);
}
}

//instance fields for callsites and thunks
for(int i=0;i<protocolCallsites.count();i++)
{
    cv.visitField(ACC_PRIVATE, cachedClassName(i),
        CLASS_TYPE.getDescriptor(), null, null);
    cv.visitField(ACC_PRIVATE, cachedProtoFnName(i),
        AFUNCTION_TYPE.getDescriptor(), null, null);
    cv.visitField(ACC_PRIVATE, cachedProtoImplName(i),
        IFN_TYPE.getDescriptor(), null, null);
}

```

```

//ctor that takes closed-overs and inits base + fields
Method m = new Method("<init>", Type.VOID_TYPE, ctorTypes());
GeneratorAdapter ctorgen = new GeneratorAdapter(ACC_PUBLIC,
                                                m,
                                                null,
                                                null,
                                                cv);
Label start = ctorgen.newLabel();
Label end = ctorgen.newLabel();
ctorgen.visitCode();
ctorgen.visitLineNumber(line, ctorgen.mark());
ctorgen.visitLabel(start);
ctorgen.loadThis();
//    if(superName != null)
//        ctorgen.invokeConstructor(Type.getObjectType(superName),
//                                  voidctor);
//    else if(isVariadic()) //RestFn ctor takes reqArity arg
//    {
//        ctorgen.push(variadicMethod.reqParms.count());
//        ctorgen.invokeConstructor(restFnType, restfnctor);
//    }
//    else
//        ctorgen.invokeConstructor(aFnType, voidctor);

//    if(vars.count() > 0)
//    {
//        ctorgen.loadThis();
//        ctorgen.getStatic(VAR_TYPE, "rev", Type.INT_TYPE);
//        ctorgen.push(-1);
//        ctorgen.visitInsn(Opcodes.IADD);
//        ctorgen.putField(objtype, "__varrev__", Type.INT_TYPE);
//    }

if(!isDeftype())
{
    ctorgen.loadThis();
    ctorgen.visitVarInsn(IPERSISTENTMAP_TYPE
                        .getOpcode(Opcodes.ILOAD), 1);
    ctorgen.putField(objtype, "__meta", IPERSISTENTMAP_TYPE);
}

int a = isDeftype()?1:2;
for(ISeq s = RT.keys(closes); s != null; s = s.next(), ++a)
{
    LocalBinding lb = (LocalBinding) s.first();
    ctorgen.loadThis();
    Class primc = lb.getPrimitiveType();
    if(primc != null)
    {

```

```

        ctorgen.visitVarInsn(
            Type.getType(primc).getOpcode(Opcodes.ILOAD), a);
        ctorgen.putField(objtype, lb.name, Type.getType(primc));
        if(primc == Long.TYPE || primc == Double.TYPE)
            ++a;
    }
    else
    {
        ctorgen.visitVarInsn(
            OBJECT_TYPE.getOpcode(Opcodes.ILOAD), a);
        ctorgen.putField(objtype, lb.name, OBJECT_TYPE);
    }
    closesExprs =
    closesExprs.cons(new LocalBindingExpr(lb, null));
}

ctorgen.visitLabel(end);

ctorgen.returnValue();

ctorgen.endMethod();

if(altCtorDrops > 0)
{
    //ctor that takes closed-overs and inits base + fields
    Type[] ctorTypes = ctorTypes();
    Type[] altCtorTypes =
        new Type[ctorTypes.length-altCtorDrops];
    for(int i=0;i<altCtorTypes.length;i++)
        altCtorTypes[i] = ctorTypes[i];
    Method alt =
        new Method("<init>", Type.VOID_TYPE, altCtorTypes);
    ctorgen = new GeneratorAdapter(ACC_PUBLIC,
                                    alt,
                                    null,
                                    null,
                                    cv);
    ctorgen.visitCode();
    ctorgen.loadThis();
    ctorgen.loadArgs();
    for(int i=0;i<altCtorDrops;i++)
        ctorgen.visitInsn(Opcodes.ACONST_NULL);

    ctorgen.invokeConstructor(objtype,
                            new Method("<init>",
                                      Type.VOID_TYPE, ctorTypes));

    ctorgen.returnValue();
    ctorgen.endMethod();
}

```

```

        }

    if(!isDeftype())
    {
        //ctor that takes closed-overs but not meta
        Type[] ctorTypes = ctorTypes();
        Type[] noMetaCtorTypes = new Type[ctorTypes.length-1];
        for(int i=1;i<ctorTypes.length;i++)
            noMetaCtorTypes[i-1] = ctorTypes[i];
        Method alt = new Method("<init>",
                               Type.VOID_TYPE,
                               noMetaCtorTypes);
        ctorgen = new GeneratorAdapter(ACC_PUBLIC,
                                       alt,
                                       null,
                                       null,
                                       cv);

        ctorgen.visitCode();
        ctorgen.loadThis();
        ctorgen.visitInsn(Opcodes.ACONST_NULL);      //null meta
        ctorgen.loadArgs();
        ctorgen.invokeConstructor(objtype,
                                 new Method("<init>",
                                           Type.VOID_TYPE, ctorTypes));

        ctorgen.returnValue();
        ctorgen.endMethod();

        //meta()
        Method meth =
            Method.getMethod("clojure.lang.IPersistentMap meta()");

        GeneratorAdapter gen = new GeneratorAdapter(ACC_PUBLIC,
                                                     meth,
                                                     null,
                                                     null,
                                                     cv);
        gen.visitCode();
        gen.loadThis();
        gen.getField(objtype,"__meta",IPERSISTENTMAP_TYPE);

        gen.returnValue();
        gen.endMethod();

        //withMeta()
        meth =
            Method.getMethod(
                "clojure.lang.IObj withMeta(clojure.lang.IPersistentMap)");
        gen = new GeneratorAdapter(ACC_PUBLIC,

```

```

meth,
null,
null,
cv);

gen.visitCode();
gen.newInstance(objtype);
gen.dup();
gen.loadArg(0);

for(ISeq s = RT.keys(closes); s != null; s = s.next(), ++a)
{
    LocalBinding lb = (LocalBinding) s.first();
    gen.loadThis();
    Class primc = lb.getPrimitiveType();
    if(primc != null)
    {
        gen.getField(objtype, lb.name, Type.getType(primc));
    }
    else
    {
        gen.getField(objtype, lb.name, OBJECT_TYPE);
    }
}

gen.invokeConstructor(objtype,
                     new Method("<init>",
                               Type.VOID_TYPE, ctorTypes));
gen.returnValue();
gen.endMethod();
}

emitMethods(cv);

if(keywordCallsites.count() > 0)
{
    Method meth =
        Method.getMethod(
            "void swapThunk(int,clojure.lang.ILookupThunk)");

    GeneratorAdapter gen = new GeneratorAdapter(ACC_PUBLIC,
                                                meth,
                                                null,
                                                null,
                                                cv);

    gen.visitCode();
    Label endLabel = gen.newLabel();

    Label[] labels = new Label[keywordCallsites.count()];
    for(int i = 0; i < keywordCallsites.count();i++)
    {

```

```

        labels[i] = gen.newLabel();
    }
    gen.loadArg(0);
    gen.visitTableSwitchInsn(0, keywordCallsites.count()-1,
                           endLabel, labels);

    for(int i = 0; i < keywordCallsites.count();i++)
    {
        gen.mark(labels[i]);
        gen.loadThis();
        gen.loadArg(1);
        gen.putStatic(objtype,
                      thunkNameStatic(i), ILOOKUP_THUNK_TYPE);
        gen.goTo(endLabel);
    }

    gen.mark(endLabel);

    gen.returnValue();
    gen.endMethod();
}

//end of class
cv.visitEnd();

bytecode = cw.toByteArray();
if(RT.booleanCast(COMPILATION_FILES.deref()))
    writeClassFile(internalName, bytecode);
//    else
//        getCompiledClass();
}

private void emitKeywordCallsites(GeneratorAdapter clinitgen){
    for(int i=0;i<keywordCallsites.count();i++)
    {
        Keyword k = (Keyword) keywordCallsites.nth(i);
        clinitgen.newInstance(KEYWORD_LOOKUPSITE_TYPE);
        clinitgen.dup();
        emitValue(k,clinitgen);
        clinitgen.invokeConstructor(KEYWORD_LOOKUPSITE_TYPE,
                                   Method.getMethod("void <init>(clojure.lang.Keyword)"));
        clinitgen.dup();
        clinitgen.putStatic(objtype, siteNameStatic(i),
                           KEYWORD_LOOKUPSITE_TYPE);
        clinitgen.putStatic(objtype, thunkNameStatic(i),
                           ILOOKUP_THUNK_TYPE);
    }
}

protected void emitMethods(ClassVisitor gen){
}

```

```

    }

void emitListAsObjectArray(Object value, GeneratorAdapter gen){
    gen.push(((List) value).size());
    gen newArray(OBJECT_TYPE);
    int i = 0;
    for(Iterator it = ((List) value).iterator(); it.hasNext(); i++)
    {
        gen.dup();
        gen.push(i);
        emitValue(it.next(), gen);
        gen.arrayStore(OBJECT_TYPE);
    }
}

void emitValue(Object value, GeneratorAdapter gen){
    boolean partial = true;
    //System.out.println(value.getClass().toString());

    if(value instanceof String)
    {
        gen.push((String) value);
    }
    else if(value instanceof Integer)
    {
        gen.push((Integer) value).intValue();
        gen.invokeStatic(Type.getType(Integer.class),
                        Method.getMethod("Integer valueOf(int)"));
    }
    else if(value instanceof Long)
    {
        gen.push((Long) value).longValue();
        gen.invokeStatic(Type.getType(Long.class),
                        Method.getMethod("Long valueOf(long)"));
    }
    else if(value instanceof Double)
    {
        gen.push((Double) value).doubleValue();
        gen.invokeStatic(Type.getType(Double.class),
                        Method.getMethod("Double valueOf(double)"));
    }
    else if(value instanceof Character)
    {
        gen.push((Character) value).charValue();
        gen.invokeStatic(Type.getType(Character.class),
                        Method.getMethod("Character valueOf(char)"));
    }
    else if(value instanceof Class)
    {
        Class cc = (Class)value;
    }
}

```

```

if(cc.isPrimitive())
{
    Type bt;
    if ( cc == boolean.class )
        bt = Type.getType(Boolean.class);
    else if ( cc == byte.class )
        bt = Type.getType(Byte.class);
    else if ( cc == char.class )
        bt = Type.getType(Character.class);
    else if ( cc == double.class )
        bt = Type.getType(Double.class);
    else if ( cc == float.class )
        bt = Type.getType(Float.class);
    else if ( cc == int.class )
        bt = Type.getType(Integer.class);
    else if ( cc == long.class )
        bt = Type.getType(Long.class);
    else if ( cc == short.class )
        bt = Type.getType(Short.class);
    else throw new RuntimeException(
        "Can't embed unknown primitive in code: " + value);
    gen.getStatic( bt, "TYPE", Type.getType(Class.class) );
}
else
{
    gen.push(destubClassName(cc.getName()));
    gen.invokeStatic(Type.getType(Class.class),
                    Method.getMethod("Class forName(String)"));
}
}
else if(value instanceof Symbol)
{
    gen.push(((Symbol) value).ns);
    gen.push(((Symbol) value).name);
    gen.invokeStatic(Type.getType(Symbol.class),
                    Method.getMethod(
                        "clojure.lang.Symbol intern(String,String)"));
}
else if(value instanceof Keyword)
{
    emitValue(((Keyword) value).sym, gen);
    gen.invokeStatic(Type.getType(Keyword.class),
                    Method.getMethod(
                        "clojure.lang.Keyword intern(clojure.lang.Symbol)"));
}
//    else if(value instanceof KeywordCallSite)
//{
//        emitValue(((KeywordCallSite) value).k.sym, gen);
//        gen.invokeStatic(Type.getType(KeywordCallSite.class),
//                        Method.getMethod(

```

```

//      "clojure.lang.KeywordCallSite create(clojure.lang.Symbol))";
//
//      }
//      else if(value instanceof Var)
//      {
//          Var var = (Var) value;
//          gen.push(var.ns.name.toString());
//          gen.push(var.sym.toString());
//          gen.invokeStatic(RT_TYPE,
//              Method.getMethod("clojure.lang.Var var(String,String)"));
//      }
//      else if(value instanceof IPersistentMap)
//      {
//          List entries = new ArrayList();
//          for(Map.Entry entry :
//              ((Set<Map.Entry>) ((Map) value).entrySet()))
//          {
//              entries.add(entry.getKey());
//              entries.add(entry.getValue());
//          }
//          emitListAsObjectArray(entries, gen);
//          gen.invokeStatic(RT_TYPE,
//              Method.getMethod(
//                  "clojure.lang.IPersistentMap map(Object[])"));
//      }
//      else if(value instanceof IPersistentVector)
//      {
//          emitListAsObjectArray(value, gen);
//          gen.invokeStatic(RT_TYPE, Method.getMethod(
//              "clojure.lang.IPersistentVector vector(Object[])"));
//      }
//      else if(value instanceof ISeq ||
//          value instanceof IPersistentList)
//      {
//          emitListAsObjectArray(value, gen);
//          gen.invokeStatic(Type.getType(java.util.Arrays.class),
//              Method.getMethod("java.util.List asList(Object[])"));
//          gen.invokeStatic(Type.getType(PersistentList.class),
//              Method.getMethod(
//                  "clojure.lang.IPersistentList create(java.util.List)"));
//      }
//      else
//      {
//          String cs = null;
//          try
//          {
//              cs = RT.printString(value);
//              //System.out.println(
//              //  "WARNING SLOW CODE: " + value.getClass() +
//              //  " -> " + cs);
//          }

```

```

        catch(Exception e)
        {
            throw new RuntimeException(
                "Can't embed object in code, maybe print-dup not defined: " +
                value);
        }
        if(cs.length() == 0)
            throw new RuntimeException(
                "Can't embed unreadable object in code: " + value);

        if(cs.startsWith("#<"))
            throw new RuntimeException(
                "Can't embed unreadable object in code: " + cs);

        gen.push(cs);
        gen.invokeStatic(RT_TYPE, readStringMethod);
        partial = false;
    }

    if(partial)
    {
        if(value instanceof IObj &&
            RT.count(((IObj) value).meta()) > 0)
        {
            gen.checkCast(IOBJ_TYPE);
            emitValue(((IObj) value).meta(), gen);
            gen.checkCast(IPERSISTENTMAP_TYPE);
            gen.invokeInterface(IOBJ_TYPE,
                Method.getMethod(
                    "clojure.lang.IObj withMeta(clojure.lang.IPersistentMap)"));
            }
        }
    }

void emitConstants(GeneratorAdapter clinitgen){
    try
    {
        Var.pushThreadBindings(RT.map(RT.PRINT_DUP, RT.T));

        for(int i = 0; i < constants.count(); i++)
        {
            emitValue(constants.nth(i), clinitgen);
            clinitgen.checkCast(constantType(i));
            clinitgen.putStatic(objtype, constantName(i),
                constantType(i));
        }
    }
    finally
    {

```

```

        Var.popThreadBindings();
    }
}

boolean isMutable(LocalBinding lb){
    return isVolatile(lb) ||
        RT.booleanCast(RT.contains(fields, lb.sym)) &&
        RT.booleanCast(RT.get(lb.sym.meta(),
            Keyword.intern("unsynchronized-mutable")));
}

boolean isVolatile(LocalBinding lb){
    return RT.booleanCast(RT.contains(fields, lb.sym)) &&
        RT.booleanCast(RT.get(lb.sym.meta(),
            Keyword.intern("volatile-mutable")));
}

boolean isDeftype(){
    return fields != null;
}

void emitClearCloses(GeneratorAdapter gen){
//    int a = 1;
//    for(ISeq s = RT.keys(closes); s != null; s = s.next(), ++a)
//    {
//        LocalBinding lb = (LocalBinding) s.first();
//        Class primc = lb.getPrimitiveType();
//        if(primc == null)
//        {
//            gen.loadThis();
//            gen.visitInsn(Opcodes.ACONST_NULL);
//            gen.putField(objtype, lb.name, OBJECT_TYPE);
//        }
//    }
}

synchronized Class getCompiledClass(){
    if(compiledClass == null)
        try
        {
//            if(RT.booleanCast(COMPILER_FILES.deref()))
//                compiledClass =
//                    RT.classForName(name); //loader.defineClass(name, bytecode);
//            else
//            {
//                loader = (DynamicClassLoader) LOADER.deref();
//                compiledClass =
//                    loader.defineClass(name, bytecode, src);
//            }
        }
}

```

```

        catch(Exception e)
        {
            throw new RuntimeException(e);
        }
        return compiledClass;
    }

    public Object eval() throws Exception{
        if(isDeftype())
            return null;
        return getCompiledClass().newInstance();
    }

    public void emitLetFnInits(GeneratorAdapter gen,
                               ObjExpr objx,
                               IPersistentSet letFnLocals){
        //objx arg is enclosing objx, not this
        gen.checkCast(objtype);

        for(ISeq s = RT.keys(closes); s != null; s = s.next())
        {
            LocalBinding lb = (LocalBinding) s.first();
            if(letFnLocals.contains(lb))
            {
                Class primc = lb.getPrimitiveType();
                gen.dup();
                if(primc != null)
                {
                    objx.emitUnboxedLocal(gen, lb);
                    gen.putField(objtype, lb.name, Type.getType(primc));
                }
                else
                {
                    objx.emitLocal(gen, lb, false);
                    gen.putField(objtype, lb.name, OBJECT_TYPE);
                }
            }
            gen.pop();
        }
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        //emitting a Fn means constructing an instance, feeding
        // closed-overs from enclosing scope, if any
        //objx arg is enclosing objx, not this
        //getCompiledClass();
        if(isDeftype())
        {
            gen.visitInsn(Opcodes.ACONST_NULL);
        }
    }
}

```

```

        }
    else
    {
        gen.newInstance(objtype);
        gen.dup();
        gen.visitInsn(OpCodes.ACONST_NULL);
        for(ISeq s = RT.seq(closesExprs); s != null; s = s.next())
        {
            LocalBindingExpr lbe = (LocalBindingExpr) s.first();
            LocalBinding lb = lbe.b;
            if(lb.getPrimitiveType() != null)
                objx.emitUnboxedLocal(gen, lb);
            else
                objx.emitLocal(gen, lb, lbe.shouldClear);
        }
        gen.invokeConstructor(objtype,
                             new Method("<init>", Type.VOID_TYPE, ctorTypes()));
    }
    if(context == C.STATEMENT)
        gen.pop();
}

public boolean hasJavaClass() throws Exception{
    return true;
}

public Class getJavaClass() throws Exception{
    return (compiledClass != null) ? compiledClass
        : (tag != null) ? HostExpr.tagToClass(tag)
        : IFn.class;
}

public void emitAssignLocal(GeneratorAdapter gen,
                           LocalBinding lb,
                           Expr val){
    if(!isMutable(lb))
        throw new IllegalArgumentException(
            "Cannot assign to non-mutable: " + lb.name);
    Class primc = lb.getPrimitiveType();
    gen.loadThis();
    if(primc != null)
    {
        if(!(val instanceof MaybePrimitiveExpr &&
             ((MaybePrimitiveExpr) val).canEmitPrimitive()))
            throw new IllegalArgumentException(
                "Must assign primitive to primitive mutable: " +
                lb.name);
        MaybePrimitiveExpr me = (MaybePrimitiveExpr) val;
        me.emitUnboxed(C.EXPRESSION, this, gen);
        gen.putField(objtype, lb.name, Type.getType(primc));
    }
}

```

```

        }
    else
    {
        val.emit(C.EXPRESSION, this, gen);
        gen.putField(objtype, lb.name, OBJECT_TYPE);
    }
}

private void emitLocal(GeneratorAdapter gen,
                      LocalBinding lb,
                      boolean clear){
    if(closes.containsKey(lb))
    {
        Class primc = lb.getPrimitiveType();
        gen.loadThis();
        if(primc != null)
        {
            gen.getField(objtype, lb.name, Type.getType(primc));
            HostExpr.emitBoxReturn(this, gen, primc);
        }
        else
        {
            gen.getField(objtype, lb.name, OBJECT_TYPE);
            if(onceOnly && clear && lb.canBeCleared)
            {
                gen.loadThis();
                gen.visitInsn(OpCodes.ACONST_NULL);
                gen.putField(objtype, lb.name, OBJECT_TYPE);
            }
        }
    }
    else
    {
        int argoff = isStatic?0:1;
        Class primc = lb.getPrimitiveType();
        // String rep =
        // lb.sym.name + " " +
        // lb.toString().substring(
        //     lb.toString().lastIndexOf('@'));
        if(lb.isArg)
        {
            gen.loadArg(lb.idx-argoff);
            if(primc != null)
                HostExpr.emitBoxReturn(this, gen, primc);
            else
            {
                if(clear && lb.canBeCleared)
                {
                    System.out.println("clear: " + rep);
                    gen.visitInsn(OpCodes.ACONST_NULL);
                }
            }
        }
    }
}

```

```

                gen.storeArg(lb.idx - argoff);
            }
        else
        {
            System.out.println("use: " + rep);
        }
    }
else
{
    if(primc != null)
    {
        gen.visitVarInsn(Type.getType(primc)
                        .getOpcode(Opcodes.ILOAD),
                        lb.idx);
        HostExpr.emitBoxReturn(this, gen, primc);
    }
else
{
    gen.visitVarInsn(
        OBJECT_TYPE.getOpcode(Opcodes.ILOAD),
        lb.idx);
    if(clear && lb.canBeCleared)
    {
        System.out.println("clear: " + rep);
        gen.visitInsn(Opcodes.ACONST_NULL);
        gen.visitVarInsn(OBJECT_TYPE
                        .getOpcode(Opcodes.ISTORE),
                        lb.idx);
    }
else
{
    System.out.println("use: " + rep);
}
}
}
}

private void emitUnboxedLocal(GeneratorAdapter gen, LocalBinding lb){
    int argoff = isStatic?0:1;
    Class primc = lb.getPrimitiveType();
    if(closes.containsKey(lb))
    {
        gen.loadThis();
        gen.getField(objtype, lb.name, Type.getType(primc));
    }
    else if(lb.isArg)
        gen.loadArg(lb.idx-argoff);
    else

```

```

        gen.visitVarInsn(
            Type.getType(primc).getOpcodes(0pcodes.ILOAD), lb.idx);
    }

    public void emitVar(GeneratorAdapter gen, Var var){
        Integer i = (Integer) vars.valAt(var);
        emitConstant(gen, i);
        //gen.getStatic(fntype, munge(var.sym.toString()), VAR_TYPE);
    }

    final static Method varGetMethod =
        Method.getMethod("Object get()");
    final static Method varGetRawMethod =
        Method.getMethod("Object getRawRoot()");

    public void emitVarValue(GeneratorAdapter gen, Var v){
        Integer i = (Integer) vars.valAt(v);
        if(!v.isDynamic())
        {
            emitConstant(gen, i);
            gen.invokeVirtual(VAR_TYPE, varGetRawMethod);
        }
        else
        {
            emitConstant(gen, i);
            gen.invokeVirtual(VAR_TYPE, varGetMethod);
        }
    }

    public void emitKeyword(GeneratorAdapter gen, Keyword k){
        Integer i = (Integer) keywords.valAt(k);
        emitConstant(gen, i);
//        gen.getStatic(fntype, munge(k.sym.toString()), KEYWORD_TYPE);
    }

    public void emitConstant(GeneratorAdapter gen, int id){
        gen.getStatic(objtype, constantName(id), constantType(id));
    }

    String constantName(int id){
        return CONST_PREFIX + id;
    }

    String siteName(int n){
        return "__site__" + n;
    }

    String siteNameStatic(int n){
        return siteName(n) + "__";
    }
}

```

```

    }

    String thunkName(int n){
        return "__thunk__" + n;
    }

    String cachedClassName(int n){
        return "__cached_class__" + n;
    }

    String cachedVarName(int n){
        return "__cached_var__" + n;
    }

    String cachedProtoFnName(int n){
        return "__cached_proto_fn__" + n;
    }

    String cachedProtoImplName(int n){
        return "__cached_proto_impl__" + n;
    }

    String varCallsiteName(int n){
        return "__var__callsite__" + n;
    }

    String thunkNameStatic(int n){
        return thunkName(n) + "--";
    }

    Type constantType(int id){
        Object o = constants.nth(id);
        Class c = o.getClass();
        if(Modifier.isPublic(c.getModifiers()))
        {
            //can't emit derived fn types due to visibility
            if(LazySeq.class.isAssignableFrom(c))
                return Type.getType(ISeq.class);
            else if(c == Keyword.class)
                return Type.getType(Keyword.class);
            //else if(c == KeywordCallSite.class)
            //    return Type.getType(KeywordCallSite.class);
            else if(RestFn.class.isAssignableFrom(c))
                return Type.getType(RestFn.class);
            else if(IFn.class.isAssignableFrom(c))
                return Type.getType(IFn.class);
            else if(c == Var.class)
                return Type.getType(Var.class);
            else if(c == String.class)
                return Type.getType(String.class);
        }
    }
}

```

```

//           return Type.getType(c);
        }
        return OBJECT_TYPE;
    }

}

enum PATHTYPE {
    PATH, BRANCH;
}

static class PathNode{
    final PATHTYPE type;
    final PathNode parent;

    PathNode(PATHTYPE type, PathNode parent) {
        this.type = type;
        this.parent = parent;
    }
}

static PathNode clearPathRoot(){
    return (PathNode) CLEAR_ROOT.get();
}

enum PSTATE{
    REQ, REST, DONE
}

public static class FnMethod extends ObjMethod{
    //localbinding->localbinding
    PersistentVector reqParms = PersistentVector.EMPTY;
    LocalBinding restParm = null;
    Type[] argtypes;
    Class[] argclasses;
    Class retClass;
    String prim ;

    public FnMethod(ObjExpr objx, ObjMethod parent){
        super(objx, parent);
    }

    static public char classChar(Object x){
        Class c = null;
        if(x instanceof Class)
            c = (Class) x;
        else if(x instanceof Symbol)
            c = primClass((Symbol) x);
        if(c == null || !c.isPrimitive())

```

```

        return '0';
    if(c == long.class)
        return 'L';
    if(c == double.class)
        return 'D';
    throw new IllegalArgumentException(
        "Only long and double primitives are supported");
}

static public String primInterface(IPersistentVector arglist)
throws Exception{
    StringBuilder sb = new StringBuilder();
    for(int i=0;i<arglist.count();i++)
        sb.append(classChar(tagOf(arglist.nth(i))));
    sb.append(classChar(tagOf(arglist)));
    String ret = sb.toString();
    boolean prim = ret.contains("L") || ret.contains("D");
    if(prim && arglist.count() > 4)
        throw new IllegalArgumentException(
            "fns taking primitives support only 4 or fewer args");
    if(prim)
        return "clojure.langIFn$" + ret;
    return null;
}

static FnMethod parse(ObjExpr objx,
                     ISeq form,
                     boolean isStatic)
throws Exception{
    //([args] body...)
    IPersistentVector parms = (IPersistentVector) RT.first(form);
    ISeq body = RT.next(form);
    try
    {
        FnMethod method =
            new FnMethod(objx, (ObjMethod) METHOD.deref());
        method.line = (Integer) LINE.deref();
        //register as the current method and set up a new env frame
        PathNode pnode = (PathNode) CLEAR_PATH.get();
        if(pnode == null)
            pnode = new PathNode(PATHTYPE.PATH,null);
        Var.pushThreadBindings(
            RT.map(
                METHOD, method,
                LOCAL_ENV, LOCAL_ENV.deref(),
                LOOP_LOCALS, null,
                NEXT_LOCAL_NUM, 0
                ,CLEAR_PATH, pnode
                ,CLEAR_ROOT, pnode
                ,CLEAR_SITES, PersistentHashMap.EMPTY
            )
        );
    }
}

```

```

        });

method.prim = primInterface(parms);
if(method.prim != null)
    method.prim = method.prim.replace('.', '/');

method.retClass = tagClass(tagOf(parms));
if(method.retClass.isPrimitive() &&
   !(method.retClass == double.class ||
   method.retClass == long.class))
    throw new IllegalArgumentException(
        "Only long and double primitives are supported");

//register 'this' as local 0
//registerLocal(THISFN, null, null);
if(!isStatic)
{
    if(objx.thisName != null)
        registerLocal(Symbol.intern(objx.thisName),
                      null, null, false);
    else
        getAndIncLocalNum();
}
PSTATE state = PSTATE.REQ;
PersistentVector argLocals = PersistentVector.EMPTY;
ArrayList<Type> argtypes = new ArrayList();
ArrayList<Class> argclasses = new ArrayList();
for(int i = 0; i < parms.count(); i++)
{
    if(!(parms.nth(i) instanceof Symbol))
        throw new IllegalArgumentException(
            "fn params must be Symbols");
    Symbol p = (Symbol) parms.nth(i);
    if(p.getNamespace() != null)
        throw new Exception(
            "Can't use qualified name as parameter: " + p);
    if(p.equals(_AMP_))
    {
        // if(isStatic)
        //     throw new Exception(
        //         "Variadic fns cannot be static");
        if(state == PSTATE.REQ)
            state = PSTATE.REST;
        else
            throw new Exception("Invalid parameter list");
    }
    else
    {
        Class pc = primClass(tagClass(tagOf(p)));

```

```

// if(pc.isPrimitive() && !isStatic)
// {
//     pc = Object.class;
//     p = (Symbol)
//         ((IObj) p)
//         .withMeta(
//             (IPersistentMap) RT.assoc(
//                 RT.meta(p), RT.TAG_KEY, null));
// }
// throw new Exception(
// "Non-static fn can't have primitive parameter: " + p);
// if(pc.isPrimitive() &&
//     !(pc == double.class || pc == long.class))
//     throw new IllegalArgumentException(
// "Only long and double primitives are supported: " + p);

if(state == PSTATE.REST && tagOf(p) != null)
    throw new Exception(
        "& arg cannot have type hint");
if(state == PSTATE.REST && method.prim != null)
    throw new Exception(
        "fns taking primitives cannot be variadic");

if(state == PSTATE.REST)
    pc = ISeq.class;
argtypes.add(Type.getType(pc));
argclasses.add(pc);
LocalBinding lb =
    pc.isPrimitive()
    ? registerLocal(p, null,
        new MethodParamExpr(pc), true)
    : registerLocal(p, state == PSTATE.REST
        ? ISEQ
        : tagOf(p), null, true);
argLocals = argLocals.cons(lb);
switch(state)
{
    case REQ:
        method.reqParms = method.reqParms.cons(lb);
        break;
    case REST:
        method.restParm = lb;
        state = PSTATE.DONE;
        break;

    default:
        throw new Exception("Unexpected parameter"));
}
}

```

```

        if(method.reqParms.count() > MAX_POSITIONAL_ARITY)
            throw new Exception("Can't specify more than " +
                MAX_POSITIONAL_ARITY + " params");
    LOOP_LOCALS.set(argLocals);
    method.argLocals = argLocals;
//    if(isStatic)
//        if(method.prim != null)
//    {
//        method.argtypes =
//            argtypes.toArray(new Type[argtypes.size()]);
//        method.argclasses =
//            argclasses.toArray(new Class[argtypes.size()]);
//        for(int i = 0; i < method.argclasses.length; i++)
//        {
//            if(method.argclasses[i] == long.class ||
//                method.argclasses[i] == double.class)
//                getAndIncLocalNum();
//        }
//    }
    method.body = (new BodyExpr.Parser()).parse(C.RETURN, body);
    return method;
}
finally
{
    Var.popThreadBindings();
}
}

public void emit(ObjExpr fn, ClassVisitor cv){
    if(prim != null)
        doEmitPrim(fn, cv);
    else if(fn.isStatic)
        doEmitStatic(fn, cv);
    else
        doEmit(fn, cv);
}

public void doEmitStatic(ObjExpr fn, ClassVisitor cv){
    Method ms =
        new Method("invokeStatic", getReturnType(), argtypes);

    GeneratorAdapter gen =
        new GeneratorAdapter(ACC_PUBLIC + ACC_STATIC,
            ms,
            null,
            //todo don't hardwire this
            EXCEPTION_TYPES,
            cv);
    gen.visitCode();
    Label loopLabel = gen.mark();
}

```

```

gen.visitLineNumber(line, loopLabel);
try
{
    Var.pushThreadBindings(
        RT.map(LOOP_LABEL, loopLabel, METHOD, this));
    emitBody(objx, gen, retClass, body);

    Label end = gen.mark();
    for(ISeq lbs = argLocals.seq();
        lbs != null;
        lbs = lbs.next())
    {
        LocalBinding lb = (LocalBinding) lbs.first();
        gen.visitLocalVariable(lb.name,
            argtypes[lb.idx].getDescriptor(), null,
            loopLabel, end, lb.idx);
    }
}
catch(Exception e)
{
    throw new RuntimeException(e);
}
finally
{
    Var.popThreadBindings();
}

gen.returnValue();
//gen.visitMaxs(1, 1);
gen.endMethod();

//generate the regular invoke, calling the static method
Method m =
    new Method(getMethodName(), OBJECT_TYPE, getArgTypes());

gen = new GeneratorAdapter(ACC_PUBLIC,
    m,
    null,
    //todo don't hardwire this
    EXCEPTION_TYPES,
    cv);
gen.visitCode();
for(int i = 0; i < argtypes.length; i++)
{
    gen.loadArg(i);
    HostExpr.emitUnboxArg(fn, gen, argclasses[i]);
}
gen.invokeStatic(objx.objtype, ms);
gen.box(getReturnType());

```

```

        gen.returnValue();
        //gen.visitMaxs(1, 1);
        gen.endMethod();

    }

    public void doEmitPrim(ObjExpr fn, ClassVisitor cv){
        Method ms =
            new Method("invokePrim", getReturnType(), argtypes);

        GeneratorAdapter gen =
            new GeneratorAdapter(ACC_PUBLIC + ACC_FINAL,
                ms,
                null,
                //todo don't hardwire this
                EXCEPTION_TYPES,
                cv);
        gen.visitCode();

        Label loopLabel = gen.mark();
        gen.visitLineNumber(line, loopLabel);
        try
        {
            Var.pushThreadBindings(RT.map(LOOP_LABEL, loopLabel,
                METHOD, this));
            emitBody(objx, gen, retClass, body);

            Label end = gen.mark();
            gen.visitLocalVariable("this",
                "Ljava/lang/Object;", null,
                loopLabel, end, 0);
            for(ISeq lbs = argLocals.seq();
                lbs != null;
                lbs = lbs.next())
            {
                LocalBinding lb = (LocalBinding) lbs.first();
                gen.visitLocalVariable(lb.name,
                    argtypes[lb.idx-1].getDescriptor(), null,
                    loopLabel, end, lb.idx);
            }
        }
        catch(Exception e)
        {
            throw new RuntimeException(e);
        }
        finally
        {
            Var.popThreadBindings();
        }
    }
}

```

```

gen.returnValue();
//gen.visitMaxs(1, 1);
gen.endMethod();

//generate the regular invoke, calling the prim method
Method m =
    new Method(getMethodName(), OBJECT_TYPE, getArgTypes());

gen = new GeneratorAdapter(ACC_PUBLIC,
                           m,
                           null,
                           //todo don't hardwire this
                           EXCEPTION_TYPES,
                           cv);

gen.visitCode();
gen.loadThis();
for(int i = 0; i < argtypes.length; i++)
{
    gen.loadArg(i);
    HostExpr.emitUnboxArg(fn, gen, argclasses[i]);
}
gen.invokeInterface(Type.getType("L"+prim+";"), ms);
gen.box(getReturnType());

gen.returnValue();
//gen.visitMaxs(1, 1);
gen.endMethod();

}

public void doEmit(ObjExpr fn, ClassVisitor cv){
    Method m =
        new Method(getMethodName(), getReturnType(), getArgTypes());

    GeneratorAdapter gen =
        new GeneratorAdapter(ACC_PUBLIC,
                           m,
                           null,
                           //todo don't hardwire this
                           EXCEPTION_TYPES,
                           cv);

    gen.visitCode();

    Label loopLabel = gen.mark();
    gen.visitLineNumber(line, loopLabel);
    try
    {
        Var.pushThreadBindings(
            RT.map(LOOP_LABEL, loopLabel, METHOD, this));
    }
}

```

```

        body.emit(C.RETURN, fn, gen);
        Label end = gen.mark();

        gen.visitLocalVariable("this", "Ljava/lang/Object;",
                               null, loopLabel, end, 0);
        for(ISeq lbs = argLocals.seq();
            lbs != null;
            lbs = lbs.next())
        {
            LocalBinding lb = (LocalBinding) lbs.first();
            gen.visitLocalVariable(lb.name, "Ljava/lang/Object;",
                                   null, loopLabel, end, lb.idx);
        }
    }
    finally
    {
        Var.popThreadBindings();
    }

    gen.returnValue();
    //gen.visitMaxs(1, 1);
    gen.endMethod();
}

public final PersistentVector reqParms(){
    return reqParms;
}

public final LocalBinding restParm(){
    return restParm;
}

boolean isVariadic(){
    return restParm != null;
}

int numParams(){
    return reqParms.count() + (isVariadic() ? 1 : 0);
}

String getMethodName(){
    return isVariadic()?"doInvoke":"invoke";
}

Type getReturnType(){
    if(prim != null) //objx.isStatic)
        return Type.getType(retClass);
}

```

```

        return OBJECT_TYPE;
    }

    Type[] getArgTypes(){
        if(isVariadic() && reqParms.count() == MAX_POSITIONAL_ARITY)
        {
            Type[] ret = new Type[MAX_POSITIONAL_ARITY + 1];
            for(int i = 0;i<MAX_POSITIONAL_ARITY + 1;i++)
                ret[i] = OBJECT_TYPE;
            return ret;
        }
        return ARG_TYPES[numParams()];
    }

    void emitClearLocals(GeneratorAdapter gen){
//        for(int i = 1; i < numParams() + 1; i++)
//        {
//            if(!localsUsedInCatchFinally.contains(i))
//            {
//                gen.visitInsn(Opcodes.ACONST_NULL);
//                gen.visitVarInsn(
//                    OBJECT_TYPE.getOpcode(Opcodes.ISTORE), i);
//            }
//        }
//        for(int i = numParams() + 1; i < maxLocal + 1; i++)
//        {
//            if(!localsUsedInCatchFinally.contains(i))
//            {
//                LocalBinding b =
//                    (LocalBinding) RT.get(indexlocals, i);
//                if(b == null || maybePrimitiveType(b.init) == null)
//                {
//                    gen.visitInsn(Opcodes.ACONST_NULL);
//                    gen.visitVarInsn(
//                        OBJECT_TYPE.getOpcode(Opcodes.ISTORE), i);
//                }
//            }
//        }
//        if(((FnExpr)objx).onceOnly)
//        {
//            objx.emitClearCloses(gen);
//        }
    }

    abstract public static class ObjMethod{
        //when closures are defined inside other closures, the closed
        //over locals need to be propagated to the enclosing objx
        public final ObjMethod parent;
        //localbinding->localbinding
    }
}

```

```

IPersistentMap locals = null;
//num->localbinding
IPersistentMap indexlocals = null;
Expr body = null;
ObjExpr objx;
PersistentVector argLocals;
int maxLocal = 0;
int line;
PersistentHashSet localsUsedInCatchFinally =
    PersistentHashSet.EMPTY;
protected IPersistentMap methodMeta;

public final IPersistentMap locals(){
    return locals;
}

public final Expr body(){
    return body;
}

public final ObjExpr objx(){
    return objx;
}

public final PersistentVector argLocals(){
    return argLocals;
}

public final int maxLocal(){
    return maxLocal;
}

public final int line(){
    return line;
}

public ObjMethod(ObjExpr objx, ObjMethod parent){
    this.parent = parent;
    this.objx = objx;
}

static void emitBody(ObjExpr objx,
                    GeneratorAdapter gen,
                    Class retClass,
                    Expr body)
throws Exception{
    MaybePrimitiveExpr be = (MaybePrimitiveExpr) body;
    if(Util.isPrimitive(retClass) && be.canEmitPrimitive())
    {
}

```

```

Class bc = maybePrimitiveType(be);
if(bc == retClass)
    be.emitUnboxed(C.RETURN, objx, gen);
else if(retClass == long.class && bc == int.class)
{
    be.emitUnboxed(C.RETURN, objx, gen);
    gen.visitInsn(I2L);
}
else if(retClass == double.class && bc == float.class)
{
    be.emitUnboxed(C.RETURN, objx, gen);
    gen.visitInsn(F2D);
}
else if(retClass == int.class && bc == long.class)
{
    be.emitUnboxed(C.RETURN, objx, gen);
    gen.invokeStatic(RT_TYPE,
                      Method.getMethod("int intCast(long)"));
}
else if(retClass == float.class && bc == double.class)
{
    be.emitUnboxed(C.RETURN, objx, gen);
    gen.visitInsn(D2F);
}
else
    throw new IllegalArgumentException(
        "Mismatched primitive return, expected: "
        + retClass + ", had: " + be.getJavaClass());
}
else
{
    body.emit(C.RETURN, objx, gen);
    if(retClass == void.class)
    {
        gen.pop();
    }
    else
        gen.unbox(Type.getType(retClass));
}
}

abstract int numParams();
abstract String getMethodName();
abstract Type getReturnType();
abstract Type[] getArgTypes();

public void emit(ObjExpr fn, ClassVisitor cv){
    Method m =
        new Method(getMethodName(), getReturnType(), getArgTypes());
    GeneratorAdapter gen =

```

```

        new GeneratorAdapter(ACC_PUBLIC,
            m,
            null,
            //todo don't hardwire this
            EXCEPTION_TYPES,
            cv);
    gen.visitCode();

    Label loopLabel = gen.mark();
    gen.visitLineNumber(line, loopLabel);
    try
    {
        Var.pushThreadBindings(
            RT.map(LOOP_LABEL, loopLabel, METHOD, this));

        body.emit(C.RETURN, fn, gen);
        Label end = gen.mark();
        gen.visitLocalVariable("this", "Ljava/lang/Object;",
            null, loopLabel, end, 0);
        for(ISeq lbs = argLocals.seq();
            lbs != null;
            lbs = lbs.next())
        {
            LocalBinding lb = (LocalBinding) lbs.first();
            gen.visitLocalVariable(lb.name, "Ljava/lang/Object;",
                null, loopLabel, end, lb.idx);
        }
    }
    finally
    {
        Var.popThreadBindings();
    }

    gen.returnValue();
    //gen.visitMaxs(1, 1);
    gen.endMethod();
}

void emitClearLocals(GeneratorAdapter gen){
}

void emitClearLocalsOld(GeneratorAdapter gen){
    for(int i=0;i<argLocals.count();i++)
    {
        LocalBinding lb = (LocalBinding) argLocals.nth(i);
        if(!localsUsedInCatchFinally.contains(lb.idx) &&
            lb.getPrimitiveType() == null)
        {
            gen.visitInsn(Opcodes.ACONST_NULL);
            gen.storeArg(lb.idx - 1);
        }
    }
}

```

```

        }

    }

    for(int i = 1; i < numParams() + 1; i++)
    {
        if(!localsUsedInCatchFinally.contains(i))
        {
            gen.visitInsn(Opcodes.ACONST_NULL);
            gen.visitVarInsn(
                OBJECT_TYPE.getOpcode(Opcodes.ISTORE), i);
        }
    }

    for(int i = numParams() + 1; i < maxLocal + 1; i++)
    {
        if(!localsUsedInCatchFinally.contains(i))
        {
            LocalBinding b = (LocalBinding) RT.get(indexlocals, i);
            if(b == null || maybePrimitiveType(b.init) == null)
            {
                gen.visitInsn(Opcodes.ACONST_NULL);
                gen.visitVarInsn(
                    OBJECT_TYPE.getOpcode(Opcodes.ISTORE), i);
            }
        }
    }
}

public static class LocalBinding{
    public final Symbol sym;
    public final Symbol tag;
    public Expr init;
    public final int idx;
    public final String name;
    public final boolean isArg;
    public final PathNode clearPathRoot;
    public boolean canBeCleared = true;
    public boolean recurMismatch = false;

    public LocalBinding(int num, Symbol sym, Symbol tag, Expr init,
                       boolean isArg, PathNode clearPathRoot)
        throws Exception{
        if(maybePrimitiveType(init) != null && tag != null)
            throw new UnsupportedOperationException(
                "Can't type hint a local with a primitive initializer");
        this.idx = num;
        this.sym = sym;
        this.tag = tag;
        this.init = init;
        this.isArg = isArg;
    }
}

```

```

        this.clearPathRoot = clearPathRoot;
        name = munge(sym.name);
    }

    public boolean hasJavaClass() throws Exception{
        if(init != null && init.hasJavaClass()
            && Util.isPrimitive(init.getJavaClass())
            && !(init instanceof MaybePrimitiveExpr))
            return false;
        return tag != null
            || (init != null && init.hasJavaClass());
    }

    public Class getJavaClass() throws Exception{
        return tag != null ? HostExpr.tagToClass(tag)
            : init.getJavaClass();
    }

    public Class getPrimitiveType(){
        return maybePrimitiveType(init);
    }
}

public static class LocalBindingExpr
    implements Expr, MaybePrimitiveExpr, AssignableExpr{
    public final LocalBinding b;
    public final Symbol tag;

    public final PathNode clearPath;
    public final PathNode clearRoot;
    public boolean shouldClear = false;

    public LocalBindingExpr(LocalBinding b, Symbol tag)
        throws Exception{
        if(b.getPrimitiveType() != null && tag != null)
            throw new UnsupportedOperationException(
                "Can't type hint a primitive local");
        this.b = b;
        this.tag = tag;

        this.clearPath = (PathNode)CLEAR_PATH.get();
        this.clearRoot = (PathNode)CLEAR_ROOT.get();
        IPersistentCollection sites =
            (IPersistentCollection) RT.get(CLEAR_SITES.get(),b);

        if(b.idx > 0)
        {
            // Object dummy;
        }
    }
}

```

```

        if(sites != null)
        {
            for(ISeq s = sites.seq();s!=null;s = s.next())
            {
                LocalBindingExpr o = (LocalBindingExpr) s.first();
                PathNode common = commonPath(clearPath,o.clearPath);
                if(common != null && common.type == PATHTYPE.PATH)
                    o.shouldClear = false;
                //else
                //    dummy = null;
            }
        }

        if(clearRoot == b.clearPathRoot)
        {
            this.shouldClear = true;
            sites = RT.conj(sites,this);
            CLEAR_SITES.set(RT.assoc(CLEAR_SITES.get(), b, sites));
        }
        //else
        //    dummy = null;
    }
}

public Object eval() throws Exception{
    throw new UnsupportedOperationException("Can't eval locals");
}

public boolean canEmitPrimitive(){
    return b.getPrimitiveType() != null;
}

public void emitUnboxed(C context,
                        ObjExpr objx,
                        GeneratorAdapter gen){
    objx.emitUnboxedLocal(gen, b);
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    if(context != C.STATEMENT)
        objx.emitLocal(gen, b, shouldClear);
}

public Object evalAssign(Expr val) throws Exception{
    throw new UnsupportedOperationException("Can't eval locals");
}

public void emitAssign(C context,
                      ObjExpr objx,
                      GeneratorAdapter gen,

```

```

        Expr val){
    objx.emitAssignLocal(gen, b,val);
    if(context != C.STATEMENT)
        objx.emitLocal(gen, b, false);
}

public boolean hasJavaClass() throws Exception{
    return tag != null || b.hasJavaClass();
}

public Class getJavaClass() throws Exception{
    if(tag != null)
        return HostExpr.tagToClass(tag);
    return b.getJavaClass();
}

}

public static class BodyExpr implements Expr, MaybePrimitiveExpr{
    PersistentVector exprs;

    public final PersistentVector exprs(){
        return exprs;
    }

    public BodyExpr(PersistentVector exprs){
        this.exprs = exprs;
    }

    static class Parser implements IParser{
        public Expr parse(C context, Object frms) throws Exception{
            ISeq forms = (ISeq) frms;
            if(Util.equals(RT.first(forms), DO))
                forms = RT.next(forms);
            PersistentVector exprs = PersistentVector.EMPTY;
            for(; forms != null; forms = forms.next())
            {
                Expr e = (context != C.EVAL &&
                           (context == C.STATEMENT ||
                            forms.next() != null)) ?
                            analyze(C.STATEMENT, forms.first()) :
                            analyze(context, forms.first());
                exprs = exprs.cons(e);
            }
            if(exprs.count() == 0)
                exprs = exprs.cons(NIL_EXPR);
            return new BodyExpr(exprs);
        }
    }
}

```

```

public Object eval() throws Exception{
    Object ret = null;
    for(Object o : exprs)
    {
        Expr e = (Expr) o;
        ret = e.eval();
    }
    return ret;
}

public boolean canEmitPrimitive(){
    return lastExpr() instanceof MaybePrimitiveExpr &&
           ((MaybePrimitiveExpr)lastExpr()).canEmitPrimitive();
}

public void emitUnboxed(C context,
                        ObjExpr objx,
                        GeneratorAdapter gen){
    for(int i = 0; i < exprs.count() - 1; i++)
    {
        Expr e = (Expr) exprs.nth(i);
        e.emit(C.STATEMENT, objx, gen);
    }
    MaybePrimitiveExpr last =
        (MaybePrimitiveExpr) exprs.nth(exprs.count() - 1);
    last.emitUnboxed(context, objx, gen);
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    for(int i = 0; i < exprs.count() - 1; i++)
    {
        Expr e = (Expr) exprs.nth(i);
        e.emit(C.STATEMENT, objx, gen);
    }
    Expr last = (Expr) exprs.nth(exprs.count() - 1);
    last.emit(context, objx, gen);
}

public boolean hasJavaClass() throws Exception{
    return lastExpr().hasJavaClass();
}

public Class getJavaClass() throws Exception{
    return lastExpr().getJavaClass();
}

private Expr lastExpr(){
    return (Expr) exprs.nth(exprs.count() - 1);
}

```

```

}

public static class BindingInit{
    LocalBinding binding;
    Expr init;

    public final LocalBinding binding(){
        return binding;
    }

    public final Expr init(){
        return init;
    }

    public BindingInit(LocalBinding binding, Expr init){
        this.binding = binding;
        this.init = init;
    }
}

public static class LetFnExpr implements Expr{
    public final PersistentVector bindingInits;
    public final Expr body;

    public LetFnExpr(PersistentVector bindingInits, Expr body){
        this.bindingInits = bindingInits;
        this.body = body;
    }

    static class Parser implements IParser{
        public Expr parse(C context, Object frm) throws Exception{
            ISeq form = (ISeq) frm;
            //((letfns* [var (fn [args] body) ...] body...)
            if(!(RT.second(form) instanceof IPersistentVector))
                throw new IllegalArgumentException(
                    "Bad binding form, expected vector");

            IPersistentVector bindings =
                (IPersistentVector) RT.second(form);
            if((bindings.count() % 2) != 0)
                throw new IllegalArgumentException(
                    "Bad binding form, expected matched symbol expression pairs");

            ISeq body = RT.next(RT.next(form));

            if(context == C.EVAL)
                return
                    analyze(context,
                        RT.list(
                            RT.list(FN, PersistentVector.EMPTY,

```

```

        form));

IPersistentMap dynamicBindings =
    RT.map(LOCAL_ENV, LOCAL_ENV.deref(),
           NEXT_LOCAL_NUM, NEXT_LOCAL_NUM.deref());

try
{
    Var.pushThreadBindings(dynamicBindings);

    //pre-seed env (like Lisp labels)
    PersistentVector lbs = PersistentVector.EMPTY;
    for(int i = 0; i < bindings.count(); i += 2)
    {
        if(!(bindings.nth(i) instanceof Symbol))
            throw new IllegalArgumentException(
                "Bad binding form, expected symbol, got: " +
                bindings.nth(i));
        Symbol sym = (Symbol) bindings.nth(i);
        if(sym.getNamespace() != null)
            throw new Exception(
                "Can't let qualified name: " + sym);
        LocalBinding lb =
            registerLocal(sym, tagOf(sym), null, false);
        lb.canBeCleared = false;
        lbs = lbs.cons(lb);
    }
    PersistentVector bindingInits = PersistentVector.EMPTY;
    for(int i = 0; i < bindings.count(); i += 2)
    {
        Symbol sym = (Symbol) bindings.nth(i);
        Expr init =
            analyze(C.EXPRESSION, bindings.nth(i + 1),
                    sym.name);
        LocalBinding lb = (LocalBinding) lbs.nth(i / 2);
        lb.init = init;
        BindingInit bi = new BindingInit(lb, init);
        bindingInits = bindingInits.cons(bi);
    }
    return
        new LetFnExpr(bindingInits,
                      (new BodyExpr.Parser())
                        .parse(context, body));
}
finally
{
    Var.popThreadBindings();
}
}
}
```

```

public Object eval() throws Exception{
    throw new UnsupportedOperationException("Can't eval letfns");
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    for(int i = 0; i < bindingInits.count(); i++)
    {
        BindingInit bi = (BindingInit) bindingInits.nth(i);
        gen.visitInsn(Opcodes.ACONST_NULL);
        gen.visitVarInsn(
            OBJECT_TYPE.getOpcode(Opcodes.ISTORE), bi.binding.idx);
    }

    IPersistentSet lbset = PersistentHashSet.EMPTY;

    for(int i = 0; i < bindingInits.count(); i++)
    {
        BindingInit bi = (BindingInit) bindingInits.nth(i);
        lbset = (IPersistentSet) lbset.cons(bi.binding);
        bi.init.emit(C.EXPRESSION, objx, gen);
        gen.visitVarInsn(
            OBJECT_TYPE.getOpcode(Opcodes.ISTORE), bi.binding.idx);
    }

    for(int i = 0; i < bindingInits.count(); i++)
    {
        BindingInit bi = (BindingInit) bindingInits.nth(i);
        ObjExpr fe = (ObjExpr) bi.init;
        gen.visitVarInsn(
            OBJECT_TYPE.getOpcode(Opcodes.ILOAD), bi.binding.idx);
        fe.emitLetFnInits(gen, objx, lbset);
    }

    Label loopLabel = gen.mark();

    body.emit(context, objx, gen);

    Label end = gen.mark();
    //    gen.visitLocalVariable("this", "Ljava/lang/Object;",
    //                           null, loopLabel, end, 0);
    for(ISeq bis = bindingInits.seq(); bis != null; bis = bis.next())
    {
        BindingInit bi = (BindingInit) bis.first();
        String lname = bi.binding.name;
        if(lname.endsWith("__auto__"))
            lname += RT.nextID();
        Class primc = maybePrimitiveType(bi.init);
        if(primc != null)
            gen.visitLocalVariable(lname,

```

```

        Type.getDescriptor(primc),
        null, loopLabel, end,
        bi.binding.idx);
    else
        gen.visitLocalVariable lname, "Ljava/lang/Object;",
        null, loopLabel, end,
        bi.binding.idx);
    }
}

public boolean hasJavaClass() throws Exception{
    return body.hasJavaClass();
}

public Class getJavaClass() throws Exception{
    return body.getJavaClass();
}
}

public static class LetExpr implements Expr, MaybePrimitiveExpr{
    public final PersistentVector bindingInits;
    public final Expr body;
    public final boolean isLoop;

    public LetExpr(PersistentVector bindingInits,
                  Expr body,
                  boolean isLoop){
        this.bindingInits = bindingInits;
        this.body = body;
        this.isLoop = isLoop;
    }

    static class Parser implements IParser{
        public Expr parse(C context, Object frm) throws Exception{
            ISeq form = (ISeq) frm;
            //(let [var val var2 val2 ...] body...)
            boolean isLoop = RT.first(form).equals(LOOP);
            if(!(RT.second(form) instanceof IPersistentVector))
                throw new IllegalArgumentException(
                    "Bad binding form, expected vector");

            IPersistentVector bindings =
                (IPersistentVector) RT.second(form);
            if((bindings.count() % 2) != 0)
                throw new IllegalArgumentException(
                    "Bad binding form, expected matched symbol expression pairs");

            ISeq body = RT.next(RT.next(form));
            if(context == C.EVAL

```

```

    || (context == C.EXPRESSION && isLoop))
    return
        analyze(context,
            RT.list(RT.list(FN, PersistentVector.EMPTY,
                form)));

ObjMethod method = (ObjMethod) METHOD.deref();
IPersistentMap backupMethodLocals = method.locals;
IPersistentMap backupMethodIndexLocals =
    method.indexlocals;
PersistentVector recurMismatches = null;

// we might repeat once if a loop with a recurMismatch,
// return breaks
while(true){
    IPersistentMap dynamicBindings =
        RT.map(LOCAL_ENV, LOCAL_ENV.deref(),
            NEXT_LOCAL_NUM, NEXT_LOCAL_NUM.deref());
    method.locals = backupMethodLocals;
    method.indexlocals = backupMethodIndexLocals;

    if(isLoop)
        dynamicBindings =
            dynamicBindings.assoc(LOOP_LOCALS, null);

    try
    {
        Var.pushThreadBindings(dynamicBindings);

        PersistentVector bindingInits =
            PersistentVector.EMPTY;
        PersistentVector loopLocals =
            PersistentVector.EMPTY;
        for(int i = 0; i < bindings.count(); i += 2)
        {
            if(!(bindings.nth(i) instanceof Symbol))
                throw new IllegalArgumentException(
                    "Bad binding form, expected symbol, got: "
                    + bindings.nth(i));
            Symbol sym = (Symbol) bindings.nth(i);
            if(sym.getNamespace() != null)
                throw new Exception(
                    "Can't let qualified name: " + sym);
            Expr init =
                analyze(C.EXPRESSION, bindings.nth(i + 1),
                    sym.name);
            if(isLoop)
            {
                if(recurMismatches != null &&
                    ((LocalBinding)recurMismatches

```

```

        .nth(i/2)).recurMismatch)
    {
    init =
        new StaticMethodExpr("", 0, null,
            RT.class, "box",
            RT.vector(init));
    if(RT.booleanCast(
        RT.WARN_ON_REFLECTION.deref()))
        RT.errPrintWriter().println(
            "Auto-boxing loop arg: " + sym);
    }
    else if(maybePrimitiveType(init)==int.class)
        init =
            new StaticMethodExpr("", 0, null,
                RT.class,
                "longCast",
                RT.vector(init));
    else if(maybePrimitiveType(init) ==
        float.class)
        init =
            new StaticMethodExpr("", 0, null,
                RT.class,
                "doubleCast",
                RT.vector(init));
    }
    //sequential enhancement of env (like Lisp let*)
    LocalBinding lb =
        registerLocal(sym, tagOf(sym), init, false);
    BindingInit bi = new BindingInit(lb, init);
    bindingInits = bindingInits.cons(bi);

    if(isLoop)
        loopLocals = loopLocals.cons(lb);
    }
    if(isLoop)
        LOOP_LOCALS.set(loopLocals);
    Expr bodyExpr;
    try {
        if(isLoop)
            {
            PathNode root =
                new PathNode(PATHTYPE.PATH,
                    (PathNode) CLEAR_PATH.get());
            Var.pushThreadBindings(
                RT.map(CLEAR_PATH,
                    new PathNode(PATHTYPE.PATH,root),
                    CLEAR_ROOT,
                    new PathNode(PATHTYPE.PATH,root),
                    NO_RECUR, null));
            }
        }
    }
}

```

```

        }
        bodyExpr =
            (new BodyExpr.Parser())
                .parse(isLoop
                    ? C.RETURN
                    : context, body);
    }
    finally{
        if(isLoop)
        {
            Var.popThreadBindings();
            recurMismatches = null;
            for(int i = 0;i< loopLocals.count();i++)
            {
                LocalBinding lb =
                    (LocalBinding) loopLocals.nth(i);
                if(lb.recurMismatch)
                    recurMismatches = loopLocals;
            }
        }
        if(recurMismatches == null)
            return
                new LetExpr(bindingInits, bodyExpr, isLoop);
    }
    finally
    {
        Var.popThreadBindings();
    }
}
}

public Object eval() throws Exception{
    throw new UnsupportedOperationException(
        "Can't eval let/loop");
}

public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
    doEmit(context, objx, gen, false);
}

public void emitUnboxed(C context,
                        ObjExpr objx,
                        GeneratorAdapter gen){
    doEmit(context, objx, gen, true);
}

public void doEmit(C context,

```

```

        ObjExpr objx,
        GeneratorAdapter gen,
        boolean emitUnboxed){
    for(int i = 0; i < bindingInits.count(); i++)
    {
        BindingInit bi = (BindingInit) bindingInits.nth(i);
        Class primc = maybePrimitiveType(bi.init);
        if(primc != null)
        {
            ((MaybePrimitiveExpr) bi.init)
                .emitUnboxed(C.EXPRESSION, objx, gen);
            gen.visitVarInsn(
                Type.getType(primc)
                    .getOpcode(Opcodes.ISTORE),
                bi.binding.idx);
        }
        else
        {
            bi.init.emit(C.EXPRESSION, objx, gen);
            gen.visitVarInsn(OBJECT_TYPE.getOpcode(Opcodes.ISTORE),
                bi.binding.idx);
        }
    }
    Label loopLabel = gen.mark();
    if(isLoop)
    {
        try
        {
            Var.pushThreadBindings(RT.map(LOOP_LABEL, loopLabel));
            if	emitUnboxed
                ((MaybePrimitiveExpr)body)
                    .emitUnboxed(context, objx, gen);
            else
                body.emit(context, objx, gen);
        }
        finally
        {
            Var.popThreadBindings();
        }
    }
    else
    {
        if(emitUnboxed)
            ((MaybePrimitiveExpr)body)
                .emitUnboxed(context, objx, gen);
        else
            body.emit(context, objx, gen);
    }
    Label end = gen.mark();
    // gen.visitLocalVariable("this", "Ljava/lang/Object;",
```

```

//                                         null, loopLabel, end, 0);
for(ISeq bis = bindingInits.seq();
    bis != null;
    bis = bis.next())
{
    BindingInit bi = (BindingInit) bis.first();
    String lname = bi.binding.name;
    if(lname.endsWith("__auto__"))
        lname += RT.nextID();
    Class primc = maybePrimitiveType(bi.init);
    if(primc != null)
        gen.visitLocalVariable(lname,
                               Type.getDescriptor(primc),
                               null, loopLabel, end,
                               bi.binding.idx);
    else
        gen.visitLocalVariable(lname, "Ljava/lang/Object;",
                               null, loopLabel, end,
                               bi.binding.idx);
}
}

public boolean hasJavaClass() throws Exception{
    return body.hasJavaClass();
}

public Class getJavaClass() throws Exception{
    return body.getJavaClass();
}

public boolean canEmitPrimitive(){
    return body instanceof MaybePrimitiveExpr &&
           ((MaybePrimitiveExpr)body).canEmitPrimitive();
}

}

public static class RecurExpr implements Expr{
    public final IPersistentVector args;
    public final IPersistentVector loopLocals;
    final int line;
    final String source;

    public RecurExpr(IPersistentVector loopLocals,
                     IPersistentVector args, int line, String source){
        this.loopLocals = loopLocals;
        this.args = args;
        this.line = line;
        this.source = source;
    }
}

```

```

    }

    public Object eval() throws Exception{
        throw new UnsupportedOperationException("Can't eval recur");
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        Label loopLabel = (Label) LOOP_LABEL.deref();
        if(loopLabel == null)
            throw new IllegalStateException();
        for(int i = 0; i < loopLocals.count(); i++)
        {
            LocalBinding lb = (LocalBinding) loopLocals.nth(i);
            Expr arg = (Expr) args.nth(i);
            if(lb.getPrimitiveType() != null)
            {
                Class primc = lb.getPrimitiveType();
                try
                {
                    final Class pc = maybePrimitiveType(arg);
                    if(pc == primc)
                        ((MaybePrimitiveExpr) arg)
                            .emitUnboxed(C.EXPRESSION, objx, gen);
                    else if(primc == long.class && pc == int.class)
                    {
                        ((MaybePrimitiveExpr) arg)
                            .emitUnboxed(C.EXPRESSION, objx, gen);
                        gen.visitInsn(I2L);
                    }
                    else if(primc == double.class && pc == float.class)
                    {
                        ((MaybePrimitiveExpr) arg)
                            .emitUnboxed(C.EXPRESSION, objx, gen);
                        gen.visitInsn(F2D);
                    }
                    else if(primc == int.class && pc == long.class)
                    {
                        ((MaybePrimitiveExpr) arg)
                            .emitUnboxed(C.EXPRESSION, objx, gen);
                        gen.invokeStatic(RT_TYPE,
                            Method.getMethod("int intCast(long)"));
                    }
                    else if(primc == float.class && pc == double.class)
                    {
                        ((MaybePrimitiveExpr) arg)
                            .emitUnboxed(C.EXPRESSION, objx, gen);
                        gen.visitInsn(D2F);
                    }
                    else
                    {

```

```

//RT.booleanCast(RT.WARN_ON_REFLECTION.deref()))
//          if(true)
//              throw new IllegalArgumentException
//          RT.errPrintWriter().println
//          (//source + ":" + line +
//           " recur arg for primitive local: " +
//           lb.name +
//           " is not matching primitive, had: " +
//           (arg.hasJavaClass()
//            ? arg.getJavaClass().getName()
//            :"Object") +
//           ", needed: " +
//           primc.getName());
//          arg.emit(C.EXPRESSION, objx, gen);
//          HostExpr.emitUnboxArg(objx,gen,primc);
//      }
//  }
catch(Exception e)
{
    throw new RuntimeException(e);
}
else
{
    arg.emit(C.EXPRESSION, objx, gen);
}
}

for(int i = loopLocals.count() - 1; i >= 0; i--)
{
    LocalBinding lb = (LocalBinding) loopLocals.nth(i);
    Class primc = lb.getPrimitiveType();
    if(lb.isArg)
        gen.storeArg(lb.idx-(objx.isStatic?0:1));
    else
    {
        if(primc != null)
            gen.visitVarInsn(
                Type.getType(primc)
                .getOpcode(OpCodes.ISTORE), lb.idx);
        else
            gen.visitVarInsn(
                OBJECT_TYPE
                .getOpcode(OpCodes.ISTORE), lb.idx);
    }
}
gen.goTo(loopLabel);
}

```

```

public boolean hasJavaClass() throws Exception{
    return true;
}

public Class getJavaClass() throws Exception{
    return null;
}

static class Parser implements IParser{
    public Expr parse(C context, Object frm) throws Exception{
        int line = (Integer) LINE.deref();
        String source = (String) SOURCE.deref();

        ISeq form = (ISeq) frm;
        IPersistentVector loopLocals =
            (IPersistentVector) LOOP_LOCALS.deref();
        if(context != C.RETURN || loopLocals == null)
            throw new UnsupportedOperationException(
                "Can only recur from tail position");
        if(IN_CATCH_FINALLY.deref() != null)
            throw new UnsupportedOperationException(
                "Cannot recur from catch/finally");
        if(NO_RECUR.deref() != null)
            throw new UnsupportedOperationException(
                "Cannot recur across try");
        PersistentVector args = PersistentVector.EMPTY;
        for(ISeq s = RT.seq(form.next()); s != null; s = s.next())
        {
            args = args.cons(analyze(C.EXPRESSION, s.first()));
        }
        if(args.count() != loopLocals.count())
            throw new IllegalArgumentException(
                String.format(
                    "Mismatched argument count to recur, expected: %d args, got: %d",
                    loopLocals.count(), args.count()));
        for(int i = 0;i< loopLocals.count();i++)
        {
            LocalBinding lb = (LocalBinding) loopLocals.nth(i);
            Class primc = lb.getPrimitiveType();
            if(primc != null)
            {
                boolean mismatch = false;
                final Class pc =
                    maybePrimitiveType((Expr) args.nth(i));
                if(primc == long.class)
                {
                    if(!(pc == long.class
                        || pc == int.class
                        || pc == short.class
                        || pc == char.class
                        || pc == double.class
                        || pc == float.class
                        || pc == byte.class
                        || pc == short.class
                        || pc == boolean.class
                        || pc == long.class))
                        mismatch = true;
                }
                if(mismatch)
                    throw new UnsupportedOperationException(
                        "Cannot recur from tail position");
            }
        }
    }
}

```

```

        || pc == byte.class))
        mismatch = true;
    }
    else if(primc == double.class)
    {
        if(!(pc == double.class
        || pc == float.class))
        mismatch = true;
    }
    if(mismatch)
    {
        lb.recurMistmatch = true;
        if(RT.booleanCast(RT.WARN_ON_REFLECTION.deref()))
            RT.errPrintWriter().println
            (source + ":" + line +
             " recur arg for primitive local: " +
             lb.name +
             " is not matching primitive, had: " +
             (pc != null ? pc.getName():"Object") +
             ", needed: " +
             primc.getName());
    }
}
return new RecurExpr(loopLocals, args, line, source);
}
}
}

private static LocalBinding registerLocal(Symbol sym,
                                         Symbol tag,
                                         Expr init,
                                         boolean isArg)
throws Exception{
    int num = getAndIncLocalNum();
    LocalBinding b =
        new LocalBinding(num, sym, tag, init, isArg, clearPathRoot());
    IPersistentMap localsMap = (IPersistentMap) LOCAL_ENV.deref();
    LOCAL_ENV.set(RT.assoc(localsMap, b.sym, b));
    ObjMethod method = (ObjMethod) METHOD.deref();
    method.locals = (IPersistentMap) RT.assoc(method.locals, b, b);
    method.indexlocals =
        (IPersistentMap) RT.assoc(method.indexlocals, num, b);
    return b;
}

private static int getAndIncLocalNum(){
    int num = ((Number) NEXT_LOCAL_NUM.deref()).intValue();
    ObjMethod m = (ObjMethod) METHOD.deref();
    if(num > m.maxLocal)

```

```

        m.maxLocal = num;
        NEXT_LOCAL_NUM.set(num + 1);
        return num;
    }

    public static Expr analyze(C context, Object form) throws Exception{
        return analyze(context, form, null);
    }

    private static Expr analyze(C context,
                                Object form,
                                String name)
    throws Exception{
        //todo symbol macro expansion?
        try
        {
            if(form instanceof LazySeq)
            {
                form = RT.seq(form);
                if(form == null)
                    form = PersistentList.EMPTY;
            }
            if(form == null)
                return NIL_EXPR;
            else if(form == Boolean.TRUE)
                return TRUE_EXPR;
            else if(form == Boolean.FALSE)
                return FALSE_EXPR;
            Class fclass = form.getClass();
            if(fclass == Symbol.class)
                return analyzeSymbol((Symbol) form);
            else if(fclass == Keyword.class)
                return registerKeyword((Keyword) form);
            else if(form instanceof Number)
                return NumberExpr.parse((Number) form);
            else if(fclass == String.class)
                return new StringExpr(((String) form).intern());
//            else if(fclass == Character.class)
//                return new CharExpr((Character) form);
            else if(form instanceof IPersistentCollection &&
                    ((IPersistentCollection) form).count() == 0)
            {
                Expr ret = new EmptyExpr(form);
                if(RT.meta(form) != null)
                    ret = new MetaExpr(ret, MapExpr
                        .parse(context == C.EVAL
                            ? context
                            : C.EXPRESSION,
                        ((IObj) form).meta()));
            }
        }
        return ret;
    }
}

```

```

        }
    else if(form instanceof ISeq)
        return analyzeSeq(context, (ISeq) form, name);
    else if(form instanceof IPersistentVector)
        return VectorExpr.parse(context,
                               (IPersistentVector) form);
    else if(form instanceof IPersistentMap)
        return MapExpr.parse(context, (IPersistentMap) form);
    else if(form instanceof IPersistentSet)
        return SetExpr.parse(context, (IPersistentSet) form);

//    else
//        throw new UnsupportedOperationException();
//    return new ConstantExpr(form);
}

catch(Throwable e)
{
    if(!(e instanceof CompilerException))
        throw new CompilerException(
            (String) SOURCE_PATH.deref(), (Integer) LINE.deref(), e);
    else
        throw (CompilerException) e;
}
}

static public class CompilerException extends Exception{
    final public String source;

    public CompilerException(String source, int line, Throwable cause){
        super(errorMsg(source, line, cause.toString()), cause);
        this.source = source;
    }

    public String toString(){
        return getMessage();
    }
}

static public Var isMacro(Object op) throws Exception{
    //no local macros for now
    if(op instanceof Symbol && referenceLocal((Symbol) op) != null)
        return null;
    if(op instanceof Symbol || op instanceof Var)
    {
        Var v = (op instanceof Var)
            ? (Var) op
            : lookupVar((Symbol) op, false);
        if(v != null && v.isMacro())
        {
            if(v.ns != currentNS() && !v.isPublic())

```

```

        throw new IllegalStateException(
            "var: " + v + " is not public");
    return v;
}
return null;
}

static public IFn isInline(Object op, int arity) throws Exception{
    //no local inlines for now
    if(op instanceof Symbol && referenceLocal((Symbol) op) != null)
        return null;
    if(op instanceof Symbol || op instanceof Var)
    {
        Var v = (op instanceof Var)
            ? (Var) op
            : lookupVar((Symbol) op, false);
        if(v != null)
        {
            if(v.ns != currentNS() && !v.isPublic())
                throw new IllegalStateException(
                    "var: " + v + " is not public");
            IFn ret = (IFn) RT.get(v.meta(), inlineKey);
            if(ret != null)
            {
                IFn arityPred = (IFn) RT.get(v.meta(), inlineAritiesKey);
                if(arityPred == null ||
                   RT.booleanCast(arityPred.invoke(arity)))
                    return ret;
            }
        }
        return null;
    }
    public static boolean namesStaticMember(Symbol sym){
        return sym.ns != null && namespaceFor(sym) == null;
    }

    public static Object preserveTag(ISeq src, Object dst) {
        Symbol tag = tagOf(src);
        if (tag != null && dst instanceof IObj) {
            IPersistentMap meta = RT.meta(dst);
            return ((IObj) dst)
                .withMeta(
                    (IPersistentMap) RT.assoc(meta, RT.TAG_KEY, tag));
        }
        return dst;
    }
}

```

```

public static Object macroexpand1(Object x) throws Exception{
    if(x instanceof ISeq)
    {
        ISeq form = (ISeq) x;
        Object op = RT.first(form);
        if(isSpecial(op))
            return x;
        //macro expansion
        Var v = isMacro(op);
        if(v != null)
        {
            try
            {
                return
                    v.applyTo(
                        RT.cons(
                            form,
                            RT.cons(LOCAL_ENV.get(),form.next())));
            }
            catch(ArityException e)
            {
                // hide the 2 extra params for a macro
                throw new ArityException(e.actual - 2, e.name);
            }
        }
        else
        {
            if(op instanceof Symbol)
            {
                Symbol sym = (Symbol) op;
                String sname = sym.name;
                //(.substring s 2 5) => (. s substring 2 5)
                if(sym.name.charAt(0) == '.')
                {
                    if(RT.length(form) < 2)
                        throw new IllegalArgumentException(
                            "Malformed member expression, expecting (.member target ...)");
                    Symbol meth = Symbol.intern(sname.substring(1));
                    Object target = RT.second(form);
                    if(HostExpr.maybeClass(target, false) != null)
                    {
                        target =
                            ((IObj)RT.list(IDENTITY, target))
                                .withMeta(RT.map(RT.TAG_KEY,CLASS));
                    }
                }
                return
                    preserveTag(form,
                        RT.listStar(DOT,
                            target,
                            meth,

```

```

                form.next().next());
}
else if(namesStaticMember(sym))
{
    Symbol target = Symbol.intern(sym.ns);
    Class c = HostExpr.maybeClass(target, false);
    if(c != null)
    {
        Symbol meth = Symbol.intern(sym.name);
        return
            preserveTag(form,
                RT.listStar(DOT,
                    target,
                    meth,
                    form.next()));
    }
}
else
{
    //((s.substring 2 5) => (. s substring 2 5)
    //also (package.class.name ...) (. package.class name ...)
    int idx = sname.lastIndexOf('.');
    //
    if(idx > 0 && idx < sname.length() - 1)
    {
        //
        Symbol target = Symbol.intern(sname.substring(0, idx));
        Symbol meth = Symbol.intern(sname.substring(idx + 1));
        return RT.listStar(DOT, target, meth, form.rest());
    }
    //((StringBuilder. "foo") => (new StringBuilder "foo")
    //else
    if(idx == sname.length() - 1)
        return
            RT.listStar(NEW,
                Symbol.intern(sname.substring(0, idx)),
                form.next());
    }
}
return x;
}

static Object macroexpand(Object form) throws Exception{
    Object exf = macroexpand1(form);
    if(exf != form)
        return macroexpand(exf);
    return form;
}

private static Expr analyzeSeq(C context,

```

```

        ISeq form,
        String name)
throws Exception{
    Integer line = (Integer) LINE.deref();
    if(RT.meta(form) != null && RT.meta(form).containsKey(RT.LINE_KEY))
        line = (Integer) RT.meta(form).valAt(RT.LINE_KEY);
    Var.pushThreadBindings(
        RT.map(LINE, line));
    try
    {
        Object me = macroexpand1(form);
        if(me != form)
            return analyze(context, me, name);

        Object op = RT.first(form);
        if(op == null)
            throw new IllegalArgumentException("Can't call nil");
        IFn inline = isInline(op, RT.count(RT.next(form)));
        if(inline != null)
            return
                analyze(context,
                    preserveTag(form, inline.applyTo(RT.next(form))));
        IParser p;
        if(op.equals(FN))
            return FnExpr.parse(context, form, name);
        else if((p = (IParser) specials.valAt(op)) != null)
            return p.parse(context, form);
        else
            return InvokeExpr.parse(context, form);
    }
    catch(Throwable e)
    {
        if(!(e instanceof CompilerException))
            throw new CompilerException(
                (String) SOURCE_PATH.deref(), (Integer) LINE.deref(), e);
        else
            throw (CompilerException) e;
    }
    finally
    {
        Var.popThreadBindings();
    }
}

static String errorMsg(String source, int line, String s){
    return String.format("%s, compiling:(%s:%d)", s, source, line);
}

public static Object eval(Object form) throws Exception{
    return eval(form, true);
}

```

```

    }

public static Object eval(Object form, boolean freshLoader)
throws Exception{
    boolean createdLoader = false;
    if(true)//!LOADER.isBound())
    {
        Var.pushThreadBindings(RT.map(LOADER, RT.makeClassLoader()));
        createdLoader = true;
    }
    try
    {
        Integer line = (Integer) LINE.deref();
        if(RT.meta(form) != null &&
           RT.meta(form).containsKey(RT.LINE_KEY))
            line = (Integer) RT.meta(form).valAt(RT.LINE_KEY);
        Var.pushThreadBindings(RT.map(LINE, line));
        try
        {
            form = macroexpand(form);
            if(form instanceof IPersistentCollection &&
               Util.equals(RT.first(form), DO))
            {
                ISeq s = RT.next(form);
                for(; RT.next(s) != null; s = RT.next(s))
                    eval(RT.first(s), false);
                return eval(RT.first(s), false);
            }
            else if(form instanceof IPersistentCollection
                   && !(RT.first(form) instanceof Symbol
                   && ((Symbol) RT.first(form)).name
                           .startsWith("def")))
            {
                ObjExpr fexpr =
                    (ObjExpr) analyze(C.EXPRESSION,
                                      RT.list(FN, PersistentVector.EMPTY, form),
                                      "eval" + RT.nextID());
                IFn fn = (IFn) fexpr.eval();
                return fn.invoke();
            }
            else
            {
                Expr expr = analyze(C.EVAL, form);
                return expr.eval();
            }
        }
        catch(Throwable e)
        {
            if(!(e instanceof Exception))
                throw new RuntimeException(e);
        }
    }
}

```

```

        throw (Exception)e;
    }
    finally
    {
        Var.popThreadBindings();
    }
}

finally
{
    if(createdLoader)
        Var.popThreadBindings();
}
}

private static int registerConstant(Object o){
    if(!CONSTANTS.isBound())
        return -1;
    PersistentVector v = (PersistentVector) CONSTANTS.deref();
    IdentityHashMap<Object, Integer> ids =
        (IdentityHashMap<Object, Integer>) CONSTANT_IDS.deref();
    Integer i = ids.get(o);
    if(i != null)
        return i;
    CONSTANTS.set(RT.conj(v, o));
    ids.put(o, v.count());
    return v.count();
}

private static KeywordExpr registerKeyword(Keyword keyword){
    if(!KEYWORDS.isBound())
        return new KeywordExpr(keyword);

    IPersistentMap keywordsMap = (IPersistentMap) KEYWORDS.deref();
    Object id = RT.get(keywordsMap, keyword);
    if(id == null)
    {
        KEYWORDS.set(RT.assoc(keywordsMap,
            keyword,
            registerConstant(keyword)));
    }
    return new KeywordExpr(keyword);
//    KeywordExpr ke = (KeywordExpr) RT.get(keywordsMap, keyword);
//    if(ke == null)
//        KEYWORDS.set(
//            RT.assoc(keywordsMap, keyword,
//                ke = new KeywordExpr(keyword)));
//    return ke;
}
}

```

```

private static int registerKeywordCallsite(Keyword keyword){
    if(!KEYWORD_CALLSITES.isBound())
        throw new IllegalAccessError("KEYWORD_CALLSITES is not bound");

    IPersistentVector keywordCallsites =
        (IPersistentVector) KEYWORD_CALLSITES.deref();

    keywordCallsites = keywordCallsites.cons(keyword);
    KEYWORD_CALLSITES.set(keywordCallsites);
    return keywordCallsites.count()-1;
}

private static int registerProtocolCallsite(Var v){
    if(!PROTOCOL_CALLSITES.isBound())
        throw new IllegalAccessError("PROTOCOL_CALLSITES is not bound");

    IPersistentVector protocolCallsites =
        (IPersistentVector) PROTOCOL_CALLSITES.deref();

    protocolCallsites = protocolCallsites.cons(v);
    PROTOCOL_CALLSITES.set(protocolCallsites);
    return protocolCallsites.count()-1;
}

private static void registerVarCallsite(Var v){
    if(!VAR_CALLSITES.isBound())
        throw new IllegalAccessError("VAR_CALLSITES is not bound");

    IPersistentCollection varCallsites =
        (IPersistentCollection) VAR_CALLSITES.deref();

    varCallsites = varCallsites.cons(v);
    VAR_CALLSITES.set(varCallsites);
//    return varCallsites.count()-1;
}

static ISeq fwdPath(PathNode p1){
    ISeq ret = null;
    for(;p1 != null;p1 = p1.parent)
        ret = RT.cons(p1,ret);
    return ret;
}

static PathNode commonPath(PathNode n1, PathNode n2){
    ISeq xp = fwdPath(n1);
    ISeq yp = fwdPath(n2);
    if(RT.first(xp) != RT.first(yp))
        return null;
    while(RT.second(xp) != null && RT.second(xp) == RT.second(yp))
    {

```

```

        xp = xp.next();
        yp = yp.next();
    }
    return (PathNode) RT.first(xp);
}

static void addAnnotation(Object visitor, IPersistentMap meta){
    try{
        if(meta != null && ADD_ANNOTATIONS.isBound())
            ADD_ANNOTATIONS.invoke(visitor, meta);
    }
    catch (Exception e)
    {
        throw new RuntimeException(e);
    }
}

static void addParameterAnnotation(Object visitor,
                                  IPersistentMap meta,
                                  int i){
    try{
        if(meta != null && ADD_ANNOTATIONS.isBound())
            ADD_ANNOTATIONS.invoke(visitor, meta, i);
    }
    catch (Exception e)
    {
        throw new RuntimeException(e);
    }
}

private static Expr analyzeSymbol(Symbol sym) throws Exception{
    Symbol tag = tagOf(sym);
    if(sym.ns == null) //ns-qualified syms are always Vars
    {
        LocalBinding b = referenceLocal(sym);
        if(b != null)
        {
            return new LocalBindingExpr(b, tag);
        }
    }
    else
    {
        if(namespaceFor(sym) == null)
        {
            Symbol nsSym = Symbol.intern(sym.ns);
            Class c = HostExpr.maybeClass(nsSym, false);
            if(c != null)
            {
                if(Reflector.getField(c, sym.name, true) != null)
                    return new StaticFieldExpr(

```

```

        (Integer) LINE.deref(), c, sym.name, tag);
        throw new Exception(
            "Unable to find static field: " + sym.name + " in " + c);
    }
}
}

//Var v = lookupVar(sym, false);
//  Var v = lookupVar(sym, false);
//  if(v != null)
//      return new VarExpr(v, tag);
Object o = resolve(sym);
if(o instanceof Var)
{
    Var v = (Var) o;
    if(isMacro(v) != null)
        throw new Exception("Can't take value of a macro: " + v);
    registerVar(v);
    return new VarExpr(v, tag);
}
else if(o instanceof Class)
    return new ConstantExpr(o);
else if(o instanceof Symbol)
    return new UnresolvedVarExpr((Symbol) o);

throw new Exception(
    "Unable to resolve symbol: " + sym + " in this context");
}

static String destubClassName(String className){
    //skip over prefix + '.' or '/'
    if(className.startsWith(COMPILATION_STUB_PREFIX))
        return className.substring(COMPILATION_STUB_PREFIX.length()+1);
    return className;
}

static Type getType(Class c){
    String descriptor = Type.getType(c).getDescriptor();
    if(descriptor.startsWith("L"))
        descriptor = "L" + destubClassName(descriptor.substring(1));
    return Type.getType(descriptor);
}

static Object resolve(Symbol sym, boolean allowPrivate)
throws Exception{
    return resolveIn(currentNS(), sym, allowPrivate);
}

static Object resolve(Symbol sym) throws Exception{
    return resolveIn(currentNS(), sym, false);
}

```

```

}

static Namespace namespaceFor(Symbol sym){
    return namespaceFor(currentNS(), sym);
}

static Namespace namespaceFor(Namespace inns, Symbol sym){
    //note, presumes non-nil sym.ns
    // first check against currentNS' aliases...
    Symbol nsSym = Symbol.intern(sym.ns);
    Namespace ns = inns.lookupAlias(nsSym);
    if(ns == null)
    {
        // ...otherwise check the Namespaces map.
        ns = Namespace.find(nsSym);
    }
    return ns;
}

static public Object resolveIn(Namespace n,
                               Symbol sym,
                               boolean allowPrivate)
throws Exception{
    //note - ns-qualified vars must already exist
    if(sym.ns != null)
    {
        Namespace ns = namespaceFor(n, sym);
        if(ns == null)
            throw new Exception("No such namespace: " + sym.ns);

        Var v = ns.findInternedVar(Symbol.intern(sym.name));
        if(v == null)
            throw new Exception("No such var: " + sym);
        else if(v.ns != currentNS() && !v.isPublic() && !allowPrivate)
            throw new IllegalStateException(
                "var: " + sym + " is not public");
        return v;
    }
    else if(sym.name.indexOf('.') > 0 || sym.name.charAt(0) == '[')
    {
        return RT.classForName(sym.name);
    }
    else if(sym.equals(NS))
        return RT.NS_VAR;
    else if(sym.equals(IN_NS))
        return RT.IN_NS_VAR;
    else
    {
        if(Util.equals(sym, COMPILE_STUB_SYM.get()))
            return COMPILE_STUB_CLASS.get();
    }
}

```

```

Object o = n.getMapping(sym);
if(o == null)
{
    if(RT.booleanCast(RT.ALLOW_UNRESOLVED_VARS.deref()))
    {
        return sym;
    }
    else
    {
        throw new Exception(
            "Unable to resolve symbol: " + sym +
            " in this context");
    }
}
return o;
}

static public Object maybeResolveIn(Namespace n, Symbol sym)
throws Exception{
    //note - ns-qualified vars must already exist
    if(sym.ns != null)
    {
        Namespace ns = namespaceFor(n, sym);
        if(ns == null)
            return null;
        Var v = ns.findInternedVar(Symbol.intern(sym.name));
        if(v == null)
            return null;
        return v;
    }
    else if(sym.name.indexOf('.') > 0 && !sym.name.endsWith(".")
           || sym.name.charAt(0) == '[')
    {
        return RT.classForName(sym.name);
    }
    else if(sym.equals(NS))
        return RT.NS_VAR;
    else if(sym.equals(IN_NS))
        return RT.IN_NS_VAR;
    else
    {
        Object o = n.getMapping(sym);
        return o;
    }
}

static Var lookupVar(Symbol sym, boolean internNew) throws Exception{

```

```

Var var = null;

//note - ns-qualified vars in other namespaces must already exist
if(sym.ns != null)
{
    Namespace ns = namespaceFor(sym);
    if(ns == null)
        return null;
    //throw new Exception("No such namespace: " + sym.ns);
    Symbol name = Symbol.intern(sym.name);
    if(internNew && ns == currentNS())
        var = currentNS().intern(name);
    else
        var = ns.findInternedVar(name);
}
else if(sym.equals(NS))
    var = RT.NS_VAR;
else if(sym.equals(IN_NS))
    var = RT.IN_NS_VAR;
else
{
    //is it mapped?
    Object o = currentNS().getMapping(sym);
    if(o == null)
    {
        //introduce a new var in the current ns
        if(internNew)
            var = currentNS().intern(Symbol.intern(sym.name));
    }
    else if(o instanceof Var)
    {
        var = (Var) o;
    }
    else
    {
        throw new Exception(
            "Expecting var, but " + sym + " is mapped to " + o);
    }
}
if(var != null)
    registerVar(var);
return var;
}

private static void registerVar(Var var) throws Exception{
    if(!VARS.isBound())
        return;
    IPersistentMap varsMap = (IPersistentMap) VARS.deref();
    Object id = RT.get(varsMap, var);
    if(id == null)

```

```

        {
            VARS.set(RT.assoc(varsMap, var, registerConstant(var)));
        }
    //    if(varsMap != null && RT.get(varsMap, var) == null)
    //        VARS.set(RT.assoc(varsMap, var, var));
    }

    static Namespace currentNS(){
        return (Namespace) RT.CURRENT_NS.deref();
    }

    static void closeOver(LocalBinding b, ObjMethod method){
        if(b != null && method != null)
        {
            if(RT.get(method.locals, b) == null)
            {
                method.objx.closes =
                    (IPersistentMap) RT.assoc(method.objx.closes, b, b);
                closeOver(b, method.parent);
            }
            else if(IN_CATCH_FINALLY.deref() != null)
            {
                method.localsUsedInCatchFinally =
                    (PersistentHashSet)
                        method.localsUsedInCatchFinally.cons(b.idx);
            }
        }
    }

    static LocalBinding referenceLocal(Symbol sym) throws Exception{
        if(!LOCAL_ENV.isBound())
            return null;
        LocalBinding b = (LocalBinding) RT.get(LOCAL_ENV.deref(), sym);
        if(b != null)
        {
            ObjMethod method = (ObjMethod) METHOD.deref();
            closeOver(b, method);
        }
        return b;
    }

    private static Symbol tagOf(Object o){
        Object tag = RT.get(RT.meta(o), RT.TAG_KEY);
        if(tag instanceof Symbol)
            return (Symbol) tag;
        else if(tag instanceof String)
            return Symbol.intern(null, (String) tag);
        return null;
    }
}

```

```

public static Object loadFile(String file) throws Exception{
//    File fo = new File(file);
//    if(!fo.exists())
//        return null;

    FileInputStream f = new FileInputStream(file);
    try
    {
        return load(new InputStreamReader(f, RT.UTF8),
                    new File(file).getAbsolutePath(),
                    (new File(file)).getName());
    }
    finally
    {
        f.close();
    }
}

public static Object load(Reader rdr) throws Exception{
    return load(rdr, null, "NO_SOURCE_FILE");
}

public static Object load(Reader rdr,
                        String sourcePath,
                        String sourceName)
throws Exception{
    Object EOF = new Object();
    Object ret = null;
    LineNumberingPushbackReader pushbackReader =
        (rdr instanceof LineNumberingPushbackReader)
        ? (LineNumberingPushbackReader) rdr
        : new LineNumberingPushbackReader(rdr);
    Var.pushThreadBindings(
        RT.map(LOADER, RT.makeClassLoader(),
               SOURCE_PATH, sourcePath,
               SOURCE, sourceName,
               METHOD, null,
               LOCAL_ENV, null,
               LOOP_LOCALS, null,
               NEXT_LOCAL_NUM, 0,
               RT.CURRENT_NS, RT.CURRENT_NS.deref(),
               LINE_BEFORE, pushbackReader.getLineNumber(),
               LINE_AFTER, pushbackReader.getLineNumber()
               ,UNCHECKED_MATH, UNCHECKED_MATH.deref()
               ,RT.WARN_ON_REFLECTION, RT.WARN_ON_REFLECTION.deref()
        ));
}

try
{

```

```

        for(Object r =
            LispReader.read(pushbackReader, false, EOF, false); r != EOF;
            r = LispReader.read(pushbackReader, false, EOF, false))
        {
            LINE_AFTER.set(pushbackReader.getLineNumber());
            ret = eval(r,false);
            LINE_BEFORE.set(pushbackReader.getLineNumber());
        }
    }
    catch(LispReader.ReaderException e)
    {
        throw new CompilerException(sourcePath, e.line, e.getCause());
    }
    finally
    {
        Var.popThreadBindings();
    }
    return ret;
}

static public void writeClassFile(String internalName,
                                  byte[] bytecode)
throws Exception{
    String genPath = (String) COMPILE_PATH.deref();
    if(genPath == null)
        throw new Exception("*compile-path* not set");
    String[] dirs = internalName.split("/");
    String p = genPath;
    for(int i = 0; i < dirs.length - 1; i++)
    {
        p += File.separator + dirs[i];
        (new File(p)).mkdir();
    }
    String path = genPath + File.separator + internalName + ".class";
    File cf = new File(path);
    cf.createNewFile();
    FileOutputStream cfs = new FileOutputStream(cf);
    try
    {
        cfs.write(bytecode);
        cfs.flush();
        cfs.getFD().sync();
    }
    finally
    {
        cfs.close();
    }
}

public static void pushNS(){
}

```

```

Var.pushThreadBindings(
    PersistentHashMap.create(
        Var.intern(
            Symbol.intern("clojure.core"),
            Symbol.intern("*ns*").setDynamic(), null)));
}

public static ILookupThunk getLookupThunk(Object target, Keyword k){
//To change body of created methods use File | Settings | File Templates.
    return null;
}

static void compile1(GeneratorAdapter gen,
                     ObjExpr objx,
                     Object form)
throws Exception{
    Integer line = (Integer) LINE.deref();
    if(RT.meta(form) != null && RT.meta(form).containsKey(RT.LINE_KEY))
        line = (Integer) RT.meta(form).valAt(RT.LINE_KEY);
    Var.pushThreadBindings(
        RT.map(LINE, line
              ,LOADER, RT.makeClassLoader()
              ));
    try
    {
        form = macroexpand(form);
        if(form instanceof IPersistentCollection &&
           Util.equals(RT.first(form), D0))
        {
            for(ISeq s = RT.next(form); s != null; s = RT.next(s))
            {
                compile1(gen, objx, RT.first(s));
            }
        }
        else
        {
            Expr expr = analyze(C.EVAL, form);
            objx.keywords = (IPersistentMap) KEYWORDS.deref();
            objx.vars = (IPersistentMap) VARS.deref();
            objx.constants = (PersistentVector) CONSTANTS.deref();
            expr.emit(C.EXPRESSION, objx, gen);
            expr.eval();
        }
    }
    finally
    {
        Var.popThreadBindings();
    }
}
}

```

```

public static Object compile(Reader rdr,
                           String sourcePath,
                           String sourceName)
throws Exception{
    if(COMPILATION_PATH.deref() == null)
        throw new Exception("*compile-path* not set");

    Object EOF = new Object();
    Object ret = null;
    LineNumberingPushbackReader pushbackReader =
        (rdr instanceof LineNumberingPushbackReader)
        ? (LineNumberingPushbackReader) rdr
        : new LineNumberingPushbackReader(rdr);
    Var.pushThreadBindings(
        RT.map(SOURCE_PATH, sourcePath,
               SOURCE, sourceName,
               METHOD, null,
               LOCAL_ENV, null,
               LOOP_LOCALS, null,
               NEXT_LOCAL_NUM, 0,
               RT.CURRENT_NS, RT.CURRENT_NS.deref(),
               LINE_BEFORE, pushbackReader.getLineNumber(),
               LINE_AFTER, pushbackReader.getLineNumber(),
               CONSTANTS, PersistentVector.EMPTY,
               CONSTANT_IDS, new IdentityHashMap(),
               KEYWORDS, PersistentHashMap.EMPTY,
               VARS, PersistentHashMap.EMPTY
               ,UNCHECKED_MATH, UNCHECKED_MATH.deref()
               ,RT.WARN_ON_REFLECTION, RT.WARN_ON_REFLECTION.deref()
               // ,LOADER, RT.makeClassLoader()
        ));
}

try
{
    //generate loader class
    ObjExpr objx = new ObjExpr(null);
    objx.internalName =
        sourcePath.replace(File.separator, "/")
        .substring(0, sourcePath.lastIndexOf('.'))
        + RT.LOADER_SUFFIX;

    objx.objtype = Type.getObjectType(objx.internalName);
    ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAXS);
    ClassVisitor cv = cw;
    cv.visit(V1_5, ACC_PUBLIC + ACC_SUPER,
             objx.internalName, null,
             "java/lang/Object", null);

    //static load method
    GeneratorAdapter gen =

```

```

        new GeneratorAdapter(ACC_PUBLIC + ACC_STATIC,
            Method.getMethod("void load ()"),
            null,
            null,
            cv);
    gen.visitCode();

    for(Object r =
        LispReader.read(pushbackReader, false, EOF, false);
        r != EOF;
        r = LispReader.read(pushbackReader, false, EOF, false))
    {
        LINE_AFTER.set(pushbackReader.getLineNumber());
        compile1(gen, objx, r);
        LINE_BEFORE.set(pushbackReader.getLineNumber());
    }
    //end of load
    gen.returnValue();
    gen.endMethod();

    //static fields for constants
    for(int i = 0; i < objx.constants.count(); i++)
    {
        cv.visitField(ACC_PUBLIC + ACC_FINAL + ACC_STATIC,
            objx.constantName(i),
            objx.constantType(i).getDescriptor(),
            null, null);
    }

    final int INITS_PER = 100;
    int numInits =  objx.constants.count() / INITS_PER;
    if(objx.constants.count() % INITS_PER != 0)
        ++numInits;

    for(int n = 0;n<numInits;n++)
    {
        GeneratorAdapter clinitgen =
            new GeneratorAdapter(ACC_PUBLIC + ACC_STATIC,
                Method.getMethod("void __init" + n + "()"),
                null,
                null,
                cv);
        clinitgen.visitCode();
        try
        {
            Var.pushThreadBindings(RT.map(RT.PRINT_DUP, RT.T));
            for(int i = n*INITS_PER;
                i < objx.constants.count() &&
                i < (n+1)*INITS_PER;

```

```

        i++)
    {
        objx.emitValue(objx.constants.nth(i), clinitgen);
        clinitgen.checkCast(objx.constantType(i));
        clinitgen.putStatic(
            objx.objtype,
            objx.constantName(i),
            objx.constantType(i));
    }
}
finally
{
    Var.popThreadBindings();
}
clinitgen.returnValue();
clinitgen.endMethod();
}

//static init for constants, keywords and vars
GeneratorAdapter clinitgen =
    new GeneratorAdapter(ACC_PUBLIC + ACC_STATIC,
        Method.getMethod("void <clinit> ()"),
        null,
        null,
        cv);
clinitgen.visitCode();
Label startTry = clinitgen.newLabel();
Label endTry = clinitgen.newLabel();
Label end = clinitgen.newLabel();
Label finallyLabel = clinitgen.newLabel();

//        if(objx.constants.count() > 0)
//        {
//            objx.emitConstants(clinitgen);
//        }
for(int n = 0;n<numInits;n++)
    clinitgen.invokeStatic(
        objx.objtype,
        Method.getMethod("void __init" + n + "()"));

    clinitgen.invokeStatic(Type.getType(Compiler.class),
        Method.getMethod("void pushNS()"));
    clinitgen.mark(startTry);
    clinitgen.invokeStatic(objx.objtype,
        Method.getMethod("void load()"));
    clinitgen.mark(endTry);
    clinitgen.invokeStatic(VAR_TYPE,
        Method.getMethod("void popThreadBindings()"));
    clinitgen.goTo(end);
}

```

```

        clinitgen.mark(finallyLabel);
        //exception should be on stack
        clinitgen.invokeStatic(VAR_TYPE,
            Method.getMethod("void popThreadBindings()"));
        clinitgen.throwException();
        clinitgen.mark(end);
        clinitgen.visitTryCatchBlock(startTry, endTry,
            finallyLabel, null);

        //end of static init
        clinitgen.returnValue();
        clinitgen.endMethod();

        //end of class
        cv.visitEnd();

        writeClassFile(objx.internalName, cw.toByteArray());
    }
    catch(LispReader.ReaderException e)
    {
        throw new CompilerException(sourcePath, e.line, e.getCause());
    }
    finally
    {
        Var.popThreadBindings();
    }
    return ret;
}

static public class NewInstanceExpr extends ObjExpr{
    //IPersistentMap optionsMap = PersistentArrayMap.EMPTY;
    IPersistentCollection methods;

    Map<IPersistentVector,java.lang.reflect.Method> mmap;
    Map<IPersistentVector,Set<Class>> covariants;

    public NewInstanceExpr(Object tag){
        super(tag);
    }

    static class DeftypeParser implements IParser{
        public Expr parse(C context, final Object frm) throws Exception{
            ISeq rform = (ISeq) frm;
            //(deftype* tagname classname [fields]
            //  :implements [interfaces] :tag tagname methods*)
            rform = RT.next(rform);
            String tagname = ((Symbol) rform.first()).toString();
            rform = rform.next();
            Symbol classname = (Symbol) rform.first();
        }
    }
}

```

```

rform = rform.next();
IPersistentVector fields = (IPersistentVector) rform.first();
rform = rform.next();
IPersistentMap opts = PersistentHashMap.EMPTY;
while(rform != null && rform.first() instanceof Keyword)
{
    opts = opts.assoc(rform.first(), RT.second(rform));
    rform = rform.next().next();
}
ObjExpr ret =
build((IPersistentVector)RT.get(opts, implementsKey,
    PersistentVector.EMPTY), fields, null, tagname,
    classname, (Symbol) RT.get(opts, RT.TAG_KEY),
    rform, frm);
return ret;
}
}

static class ReifyParser implements IParser{
public Expr parse(C context, Object frm) throws Exception{
    //((reify this-name? [interfaces] (method-name [args] body)*)
    ISeq form = (ISeq) frm;
    ObjMethod enclosingMethod = (ObjMethod) METHOD.deref();
    String basename = enclosingMethod != null ?
        trimGenID(enclosingMethod.objx.name) + "$"
        : (munge(currentNS().name.name) + "$");
    String simpleName = "reify__" + RT.nextID();
    String classname = basename + simpleName;

    ISeq rform = RT.next(form);

    IPersistentVector interfaces =
        ((IPersistentVector) RT.first(rform))
        .cons(Symbol.intern("clojure.lang.IObj"));

    rform = RT.next(rform);

    ObjExpr ret =
        build(interfaces, null, null, classname,
            Symbol.intern(classname), null, rform, frm);
    if(frm instanceof IObj && ((IObj) frm).meta() != null)
        return new MetaExpr(ret, MapExpr
            .parse(context == C.EVAL
                ? context
                : C.EXPRESSION, ((IObj) frm).meta()));
    else
        return ret;
}
}

```

```

        }

    static ObjExpr build(IPersistentVector interfaceSyms,
                         IPersistentVector fieldSyms,
                         Symbol thisSym,
                         String tagName, Symbol className,
                         Symbol typeTag, ISeq methodForms, Object frm)
    throws Exception{
    NewInstanceExpr ret = new NewInstanceExpr(null);

    ret.src = frm;
    ret.name = className.toString();
    ret.classMeta = RT.meta(className);
    ret.internalName = ret.name.replace('.', '/');
    ret.objtype = Type.getObjectType(ret.internalName);

    if(thisSym != null)
        ret.thisName = thisSym.name;

    if(fieldSyms != null)
    {
        IPersistentMap fmap = PersistentHashMap.EMPTY;
        Object[] closesvec = new Object[2 * fieldSyms.count()];
        for(int i=0;i<fieldSyms.count();i++)
        {
            Symbol sym = (Symbol) fieldSyms.nth(i);
            LocalBinding lb = new LocalBinding(-1, sym, null,
                                              new MethodParamExpr(tagClass(tagOf(sym))),
                                              false,null);
            fmap = fmap.assoc(sym, lb);
            closesvec[i*2] = lb;
            closesvec[i*2 + 1] = lb;
        }

        //todo - inject __meta et al into closes - when?
        //use array map to preserve ctor order
        ret.closes = new PersistentArrayMap(closesvec);
        ret.fields = fmap;
        for(int i=fieldSyms.count()-1;i >= 0 &&
            ((Symbol)fieldSyms.nth(i)).name.startsWith("__");--i)
            ret.altCtorDrops++;
    }
    //todo - set up volatiles
    //
    ret.volatiles =
        PersistentHashSet.create(
            RT.seq(RT.get(ret.optionsMap, volatileKey)));
}

PersistentVector interfaces = PersistentVector.EMPTY;
for(ISeq s = RT.seq(interfaceSyms);s!=null;s = s.next())

```

```

{
  Class c = (Class) resolve((Symbol) s.first());
  if(!c.isInterface())
    throw new IllegalArgumentException(
      "only interfaces are supported, had: " + c.getName());
  interfaces = interfaces.cons(c);
}

Class superClass = Object.class;
Map[] mc = gatherMethods(superClass, RT.seq(interfaces));
Map overrideables = mc[0];
Map covariants = mc[1];
ret.mmap = overrideables;
ret.covariants = covariants;

String[] inames = interfaceNames(interfaces);

Class stub = compileStub(slashname(superClass), ret, inames, frm);
Symbol thistag = Symbol.intern(null, stub.getName());

try
{
  Var.pushThreadBindings(
    RT.map(CONSTANTS, PersistentVector.EMPTY,
           CONSTANT_IDS, new IdentityHashMap(),
           KEYWORDS, PersistentHashMap.EMPTY,
           VARS, PersistentHashMap.EMPTY,
           KEYWORD_CALLSITES, PersistentVector.EMPTY,
           PROTOCOL_CALLSITES, PersistentVector.EMPTY,
           VAR_CALLSITES, emptyVarCallSites(),
           NO_RECUR, null));

  if(ret.isDeftype())
  {
    Var.pushThreadBindings(RT.map(METHOD, null,
                                   LOCAL_ENV, ret.fields
                                   , COMPILE_STUB_SYM, Symbol.intern(null, tagName)
                                   , COMPILE_STUB_CLASS, stub));
  }

  //now (methodname [args] body)*
  ret.line = (Integer) LINE.deref();
  IPersistentCollection methods = null;
  for(ISeq s = methodForms; s != null; s = RT.next(s))
  {
    NewInstanceMethod m =
      NewInstanceMethod.parse(ret, (ISeq) RT.first(s),
                             thistag, overrideables);
    methods = RT.conj(methods, m);
  }
}

```

```

        ret.methods = methods;
        ret.keywords = (IPersistentMap) KEYWORDS.deref();
        ret.vars = (IPersistentMap) VARS.deref();
        ret.constants = (PersistentVector) CONSTANTS.deref();
        ret.constantsID = RT.nextID();
        ret.keywordCallsites =
            (IPersistentVector) KEYWORD_CALLSITES.deref();
        ret.protocolCallsites =
            (IPersistentVector) PROTOCOL_CALLSITES.deref();
        ret.varCallsites = (IPersistentSet) VAR_CALLSITES.deref();
    }
}

finally
{
    if(ret.isDeftype())
        Var.popThreadBindings();
    Var.popThreadBindings();
}

ret.compile(slashname(superClass), inames, false);
ret.getCompiledClass();
return ret;
}

/**
 * Current host interop uses reflection, which requires
 * pre-existing classes. Work around this by:
 * Generate a stub class that has the same interfaces and fields
 * as the class we are generating.
 * Use it as a type hint for this, and bind the simple name of
 * the class to this stub (in resolve etc)
 * Unmunge the name (using a magic prefix) on any code gen
 * for classes
 */
static Class compileStub(String superName,
                        NewInstanceExpr ret,
                        String[] interfaceNames,
                        Object frm){
    ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAXS);
    ClassVisitor cv = cw;
    cv.visit(V1_5, ACC_PUBLIC + ACC_SUPER,
             COMPILE_STUB_PREFIX + "/" + ret.internalName,
             null, superName, interfaceNames);

    //instance fields for closed-overs
    for(ISeq s = RT.keys(ret.closes); s != null; s = s.next())
    {
        LocalBinding lb = (LocalBinding) s.first();
        int access =
            ACC_PUBLIC + (ret.isVolatile(lb) ? ACC_VOLATILE :
                          ret.isMutable(lb) ? 0 :

```

```

        ACC_FINAL);
if(lb.getPrimitiveType() != null)
    cv.visitField(access,
        lb.name,
        Type.getType(lb.getPrimitiveType())
            .getDescriptor(),
        null, null);
else
//todo - when closed-overs are fields, use more specific
// types here and in ctor and emitLocal?
    cv.visitField(access,
        lb.name, OBJECT_TYPE.getDescriptor(), null, null);
}

//ctor that takes closed-overs and does nothing
Method m = new Method("<init>",
                    Type.VOID_TYPE, ret.ctorTypes());
GeneratorAdapter ctorgen =
new GeneratorAdapter(ACC_PUBLIC,
                     m,
                     null,
                     null,
                     cv);
ctorgen.visitCode();
ctorgen.loadThis();
ctorgen.invokeConstructor(Type.getObjectType(superName),
                         voidctor);
ctorgen.returnValue();
ctorgen.endMethod();

if(ret.altCtorDrops > 0)
{
    Type[] ctorTypes = ret.ctorTypes();
    Type[] altCtorTypes =
        new Type[ctorTypes.length-ret.altCtorDrops];
    for(int i=0;i<altCtorTypes.length;i++)
        altCtorTypes[i] = ctorTypes[i];
    Method alt = new Method("<init>",
                           Type.VOID_TYPE, altCtorTypes);
    ctorgen = new GeneratorAdapter(ACC_PUBLIC,
                                   alt,
                                   null,
                                   null,
                                   cv);
    ctorgen.visitCode();
    ctorgen.loadThis();
    ctorgen.loadArgs();
    for(int i=0;i<ret.altCtorDrops;i++)
        ctorgen.visitInsn(Opcodes.ACONST_NULL);
}

```

```

        ctorgen.invokeConstructor(
            Type.getObjectType(
                COMPILE_STUB_PREFIX + "/" + ret.internalName),
            new Method("<init>", Type.VOID_TYPE, ctorTypes));

        ctorgen.returnValue();
        ctorgen.endMethod();
    }
}

//end of class
cv.visitEnd();

byte[] bytecode = cw.toByteArray();
DynamicClassLoader loader = (DynamicClassLoader) LOADER.deref();
return loader.defineClass(
    COMPILE_STUB_PREFIX + "." + ret.name,
    bytecode, frm);
}

static String[] interfaceNames(IPersistentVector interfaces){
    int icnt = interfaces.count();
    String[] inames = icnt > 0 ? new String[icnt] : null;
    for(int i=0;i<icnt;i++)
        inames[i] = slashname((Class) interfaces.nth(i));
    return inames;
}

static String slashname(Class c){
    return c.getName().replace('.', '/');
}

protected void emitMethods(ClassVisitor cv){
    for(ISeq s = RT.seq(methods); s != null; s = s.next())
    {
        ObjMethod method = (ObjMethod) s.first();
        method.emit(this, cv);
    }
}

//emit bridge methods
for(Map.Entry<IPersistentVector,Set<Class>> e
    : covariants.entrySet())
{
    java.lang.reflect.Method m = mmap.get(e.getKey());
    Class[] params = m.getParameterTypes();
    Type[] argTypes = new Type[params.length];

    for(int i = 0; i < params.length; i++)
    {
        argTypes[i] = Type.getType(params[i]);
    }
}

```

```

Method target =
    new Method(m.getName(),
                Type.getType(m.getReturnType()),
                argTypes);

for(Class retType : e.getValue())
{
    Method meth =
        new Method(m.getName(),
                    Type.getType(retType),
                    argTypes);

    GeneratorAdapter gen =
        new GeneratorAdapter(ACC_PUBLIC + ACC_BRIDGE,
                             meth,
                             null,
                             //todo don't hardwire this
                             EXCEPTION_TYPES,
                             cv);
    gen.visitCode();
    gen.loadThis();
    gen.loadArgs();
    gen.invokeInterface(
        Type.getType(m.getDeclaringClass()),target);
    gen.returnValue();
    gen.endMethod();
}
}

static public IPersistentVector msig(java.lang.reflect.Method m){
    return
        RT.vector(m.getName(),
                  RT.seq(m.getParameterTypes()),
                  m.getReturnType());
}

static void considerMethod(java.lang.reflect.Method m, Map mm){
    IPersistentVector mk = msig(m);
    int mods = m.getModifiers();

    if(!(mm.containsKey(mk)
        || !(Modifier.isPublic(mods) || Modifier.isProtected(mods))
        || Modifier.isStatic(mods)
        || Modifier.isFinal(mods)))
    {
        mm.put(mk, m);
    }
}

```

```

static void gatherMethods(Class c, Map mm){
    for(; c != null; c = c.getSuperclass())
    {
        for(java.lang.reflect.Method m : c.getDeclaredMethods())
            considerMethod(m, mm);
        for(java.lang.reflect.Method m : c.getMethods())
            considerMethod(m, mm);
    }
}

static public Map[] gatherMethods(Class sc, ISeq interfaces){
    Map allm = new HashMap();
    gatherMethods(sc, allm);
    for(; interfaces != null; interfaces = interfaces.next())
        gatherMethods((Class) interfaces.first(), allm);

    Map<IPersistentVector,java.lang.reflect.Method> mm =
        new HashMap<IPersistentVector,java.lang.reflect.Method>();
    Map<IPersistentVector,Set<Class>> covariants =
        new HashMap<IPersistentVector,Set<Class>>();
    for(Object o : allm.entrySet())
    {
        Map.Entry e = (Map.Entry) o;
        IPersistentVector mk = (IPersistentVector) e.getKey();
        mk = (IPersistentVector) mk.pop();
        java.lang.reflect.Method m =
            (java.lang.reflect.Method) e.getValue();
        if(mm.containsKey(mk)) //covariant return
        {
            Set<Class> cvs = covariants.get(mk);
            if(cvs == null)
            {
                cvs = new HashSet<Class>();
                covariants.put(mk, cvs);
            }
            java.lang.reflect.Method om = mm.get(mk);
            if(om.getReturnType()
                .isAssignableFrom(m.getReturnType()))
            {
                cvs.add(om.getReturnType());
                mm.put(mk, m);
            }
            else
                cvs.add(m.getReturnType());
        }
        else
            mm.put(mk, m);
    }
    return new Map[]{mm,covariants};
}

```

```

        }
    }

public static class NewInstanceMethod extends ObjMethod{
    String name;
    Type[] argTypes;
    Type retType;
    Class retClass;
    Class[] excludes;

    static Symbol dummyThis =
        Symbol.intern(null,"dummy_this_dlskjsdfower");
    private IPersistentVector parms;

    public NewInstanceMethod(ObjExpr objx, ObjMethod parent){
        super(objx, parent);
    }

    int numParams(){
        return argLocals.count();
    }

    String getMethodName(){
        return name;
    }

    Type getReturnType(){
        return retType;
    }

    Type[] getArgTypes(){
        return argTypes;
    }

    static public IPersistentVector msig(String name,
                                         Class[] paramTypes){
        return RT.vector(name,RT.seq(paramTypes));
    }

    static NewInstanceMethod parse(ObjExpr objx, ISeq form,
                                  Symbol thistag,
                                  Map overrideables)
        throws Exception{
        //((methodname [this-name args*] body...)
        //this-name might be nil
        NewInstanceMethod method =
            new NewInstanceMethod(objx, (ObjMethod) METHOD.deref());
        Symbol dotname = (Symbol)RT.first(form);
    }
}

```

```

Symbol name =
    (Symbol) Symbol.intern(null,
        munge(dotname.name))
        .withMeta(RT.meta(dotname));
IPersistentVector parms = (IPersistentVector) RT.second(form);
if(parms.count() == 0)
{
    throw new IllegalArgumentException(
"Must supply at least one argument for 'this' in: " + dotname);
}
Symbol thisName = (Symbol) parms.nth(0);
parms = RT.subvec(parms,1,parms.count());
ISeq body = RT.next(RT.next(form));
try
{
    method.line = (Integer) LINE.deref();
    //register as the current method and set up a new env frame
    PathNode pnode =
        new PathNode(PATHTYPE.PATH, (PathNode) CLEAR_PATH.get());
    Var.pushThreadBindings(
        RT.map(
            METHOD, method,
            LOCAL_ENV, LOCAL_ENV.deref(),
            LOOP_LOCALS, null,
            NEXT_LOCAL_NUM, 0
            ,CLEAR_PATH, pnode
            ,CLEAR_ROOT, pnode
            ,CLEAR_SITES, PersistentHashMap.EMPTY
        ));
    //register 'this' as local 0
    if(thisName != null)
        registerLocal((thisName == null)
        ? dummyThis
        : thisName, thistag, null, false);
    else
        getAndIncLocalNum();
    PersistentVector argLocals = PersistentVector.EMPTY;
    method.retClass = tagClass(tagOf(name));
    method.argTypes = new Type[parms.count()];
    boolean hinted = tagOf(name) != null;
    Class[] pclasses = new Class[parms.count()];
    Symbol[] psyms = new Symbol[parms.count()];
    for(int i = 0; i < parms.count(); i++)
    {
        if(!(parms.nth(i) instanceof Symbol))
            throw new IllegalArgumentException(
                "parms must be Symbols");
    }
}

```

```

Symbol p = (Symbol) parms.nth(i);
Object tag = tagOf(p);
if(tag != null)
    hinted = true;
if(p.getNamespace() != null)
    p = Symbol.intern(p.name);
Class pclass = tagClass(tag);
pclasses[i] = pclass;
psyms[i] = p;
}
Map matches =
    findMethodsWithNameAndArity(name.name,
                                   parms.count(),
                                   overrideables);
Object mk = msig(name.name, pclasses);
java.lang.reflect.Method m = null;
if(matches.size() > 0)
{
    //multiple methods
    if(matches.size() > 1)
    {
        //must be hinted and match one method
        if(!hinted)
            throw new IllegalArgumentException(
                "Must hint overloaded method: " + name.name);
        m = (java.lang.reflect.Method) matches.get(mk);
        if(m == null)
            throw new IllegalArgumentException(
                "Can't find matching overloaded method: " + name.name);
        if(m.getReturnType() != method.retClass)
            throw new IllegalArgumentException(
                "Mismatched return type: " + name.name +
                ", expected: " + m.getReturnType().getName() +
                ", had: " + method.retClass.getName());
    }
    else //one match
    {
        //if hinted, validate match,
        if(hinted)
        {
            m = (java.lang.reflect.Method) matches.get(mk);
            if(m == null)
                throw new IllegalArgumentException(
                    "Can't find matching method: " + name.name +
                    ", leave off hints for auto match.");
            if(m.getReturnType() != method.retClass)
                throw new IllegalArgumentException(
                    "Mismatched return type: " + name.name +
                    ", expected: " +
                    m.getReturnType().getName() +

```

```

        ", had: " + method.retClass.getName());
    }
    else //adopt found method sig
    {
        m = (java.lang.reflect.Method)
            matches.values().iterator().next();
        method.retClass = m.getReturnType();
        pclasses = m.getParameterTypes();
    }
}
}

// else if(findMethodsWithName(name.name,allmethods).size()>0)
//     throw new IllegalArgumentException(
//         "Can't override/overload method: " + name.name);
// else
//     throw new IllegalArgumentException(
//         "Can't define method not in interfaces: " + name.name);

//else
//validate unique name+arity among additional methods

method.retType = Type.getType(method.retClass);
method.exclasses = m.getExceptionTypes();

for(int i = 0; i < parms.count(); i++)
{
    LocalBinding lb =
        registerLocal(psyms[i],
                      null,
                      new MethodParamExpr(pclasses[i]),true);
    argLocals = argLocals.assocN(i,lb);
    method.argTypes[i] = Type.getType(pclasses[i]);
}
for(int i = 0; i < parms.count(); i++)
{
    if(pclasses[i] == long.class ||
       pclasses[i] == double.class)
        getAndIncLocalNum();
}
LOOP_LOCALS.set(argLocals);
method.name = name.name;
method.methodMeta = RT.meta(name);
method.parms = parms;
method.argLocals = argLocals;
method.body = (new BodyExpr.Parser()).parse(C.RETURN, body);
return method;
}
finally
{
    Var.popThreadBindings();
}

```

```

        }
    }

private static Map findMethodsWithNameAndArity(String name,
                                              int arity,
                                              Map mm){
    Map ret = new HashMap();
    for(Object o : mm.entrySet())
    {
        Map.Entry e = (Map.Entry) o;
        java.lang.reflect.Method m =
            (java.lang.reflect.Method) e.getValue();
        if(name.equals(m.getName()) &&
           m.getParameterTypes().length == arity)
            ret.put(e.getKey(), e.getValue());
    }
    return ret;
}

private static Map findMethodsWithName(String name, Map mm){
    Map ret = new HashMap();
    for(Object o : mm.entrySet())
    {
        Map.Entry e = (Map.Entry) o;
        java.lang.reflect.Method m =
            (java.lang.reflect.Method) e.getValue();
        if(name.equals(m.getName()))
            ret.put(e.getKey(), e.getValue());
    }
    return ret;
}

public void emit(ObjExpr obj, ClassVisitor cv){
    Method m =
        new Method(getMethodName(), getReturnType(), getArgTypes());

    Type[] extypes = null;
    if(exclasses.length > 0)
    {
        extypes = new Type[exclasses.length];
        for(int i=0;i<exclasses.length;i++)
            extypes[i] = Type.getType(exclasses[i]);
    }
    GeneratorAdapter gen = new GeneratorAdapter(ACC_PUBLIC,
                                                m,
                                                null,
                                                extypes,
                                                cv);
    addAnnotation(gen,methodMeta);
    for(int i = 0; i < parms.count(); i++)
}

```

```

{
  IPersistentMap meta = RT.meta(parms.nth(i));
  addParameterAnnotation(gen, meta, i);
}
gen.visitCode();

Label loopLabel = gen.mark();

gen.visitLineNumber(line, loopLabel);
try
{
  Var.pushThreadBindings(
    RT.map(LOOP_LABEL, loopLabel, METHOD, this));

  emitBody(objx, gen, retClass, body);
  Label end = gen.mark();
  gen.visitLocalVariable("this",
    obj=objtype.getDescriptor(),
    null, loopLabel, end, 0);
  for(ISeq lbs = argLocals.seq();
      lbs != null;
      lbs = lbs.next())
  {
    LocalBinding lb = (LocalBinding) lbs.first();
    gen.visitLocalVariable(lb.name,
      argTypes[lb.idx-1]
      .getDescriptor(),
      null,
      loopLabel, end, lb.idx);
  }
}
catch(Exception e)
{
  throw new RuntimeException(e);
}
finally
{
  Var.popThreadBindings();
}

gen.returnValue();
//gen.visitMaxs(1, 1);
gen.endMethod();
}
}

static Class primClass(Symbol sym){
  if(sym == null)
    return null;
  Class c = null;
}

```

```

        if(sym.name.equals("int"))
            c = int.class;
        else if(sym.name.equals("long"))
            c = long.class;
        else if(sym.name.equals("float"))
            c = float.class;
        else if(sym.name.equals("double"))
            c = double.class;
        else if(sym.name.equals("char"))
            c = char.class;
        else if(sym.name.equals("short"))
            c = short.class;
        else if(sym.name.equals("byte"))
            c = byte.class;
        else if(sym.name.equals("boolean"))
            c = boolean.class;
        else if(sym.name.equals("void"))
            c = void.class;
        return c;
    }

    static Class tagClass(Object tag) throws Exception{
        if(tag == null)
            return Object.class;
        Class c = null;
        if(tag instanceof Symbol)
            c = primClass((Symbol) tag);
        if(c == null)
            c = HostExpr.tagToClass(tag);
        return c;
    }

    static Class primClass(Class c){
        return c.isPrimitive()?c:Object.class;
    }

    static public class MethodParamExpr
        implements Expr, MaybePrimitiveExpr{
        final Class c;

        public MethodParamExpr(Class c){
            this.c = c;
        }

        public Object eval() throws Exception{
            throw new Exception("Can't eval");
        }

        public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
            throw new RuntimeException("Can't emit");
        }
    }
}

```

```
}

public boolean hasJavaClass() throws Exception{
    return c != null;
}

public Class getJavaClass() throws Exception{
    return c;
}

public boolean canEmitPrimitive(){
    return Util.isPrimitive(c);
}

public void emitUnboxed(C context,
                        ObjExpr objx,
                        GeneratorAdapter gen){
    throw new RuntimeException("Can't emit");
}
}

public static class CaseExpr extends UntypedExpr{
    public final LocalBindingExpr expr;
    public final int shift, mask, low, high;
    public final Expr defaultExpr;
    public final HashMap<Integer,Expr> tests;
    public final HashMap<Integer,Expr> thens;
    public final boolean allKeywords;

    public final int line;

    final static Method hashMethod =
        Method.getMethod("int hash(Object)");
    final static Method hashCodeMethod =
        Method.getMethod("int hashCode()");
    final static Method equalsMethod =
        Method.getMethod("boolean equals(Object, Object)");

    public CaseExpr(int line, LocalBindingExpr expr, int shift,
                   int mask, int low, int high, Expr defaultExpr,
                   HashMap<Integer,Expr> tests,
                   HashMap<Integer,Expr> thens,
                   boolean allKeywords){
        this.expr = expr;
        this.shift = shift;
        this.mask = mask;
        this.low = low;
        this.high = high;
        this.defaultExpr = defaultExpr;
```

```

        this.tests = tests;
        this.thens = thens;
        this.line = line;
        this.allKeywords = allKeywords;
    }

    public Object eval() throws Exception{
        throw new UnsupportedOperationException("Can't eval case");
    }

    public void emit(C context, ObjExpr objx, GeneratorAdapter gen){
        Label defaultLabel = gen.newLabel();
        Label endLabel = gen.newLabel();
        HashMap<Integer,Label> labels = new HashMap();

        for(Integer i : tests.keySet())
        {
            labels.put(i, gen.newLabel());
        }

        Label[] la = new Label[(high-low)+1];

        for(int i=low;i<=high;i++)
        {
            la[i-low] = labels.containsKey(i)
                ? labels.get(i)
                : defaultLabel;
        }

        gen.visitLineNumber(line, gen.mark());
        expr.emit(C.EXPRESSION, objx, gen);
        gen.invokeStatic(UTIL_TYPE,hashMethod);
        gen.push(shift);
        gen.visitInsn(ISHR);
        gen.push(mask);
        gen.visitInsn(IAND);
        gen.visitTableSwitchInsn(low, high, defaultLabel, la);

        for(Integer i : labels.keySet())
        {
            gen.mark(labels.get(i));
            expr.emit(C.EXPRESSION, objx, gen);
            tests.get(i).emit(C.EXPRESSION, objx, gen);
            if(allKeywords)
            {
                gen.visitJumpInsn(IF_ACMPNE, defaultLabel);
            }
            else
            {
                gen.invokeStatic(UTIL_TYPE, equalsMethod);
            }
        }
    }
}

```

```

        gen.ifZCmp(GeneratorAdapter.EQ, defaultLabel);
    }
    thenS.get(i).emit(C.EXPRESSION,objx,gen);
    gen.goTo(endLabel);
}

gen.mark(defaultLabel);
defaultExpr.emit(C.EXPRESSION, objx, gen);
gen.mark(endLabel);
if(context == C.STATEMENT)
    gen.pop();
}

static class Parser implements IParser{
    //(...expr shift mask low high default
    // map<minhash, [test then]> identity?)
    //prepared by case macro and presumed correct
    //case macro binds actual expr in let so expr is always a local,
    //no need to worry about multiple evaluation
    public Expr parse(C context, Object frm) throws Exception{
        ISeq form = (ISeq) frm;
        if(context == C.EVAL)
            return
                analyze(context,
                    RT.list(
                        RT.list(FN, PersistentVector.EMPTY, form)));
        PersistentVector args = PersistentVector.create(form.next());
        HashMap<Integer,Expr> tests = new HashMap();
        HashMap<Integer,Expr> thenS = new HashMap();

        LocalBindingExpr testexpr =
            (LocalBindingExpr) analyze(C.EXPRESSION, args.nth(0));
        testexpr.shouldClear = false;

        PathNode branch =
            new PathNode(PATHTYPE.BRANCH, (PathNode) CLEAR_PATH.get());
        for(Object o : ((Map)args.nth(6)).entrySet())
        {
            Map.Entry e = (Map.Entry) o;
            Integer minhash = ((Number)e.getKey()).intValue();
            MapEntry me = (MapEntry) e.getValue();
            Expr testExpr = new ConstantExpr(me.getKey());
            tests.put(minhash, testExpr);
            Expr thenExpr;
            try {
                Var.pushThreadBindings(
                    RT.map(CLEAR_PATH,
                        new PathNode(PATHTYPE.PATH,branch)));
                thenExpr = analyze(context, me.getValue());
            }
        }
    }
}

```

```

        finally{
            Var.popThreadBindings();
        }
        thens.put(minhash, thenExpr);
    }

    Expr defaultExpr;
    try {
        Var.pushThreadBindings(
            RT.map(CLEAR_PATH,
                new PathNode(PATHTYPE.PATH,branch)));
        defaultExpr = analyze(context, args.nth(5));
    }
    finally{
        Var.popThreadBindings();
    }

    return new CaseExpr(((Number)LINE.deref()).intValue(),
        testexpr,
        ((Number)args.nth(1)).intValue(),
        ((Number)args.nth(2)).intValue(),
        ((Number)args.nth(3)).intValue(),
        ((Number)args.nth(4)).intValue(),
        defaultExpr,
        tests,thens,args.nth(7) != RT.F);

    }
}
}

static IPersistentCollection emptyVarCallSites(){
    return PersistentHashSet.EMPTY;
}

```

9.25 Cons.java

(ASeq [571]) (Serializable [1723])
— Cons.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 25, 2006 11:01:29 AM */

package clojure.lang;

```

```
import java.io.Serializable;

final public class Cons extends ASeq implements Serializable {

    private final Object _first;
    private final ISeq _more;

    public Cons(Object first, ISeq _more){
        this._first = first;
        this._more = _more;
    }

    public Cons(IPersistentMap meta, Object _first, ISeq _more){
        super(meta);
        this._first = _first;
        this._more = _more;
    }

    public Object first(){
        return _first;
    }

    public ISeq next(){
        return more().seq();
    }

    public ISeq more(){
        if(_more == null)
            return PersistentList.EMPTY;
        return _more;
    }

    public int count(){
        return 1 + RT.count(_more);
    }

    public Cons withMeta(IPersistentMap meta){
        return new Cons(meta, _first, _more);
    }
}
```

9.26 Counted.java

— Counted.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

/* A class that implements Counted promises that it is a collection
 * that implement a constant-time count() */

public interface Counted {
    int count();
}
```

9.27 Delay.java

(IDeref [773])

— Delay.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Jun 28, 2007 */

package clojure.lang;

public class Delay implements IDeref{
Object val;
IFn fn;

public Delay(IFn fn){
    this.fn = fn;
    this.val = null;
}

static public Object force(Object x) throws Exception{
    return (x instanceof Delay) ?
        ((Delay) x).deref()
        : x;
}

synchronized public Object deref() throws Exception{
```

```

    if(fn != null)
    {
        val = fn.invoke();
        fn = null;
    }
    return val;
}
}

```

—————

9.28 DynamicClassLoader.java

(URLClassLoader [1723])
 — DynamicClassLoader.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Aug 21, 2007 */

package clojure.lang;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.net.URLClassLoader;
import java.net.URL;
import java.lang.ref.ReferenceQueue;
import java.lang.ref.SoftReference;

public class DynamicClassLoader extends URLClassLoader{
HashMap<Integer, Object[]> constantVals =
    new HashMap<Integer, Object[]>();
static ConcurrentHashMap<String, SoftReference<Class>>classCache =
    new ConcurrentHashMap<String, SoftReference<Class> >();

static final URL[] EMPTY_URLS = new URL[] {};

static final ReferenceQueue rq = new ReferenceQueue();

public DynamicClassLoader(){
    //pseudo test in lieu of hasContextClassLoader()
    super(EMPTY_URLS,
        (Thread.currentThread().getContextClassLoader() == null ||
        Thread.currentThread().getContextClassLoader() ==
            ClassLoader.getSystemClassLoader())
    ? Compiler.class.getClassLoader()

```

```

        : Thread.currentThread().getContextClassLoader());
    }

public DynamicClassLoader(ClassLoader parent){
    super(EMPTY_URLS,parent);
}

public Class defineClass(String name, byte[] bytes, Object srcForm){
    Util.clearCache(rq, classCache);
    Class c = defineClass(name, bytes, 0, bytes.length);
    classCache.put(name, new SoftReference(c,rq));
    return c;
}

protected Class<?> findClass(String name)
throws ClassNotFoundException{
    SoftReference<Class> cr = classCache.get(name);
    if(cr != null)
    {
        Class c = cr.get();
        if(c != null)
            return c;
        else
            classCache.remove(name, cr);
    }
    return super.findClass(name);
}

public void registerConstants(int id, Object[] val){
    constantVals.put(id, val);
}

public Object[] getConstants(int id){
    return constantVals.get(id);
}

public void addURL(URL url){
    super.addURL(url);
}
}

```

9.29 EnumerationSeq.java

(ASeq [571])
— EnumerationSeq.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 3, 2008 */

package clojure.lang;

import java.io.IOException;
import java.io.NotSerializableException;
import java.util.Enumeration;

public class EnumerationSeq extends ASeq{
    final Enumeration iter;
    final State state;

    static class State{
        volatile Object val;
        volatile Object _rest;
    }

    public static EnumerationSeq create(Enumeration iter){
        if(iter.hasMoreElements())
            return new EnumerationSeq(iter);
        return null;
    }

    EnumerationSeq(Enumeration iter){
        this.iter = iter;
        state = new State();
        this.state.val = state;
        this.state._rest = state;
    }

    EnumerationSeq(IPersistentMap meta, Enumeration iter, State state){
        super(meta);
        this.iter = iter;
        this.state = state;
    }

    public Object first(){
        if(state.val == state)
            synchronized(state)
            {
                if(state.val == state)
                    state.val = iter.nextElement();
            }
        return state.val;
    }

    public ISeq next(){
```

```

        if(state._rest == state)
            synchronized(state)
            {
                if(state._rest == state)
                {
                    first();
                    state._rest = create(iter);
                }
            }
        return (ISeq) state._rest;
    }

    public EnumerationSeq withMeta(IPersistentMap meta){
        return new EnumerationSeq(meta, iter, state);
    }

    private void writeObject (java.io.ObjectOutputStream out)
        throws IOException {
        throw new NotSerializableException(getClass().getName());
    }
}

```

9.30 Fn.java

— Fn.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Nov 25, 2008 */

package clojure.lang;

public interface Fn{
}

```

9.31 IChunk.java

(Indexed [799])

— IChunk.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Jun 18, 2009 */

package clojure.lang;

public interface IChunk extends Indexed{

IChunk dropFirst();

Object reduce(IFn f, Object start) throws Exception;
}
```

—————

9.32 IChunkedSeq.java

(ISeq [805])

— IChunkedSeq.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich May 24, 2009 */

package clojure.lang;

public interface IChunkedSeq extends ISeq{

IChunk chunkedFirst() throws Exception;

ISeq chunkedNext() throws Exception;

ISeq chunkedMore() throws Exception;
}
```

—————

9.33 IDeref.java

— IDeref.java —

/*

```

\getchunk{Clojure Copyright}
*/
/* rich Feb 9, 2009 */

package clojure.lang;

public interface IDeref{
Object deref() throws Exception;
}

```

9.34 IEditableCollection.java

— IEditableCollection.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 17, 2009 */

package clojure.lang;

public interface IEditableCollection{
ITransientCollection asTransient();
}

```

9.35 IFn.java

(Callable [1723]) (Runnable [1723])
— IFn.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 25, 2006 3:54:03 PM */

package clojure.lang;

import java.util.concurrent.Callable;

public interface IFn extends Callable, Runnable{

```

```
public Object invoke()
throws Exception;

public Object invoke(Object arg1)
throws Exception;

public Object invoke(Object arg1, Object arg2)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4, Object arg5)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4, Object arg5, Object arg6)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4, Object arg5, Object arg6,
Object arg7)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4, Object arg5, Object arg6,
Object arg7, Object arg8)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4, Object arg5, Object arg6,
Object arg7, Object arg8, Object arg9)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4, Object arg5, Object arg6,
Object arg7, Object arg8, Object arg9,
Object arg10)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4, Object arg5, Object arg6,
Object arg7, Object arg8, Object arg9,
Object arg10, Object arg11)
throws Exception;
```

```
public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18)
```

```
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20)
throws Exception;

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20, Object... args)
throws Exception;

public Object applyTo(ISeq arglist) throws Exception;

static public interface
    L{long invokePrim();}
static public interface
    D{double invokePrim();}
static public interface
    OL{long invokePrim(Object arg0);}
static public interface
    OD{double invokePrim(Object arg0);}
static public interface
    LO{Object invokePrim(long arg0);}
static public interface
    LL{long invokePrim(long arg0);}
static public interface
    LD{double invokePrim(long arg0);}
static public interface
    DO{Object invokePrim(double arg0);}
static public interface
    DL{long invokePrim(double arg0);}
static public interface
```

```
        DD{double invokePrim(double arg0);}
static public interface
    OOL{long invokePrim(Object arg0, Object arg1);}
static public interface
    OOD{double invokePrim(Object arg0, Object arg1);}
static public interface
    OLO{Object invokePrim(Object arg0, long arg1);}
static public interface
    OLL{long invokePrim(Object arg0, long arg1);}
static public interface
    OLD{double invokePrim(Object arg0, long arg1);}
static public interface
    ODO{Object invokePrim(Object arg0, double arg1);}
static public interface
    ODL{long invokePrim(Object arg0, double arg1);}
static public interface
    ODD{double invokePrim(Object arg0, double arg1);}
static public interface
    LOO{Object invokePrim(long arg0, Object arg1);}
static public interface
    LOL{long invokePrim(long arg0, Object arg1);}
static public interface
    LOD{double invokePrim(long arg0, Object arg1);}
static public interface
    LLO{Object invokePrim(long arg0, long arg1);}
static public interface
    LLL{long invokePrim(long arg0, long arg1);}
static public interface
    LLD{double invokePrim(long arg0, long arg1);}
static public interface
    LDO{Object invokePrim(long arg0, double arg1);}
static public interface
    LDL{long invokePrim(long arg0, double arg1);}
static public interface
    LDD{double invokePrim(long arg0, double arg1);}
static public interface
    DOO{Object invokePrim(double arg0, Object arg1);}
static public interface
    DOL{long invokePrim(double arg0, Object arg1);}
static public interface
    DOD{double invokePrim(double arg0, Object arg1);}
static public interface
    DLO{Object invokePrim(double arg0, long arg1);}
static public interface
    DLL{long invokePrim(double arg0, long arg1);}
static public interface
    DLD{double invokePrim(double arg0, long arg1);}
static public interface
    DDO{Object invokePrim(double arg0, double arg1);}
static public interface
```

```
    DDL{long invokePrim(double arg0, double arg1);}
static public interface
    DDD{double invokePrim(double arg0, double arg1);}
static public interface
    O0OL{long invokePrim(Object arg0, Object arg1, Object arg2);}
static public interface
    O0OD{double invokePrim(Object arg0, Object arg1, Object arg2);}
static public interface
    O0LO{Object invokePrim(Object arg0, Object arg1, long arg2);}
static public interface
    O0LL{long invokePrim(Object arg0, Object arg1, long arg2);}
static public interface
    O0LD{double invokePrim(Object arg0, Object arg1, long arg2);}
static public interface
    O0DO{Object invokePrim(Object arg0, Object arg1, double arg2);}
static public interface
    O0DL{long invokePrim(Object arg0, Object arg1, double arg2);}
static public interface
    O0DD{double invokePrim(Object arg0, Object arg1, double arg2);}
static public interface
    OLOO{Object invokePrim(Object arg0, long arg1, Object arg2);}
static public interface
    OLOL{long invokePrim(Object arg0, long arg1, Object arg2);}
static public interface
    OLOD{double invokePrim(Object arg0, long arg1, Object arg2);}
static public interface
    OLLO{Object invokePrim(Object arg0, long arg1, long arg2);}
static public interface
    OLLL{long invokePrim(Object arg0, long arg1, long arg2);}
static public interface
    OLLD{double invokePrim(Object arg0, long arg1, long arg2);}
static public interface
    OLDO{Object invokePrim(Object arg0, long arg1, double arg2);}
static public interface
    OLDL{long invokePrim(Object arg0, long arg1, double arg2);}
static public interface
    OLDD{double invokePrim(Object arg0, long arg1, double arg2);}
static public interface
    ODOO{Object invokePrim(Object arg0, double arg1, Object arg2);}
static public interface
    ODOL{long invokePrim(Object arg0, double arg1, Object arg2);}
static public interface
    ODOD{double invokePrim(Object arg0, double arg1, Object arg2);}
static public interface
    ODLO{Object invokePrim(Object arg0, double arg1, long arg2);}
static public interface
    ODLL{long invokePrim(Object arg0, double arg1, long arg2);}
static public interface
    ODLD{double invokePrim(Object arg0, double arg1, long arg2);}
static public interface
```

```

    ODDO{Object invokePrim(Object arg0, double arg1, double arg2);}
static public interface
    ODDL{long invokePrim(Object arg0, double arg1, double arg2);}
static public interface
    ODDD{double invokePrim(Object arg0, double arg1, double arg2);}
static public interface
    L000{Object invokePrim(long arg0, Object arg1, Object arg2);}
static public interface
    L00L{long invokePrim(long arg0, Object arg1, Object arg2);}
static public interface
    L00D{double invokePrim(long arg0, Object arg1, Object arg2);}
static public interface
    LOLO{Object invokePrim(long arg0, Object arg1, long arg2);}
static public interface
    LOLL{long invokePrim(long arg0, Object arg1, long arg2);}
static public interface
    LOLD{double invokePrim(long arg0, Object arg1, long arg2);}
static public interface
    LOD0{Object invokePrim(long arg0, Object arg1, double arg2);}
static public interface
    LODL{long invokePrim(long arg0, Object arg1, double arg2);}
static public interface
    LODD{double invokePrim(long arg0, Object arg1, double arg2);}
static public interface
    LL00{Object invokePrim(long arg0, long arg1, Object arg2);}
static public interface
    LL0L{long invokePrim(long arg0, long arg1, Object arg2);}
static public interface
    LL0D{double invokePrim(long arg0, long arg1, Object arg2);}
static public interface
    LLL0{Object invokePrim(long arg0, long arg1, long arg2);}
static public interface
    LLLL{long invokePrim(long arg0, long arg1, long arg2);}
static public interface
    LLLD{double invokePrim(long arg0, long arg1, long arg2);}
static public interface
    LLDO{Object invokePrim(long arg0, long arg1, double arg2);}
static public interface
    LLDL{long invokePrim(long arg0, long arg1, double arg2);}
static public interface
    LLDD{double invokePrim(long arg0, long arg1, double arg2);}
static public interface
    LD00{Object invokePrim(long arg0, double arg1, Object arg2);}
static public interface
    LDOL{long invokePrim(long arg0, double arg1, Object arg2);}
static public interface
    LDOD{double invokePrim(long arg0, double arg1, Object arg2);}
static public interface
    LDLO{Object invokePrim(long arg0, double arg1, long arg2);}
static public interface

```

```
LDLL{long invokePrim(long arg0, double arg1, long arg2);}
static public interface
    LDLD{double invokePrim(long arg0, double arg1, long arg2);}
static public interface
    LDDO{Object invokePrim(long arg0, double arg1, double arg2);}
static public interface
    LDDL{long invokePrim(long arg0, double arg1, double arg2);}
static public interface
    LDDD{double invokePrim(long arg0, double arg1, double arg2);}
static public interface
    D000{Object invokePrim(double arg0, Object arg1, Object arg2);}
static public interface
    DOOL{long invokePrim(double arg0, Object arg1, Object arg2);}
static public interface
    DOOD{double invokePrim(double arg0, Object arg1, Object arg2);}
static public interface
    DOLO{Object invokePrim(double arg0, Object arg1, long arg2);}
static public interface
    DOLL{long invokePrim(double arg0, Object arg1, long arg2);}
static public interface
    DOLD{double invokePrim(double arg0, Object arg1, long arg2);}
static public interface
    DODO{Object invokePrim(double arg0, Object arg1, double arg2);}
static public interface
    DODL{long invokePrim(double arg0, Object arg1, double arg2);}
static public interface
    DODD{double invokePrim(double arg0, Object arg1, double arg2);}
static public interface
    DL00{Object invokePrim(double arg0, long arg1, Object arg2);}
static public interface
    DL0L{long invokePrim(double arg0, long arg1, Object arg2);}
static public interface
    DL0D{double invokePrim(double arg0, long arg1, Object arg2);}
static public interface
    DLLO{Object invokePrim(double arg0, long arg1, long arg2);}
static public interface
    DLLL{long invokePrim(double arg0, long arg1, long arg2);}
static public interface
    DLLD{double invokePrim(double arg0, long arg1, long arg2);}
static public interface
    DLDO{Object invokePrim(double arg0, long arg1, double arg2);}
static public interface
    DLDL{long invokePrim(double arg0, long arg1, double arg2);}
static public interface
    DLDD{double invokePrim(double arg0, long arg1, double arg2);}
static public interface
    DD00{Object invokePrim(double arg0, double arg1, Object arg2);}
static public interface
    DDOL{long invokePrim(double arg0, double arg1, Object arg2);}
static public interface
```



```
00LLL{long invokePrim(Object arg0, Object arg1, long arg2,
                      long arg3);}
static public interface
    00LLD{double invokePrim(Object arg0, Object arg1, long arg2,
                           long arg3);}
static public interface
    00LDO{Object invokePrim(Object arg0, Object arg1, long arg2,
                           double arg3);}
static public interface
    00LDL{long invokePrim(Object arg0, Object arg1, long arg2,
                          double arg3);}
static public interface
    00LDD{double invokePrim(Object arg0, Object arg1, long arg2,
                           double arg3);}
static public interface
    00D00{Object invokePrim(Object arg0, Object arg1, double arg2,
                           Object arg3);}
static public interface
    00D0L{long invokePrim(Object arg0, Object arg1, double arg2,
                          Object arg3);}
static public interface
    00D0D{double invokePrim(Object arg0, Object arg1, double arg2,
                           Object arg3);}
static public interface
    00DL0{Object invokePrim(Object arg0, Object arg1, double arg2,
                           long arg3);}
static public interface
    00DLL{long invokePrim(Object arg0, Object arg1, double arg2,
                          long arg3);}
static public interface
    00DLD{double invokePrim(Object arg0, Object arg1, double arg2,
                           long arg3);}
static public interface
    00DD0{Object invokePrim(Object arg0, Object arg1, double arg2,
                           double arg3);}
static public interface
    00DDL{long invokePrim(Object arg0, Object arg1, double arg2,
                          double arg3);}
static public interface
    00DDD{double invokePrim(Object arg0, Object arg1, double arg2,
                           double arg3);}
static public interface
    0L000{Object invokePrim(Object arg0, long arg1, Object arg2,
                           Object arg3);}
static public interface
    0L00L{long invokePrim(Object arg0, long arg1, Object arg2,
                          Object arg3);}
static public interface
    0L00D{double invokePrim(Object arg0, long arg1, Object arg2,
                           Object arg3);}
```

```

static public interface
    OLOL0{Object invokePrim(Object arg0, long arg1, Object arg2,
                           long arg3);}
static public interface
    OLOL0{long invokePrim(Object arg0, long arg1, Object arg2,
                          long arg3);}
static public interface
    OLOLD{double invokePrim(Object arg0, long arg1, Object arg2,
                            long arg3);}
static public interface
    OLODO{Object invokePrim(Object arg0, long arg1, Object arg2,
                            double arg3);}
static public interface
    OLODL{long invokePrim(Object arg0, long arg1, Object arg2,
                          double arg3);}
static public interface
    OLODD{double invokePrim(Object arg0, long arg1, Object arg2,
                            double arg3);}
static public interface
    OLLOO{Object invokePrim(Object arg0, long arg1, long arg2,
                            Object arg3);}
static public interface
    OLLOL{long invokePrim(Object arg0, long arg1, long arg2,
                          Object arg3);}
static public interface
    OLLOD{double invokePrim(Object arg0, long arg1, long arg2,
                            Object arg3);}
static public interface
    OLLL0{Object invokePrim(Object arg0, long arg1, long arg2,
                            long arg3);}
static public interface
    OLLL0{long invokePrim(Object arg0, long arg1, long arg2,
                          long arg3);}
static public interface
    OLLLD{double invokePrim(Object arg0, long arg1, long arg2,
                            long arg3);}
static public interface
    OLLDO{Object invokePrim(Object arg0, long arg1, long arg2,
                            double arg3);}
static public interface
    OLLDL{long invokePrim(Object arg0, long arg1, long arg2,
                          double arg3);}
static public interface
    OLLDD{double invokePrim(Object arg0, long arg1, long arg2,
                            double arg3);}
static public interface
    OLD00{Object invokePrim(Object arg0, long arg1, double arg2,
                            Object arg3);}
static public interface
    OLDOL{long invokePrim(Object arg0, long arg1, double arg2,
                          Object arg3);}

```

```
        Object arg3);}
static public interface
    OLDD0{double invokePrim(Object arg0, long arg1, double arg2,
    Object arg3);}
static public interface
    OLDL0{Object invokePrim(Object arg0, long arg1, double arg2,
    long arg3);}
static public interface
    OLDLL{long invokePrim(Object arg0, long arg1, double arg2,
    long arg3);}
static public interface
    OLDDL{double invokePrim(Object arg0, long arg1, double arg2,
    long arg3);}
static public interface
    OLDD0{Object invokePrim(Object arg0, long arg1, double arg2,
    double arg3);}
static public interface
    OLDDL{long invokePrim(Object arg0, long arg1, double arg2,
    double arg3);}
static public interface
    OLDDD{double invokePrim(Object arg0, long arg1, double arg2,
    double arg3);}
static public interface
    OD000{Object invokePrim(Object arg0, double arg1, Object arg2,
    Object arg3);}
static public interface
    OD00L{long invokePrim(Object arg0, double arg1, Object arg2,
    Object arg3);}
static public interface
    OD00D{double invokePrim(Object arg0, double arg1, Object arg2,
    Object arg3);}
static public interface
    OD0L0{Object invokePrim(Object arg0, double arg1, Object arg2,
    long arg3);}
static public interface
    OD0LL{long invokePrim(Object arg0, double arg1, Object arg2,
    long arg3);}
static public interface
    OD0LD{double invokePrim(Object arg0, double arg1, Object arg2,
    long arg3);}
static public interface
    OD0D0{Object invokePrim(Object arg0, double arg1, Object arg2,
    double arg3);}
static public interface
    OD0DL{long invokePrim(Object arg0, double arg1, Object arg2,
    double arg3);}
static public interface
    OD0DD{double invokePrim(Object arg0, double arg1, Object arg2,
    double arg3);}
static public interface
```

```

ODL00{Object invokePrim(Object arg0, double arg1, long arg2,
    Object arg3);}
static public interface
    ODL01{long invokePrim(Object arg0, double arg1, long arg2,
        Object arg3);}
static public interface
    ODL02{double invokePrim(Object arg0, double arg1, long arg2,
        Object arg3);}
static public interface
    ODL03{Object invokePrim(Object arg0, double arg1, long arg2,
        long arg3);}
static public interface
    ODL04{long invokePrim(Object arg0, double arg1, long arg2,
        long arg3);}
static public interface
    ODL05{double invokePrim(Object arg0, double arg1, long arg2,
        long arg3);}
static public interface
    ODL06{Object invokePrim(Object arg0, double arg1, long arg2,
        double arg3);}
static public interface
    ODL07{long invokePrim(Object arg0, double arg1, long arg2,
        double arg3);}
static public interface
    ODL08{double invokePrim(Object arg0, double arg1, long arg2,
        double arg3);}
static public interface
    ODL09{Object invokePrim(Object arg0, double arg1, double arg2,
        Object arg3);}
static public interface
    ODL10{long invokePrim(Object arg0, double arg1, double arg2,
        Object arg3);}
static public interface
    ODL11{double invokePrim(Object arg0, double arg1, double arg2,
        Object arg3);}
static public interface
    ODL12{Object invokePrim(Object arg0, double arg1, double arg2,
        long arg3);}
static public interface
    ODL13{long invokePrim(Object arg0, double arg1, double arg2,
        long arg3);}
static public interface
    ODL14{double invokePrim(Object arg0, double arg1, double arg2,
        long arg3);}
static public interface
    ODL15{Object invokePrim(Object arg0, double arg1, double arg2,
        double arg3);}
static public interface
    ODL16{long invokePrim(Object arg0, double arg1, double arg2,
        double arg3);}

```

```
static public interface
    ODDDD{double invokePrim(Object arg0, double arg1, double arg2,
                           double arg3);}
static public interface
    L0000{Object invokePrim(long arg0, Object arg1, Object arg2,
                           Object arg3);}
static public interface
    L000L{long invokePrim(long arg0, Object arg1, Object arg2,
                          Object arg3);}
static public interface
    L000D{double invokePrim(long arg0, Object arg1, Object arg2,
                           Object arg3);}
static public interface
    L00L0{Object invokePrim(long arg0, Object arg1, Object arg2,
                           long arg3);}
static public interface
    L00LL{long invokePrim(long arg0, Object arg1, Object arg2,
                          long arg3);}
static public interface
    L00LD{double invokePrim(long arg0, Object arg1, Object arg2,
                           long arg3);}
static public interface
    L00DO{Object invokePrim(long arg0, Object arg1, Object arg2,
                           double arg3);}
static public interface
    L00DL{long invokePrim(long arg0, Object arg1, Object arg2,
                          double arg3);}
static public interface
    L00DD{double invokePrim(long arg0, Object arg1, Object arg2,
                           double arg3);}
static public interface
    L0L00{Object invokePrim(long arg0, Object arg1, long arg2,
                           Object arg3);}
static public interface
    L0L0L{long invokePrim(long arg0, Object arg1, long arg2,
                          Object arg3);}
static public interface
    L0L0D{double invokePrim(long arg0, Object arg1, long arg2,
                           Object arg3);}
static public interface
    L0L0O{Object invokePrim(long arg0, Object arg1, long arg2,
                           long arg3);}
static public interface
    L0L0L{long invokePrim(long arg0, Object arg1, long arg2,
                          long arg3);}
static public interface
    L0L0D{double invokePrim(long arg0, Object arg1, long arg2,
                           long arg3);}
static public interface
    L0L0O{Object invokePrim(long arg0, Object arg1, long arg2,
```

```
        double arg3);}
static public interface
    LOLDL{long invokePrim(long arg0, Object arg1, long arg2,
        double arg3);}
static public interface
    LOLDD{double invokePrim(long arg0, Object arg1, long arg2,
        double arg3);}
static public interface
    LODOO{Object invokePrim(long arg0, Object arg1, double arg2,
        Object arg3);}
static public interface
    LODOL{long invokePrim(long arg0, Object arg1, double arg2,
        Object arg3);}
static public interface
    LODOD{double invokePrim(long arg0, Object arg1, double arg2,
        Object arg3);}
static public interface
    LODL0{Object invokePrim(long arg0, Object arg1, double arg2,
        long arg3);}
static public interface
    LODLL{long invokePrim(long arg0, Object arg1, double arg2,
        long arg3);}
static public interface
    LODLD{double invokePrim(long arg0, Object arg1, double arg2,
        long arg3);}
static public interface
    LODD0{Object invokePrim(long arg0, Object arg1, double arg2,
        double arg3);}
static public interface
    LODDL{long invokePrim(long arg0, Object arg1, double arg2,
        double arg3);}
static public interface
    LODDD{double invokePrim(long arg0, Object arg1, double arg2,
        double arg3);}
static public interface
    LL000{Object invokePrim(long arg0, long arg1, Object arg2,
        Object arg3);}
static public interface
    LL00L{long invokePrim(long arg0, long arg1, Object arg2,
        Object arg3);}
static public interface
    LL00D{double invokePrim(long arg0, long arg1, Object arg2,
        Object arg3);}
static public interface
    LL0L0{Object invokePrim(long arg0, long arg1, Object arg2,
        long arg3);}
static public interface
    LL0L1{long invokePrim(long arg0, long arg1, Object arg2,
        long arg3);}
static public interface
```

```
    LLOLD{double invokePrim(long arg0, long arg1, Object arg2,
                           long arg3);}
static public interface
    LLODO{Object invokePrim(long arg0, long arg1, Object arg2,
                           double arg3);}
static public interface
    LLODL{long invokePrim(long arg0, long arg1, Object arg2,
                          double arg3);}
static public interface
    LLODD{double invokePrim(long arg0, long arg1, Object arg2,
                           double arg3);}
static public interface
    LLLOO{Object invokePrim(long arg0, long arg1, long arg2,
                           Object arg3);}
static public interface
    LLLOL{long invokePrim(long arg0, long arg1, long arg2,
                          Object arg3);}
static public interface
    LLLOD{double invokePrim(long arg0, long arg1, long arg2,
                           Object arg3);}
static public interface
    LLLL0{Object invokePrim(long arg0, long arg1, long arg2,
                           long arg3);}
static public interface
    LLLLL{long invokePrim(long arg0, long arg1, long arg2,
                          long arg3);}
static public interface
    LLLLLD{double invokePrim(long arg0, long arg1, long arg2,
                           long arg3);}
static public interface
    LLLDO{Object invokePrim(long arg0, long arg1, long arg2,
                           double arg3);}
static public interface
    LLLDL{long invokePrim(long arg0, long arg1, long arg2,
                          double arg3);}
static public interface
    LLLDD{double invokePrim(long arg0, long arg1, long arg2,
                           double arg3);}
static public interface
    LLDOO{Object invokePrim(long arg0, long arg1, double arg2,
                           Object arg3);}
static public interface
    LLDOL{long invokePrim(long arg0, long arg1, double arg2,
                          Object arg3);}
static public interface
    LLDD0{double invokePrim(long arg0, long arg1, double arg2,
                           Object arg3);}
static public interface
    LLDL0{Object invokePrim(long arg0, long arg1, double arg2,
                           long arg3);}
```

```

static public interface
    LLDLL{long invokePrim(long arg0, long arg1, double arg2,
                           long arg3);}
static public interface
    LLDDL{double invokePrim(long arg0, long arg1, double arg2,
                           long arg3);}
static public interface
    LLDD0{Object invokePrim(long arg0, long arg1, double arg2,
                           double arg3);}
static public interface
    LLDDL{long invokePrim(long arg0, long arg1, double arg2,
                           double arg3);}
static public interface
    LLDDD{double invokePrim(long arg0, long arg1, double arg2,
                           double arg3);}
static public interface
    LD000{Object invokePrim(long arg0, double arg1, Object arg2,
                           Object arg3);}
static public interface
    LD00L{long invokePrim(long arg0, double arg1, Object arg2,
                           Object arg3);}
static public interface
    LD00D{double invokePrim(long arg0, double arg1, Object arg2,
                           Object arg3);}
static public interface
    LD0L0{Object invokePrim(long arg0, double arg1, Object arg2,
                           long arg3);}
static public interface
    LD0LL{long invokePrim(long arg0, double arg1, Object arg2,
                           long arg3);}
static public interface
    LD0LD{double invokePrim(long arg0, double arg1, Object arg2,
                           long arg3);}
static public interface
    LD0D0{Object invokePrim(long arg0, double arg1, Object arg2,
                           double arg3);}
static public interface
    LD0DL{long invokePrim(long arg0, double arg1, Object arg2,
                           double arg3);}
static public interface
    LD0DD{double invokePrim(long arg0, double arg1, Object arg2,
                           double arg3);}
static public interface
    LDL00{Object invokePrim(long arg0, double arg1, long arg2,
                           Object arg3);}
static public interface
    LDL0L{long invokePrim(long arg0, double arg1, long arg2,
                           Object arg3);}
static public interface
    LDL0D{double invokePrim(long arg0, double arg1, long arg2,
                           Object arg3);}

```

```
        Object arg3);}

static public interface
    LDLL0{Object invokePrim(long arg0, double arg1, long arg2,
                           long arg3);}

static public interface
    LDLLL{long invokePrim(long arg0, double arg1, long arg2,
                           long arg3);}

static public interface
    LDLLD{double invokePrim(long arg0, double arg1, long arg2,
                           long arg3);}

static public interface
    LDLD0{Object invokePrim(long arg0, double arg1, long arg2,
                           double arg3);}

static public interface
    LDLDL{long invokePrim(long arg0, double arg1, long arg2,
                           double arg3);}

static public interface
    LDLDL0{double invokePrim(long arg0, double arg1, long arg2,
                           double arg3);}

static public interface
    LDD00{Object invokePrim(long arg0, double arg1, double arg2,
                           Object arg3);}

static public interface
    LDD0L{long invokePrim(long arg0, double arg1, double arg2,
                           Object arg3);}

static public interface
    LDD0D{double invokePrim(long arg0, double arg1, double arg2,
                           Object arg3);}

static public interface
    LDDL0{Object invokePrim(long arg0, double arg1, double arg2,
                           long arg3);}

static public interface
    LDDLL{long invokePrim(long arg0, double arg1, double arg2,
                           long arg3);}

static public interface
    LDDLD{double invokePrim(long arg0, double arg1, double arg2,
                           long arg3);}

static public interface
    LDDDO{Object invokePrim(long arg0, double arg1, double arg2,
                           long arg3);}

static public interface
    LDDDL{long invokePrim(long arg0, double arg1, double arg2,
                           long arg3);}

static public interface
    LDDDD{double invokePrim(long arg0, double arg1, double arg2,
                           long arg3);}

static public interface
    LDDDD0{Object invokePrim(long arg0, double arg1, double arg2,
                           double arg3);}

static public interface
    LDDDDL{long invokePrim(long arg0, double arg1, double arg2,
                           double arg3);}

static public interface
    LDDDD0{double invokePrim(long arg0, double arg1, double arg2,
                           double arg3);}

static public interface
    D0000{Object invokePrim(double arg0, Object arg1, Object arg2,
                           Object arg3);}

static public interface
```

```

D000L{long invokePrim(double arg0, Object arg1, Object arg2,
                      Object arg3);}
static public interface
D000D{double invokePrim(double arg0, Object arg1, Object arg2,
                        Object arg3);}
static public interface
D00L0{Object invokePrim(double arg0, Object arg1, Object arg2,
                        long arg3);}
static public interface
D00LL{long invokePrim(double arg0, Object arg1, Object arg2,
                      long arg3);}
static public interface
D00LD{double invokePrim(double arg0, Object arg1, Object arg2,
                        long arg3);}
static public interface
D00DO{Object invokePrim(double arg0, Object arg1, Object arg2,
                        double arg3);}
static public interface
D00DL{long invokePrim(double arg0, Object arg1, Object arg2,
                      double arg3);}
static public interface
D00DD{double invokePrim(double arg0, Object arg1, Object arg2,
                        double arg3);}
static public interface
D0L00{Object invokePrim(double arg0, Object arg1, long arg2,
                        Object arg3);}
static public interface
D0L0L{long invokePrim(double arg0, Object arg1, long arg2,
                      Object arg3);}
static public interface
D0L0D{double invokePrim(double arg0, Object arg1, long arg2,
                        Object arg3);}
static public interface
D0L0O{Object invokePrim(double arg0, Object arg1, long arg2,
                        long arg3);}
static public interface
D0L0L{long invokePrim(double arg0, Object arg1, long arg2,
                      long arg3);}
static public interface
D0L0D{double invokePrim(double arg0, Object arg1, long arg2,
                        long arg3);}
static public interface
D0L0O{Object invokePrim(double arg0, Object arg1, long arg2,
                        double arg3);}
static public interface
D0LDL{long invokePrim(double arg0, Object arg1, long arg2,
                      double arg3);}
static public interface
D0LDD{double invokePrim(double arg0, Object arg1, long arg2,
                        double arg3);}

```

```
static public interface
    DODOO{Object invokePrim(double arg0, Object arg1, double arg2,
        Object arg3);}
static public interface
    DODOL{long invokePrim(double arg0, Object arg1, double arg2,
        Object arg3);}
static public interface
    DODOD{double invokePrim(double arg0, Object arg1, double arg2,
        Object arg3);}
static public interface
    DODLO{Object invokePrim(double arg0, Object arg1, double arg2,
        long arg3);}
static public interface
    DODLL{long invokePrim(double arg0, Object arg1, double arg2,
        long arg3);}
static public interface
    DODLD{double invokePrim(double arg0, Object arg1, double arg2,
        long arg3);}
static public interface
    DODDO{Object invokePrim(double arg0, Object arg1, double arg2,
        double arg3);}
static public interface
    DODDL{long invokePrim(double arg0, Object arg1, double arg2,
        double arg3);}
static public interface
    DODDD{double invokePrim(double arg0, Object arg1, double arg2,
        double arg3);}
static public interface
    DL000{Object invokePrim(double arg0, long arg1, Object arg2,
        Object arg3);}
static public interface
    DL00L{long invokePrim(double arg0, long arg1, Object arg2,
        Object arg3);}
static public interface
    DL00D{double invokePrim(double arg0, long arg1, Object arg2,
        Object arg3);}
static public interface
    DL0L0{Object invokePrim(double arg0, long arg1, Object arg2,
        long arg3);}
static public interface
    DL0LL{long invokePrim(double arg0, long arg1, Object arg2,
        long arg3);}
static public interface
    DL0LD{double invokePrim(double arg0, long arg1, Object arg2,
        long arg3);}
static public interface
    DL0DO{Object invokePrim(double arg0, long arg1, Object arg2,
        double arg3);}
static public interface
    DL0DL{long invokePrim(double arg0, long arg1, Object arg2,
```

```

        double arg3);}
static public interface
    DL00D{double invokePrim(double arg0, long arg1, Object arg2,
        double arg3);}
static public interface
    DLL00{Object invokePrim(double arg0, long arg1, long arg2,
        Object arg3);}
static public interface
    DLL0L{long invokePrim(double arg0, long arg1, long arg2,
        Object arg3);}
static public interface
    DLL0D{double invokePrim(double arg0, long arg1, long arg2,
        Object arg3);}
static public interface
    DLLL0{Object invokePrim(double arg0, long arg1, long arg2,
        long arg3);}
static public interface
    DLLLL{long invokePrim(double arg0, long arg1, long arg2,
        long arg3);}
static public interface
    DLLLD{double invokePrim(double arg0, long arg1, long arg2,
        long arg3);}
static public interface
    DLLD0{Object invokePrim(double arg0, long arg1, long arg2,
        double arg3);}
static public interface
    DLLDL{long invokePrim(double arg0, long arg1, long arg2,
        double arg3);}
static public interface
    DLLDD{double invokePrim(double arg0, long arg1, long arg2,
        double arg3);}
static public interface
    DLD00{Object invokePrim(double arg0, long arg1, double arg2,
        Object arg3);}
static public interface
    DLDOL{long invokePrim(double arg0, long arg1, double arg2,
        Object arg3);}
static public interface
    DLDOD{double invokePrim(double arg0, long arg1, double arg2,
        Object arg3);}
static public interface
    DL0L0{Object invokePrim(double arg0, long arg1, double arg2,
        long arg3);}
static public interface
    DL0L0L{long invokePrim(double arg0, long arg1, double arg2,
        long arg3);}
static public interface
    DL0L0D{double invokePrim(double arg0, long arg1, double arg2,
        long arg3);}
static public interface
    DL0L0O{Object invokePrim(double arg0, long arg1, double arg2,
        long arg3);}

```

```
DLDDO{Object invokePrim(double arg0, long arg1, double arg2,
    double arg3);}
static public interface
    DLDDL{long invokePrim(double arg0, long arg1, double arg2,
        double arg3);}
static public interface
    DLDDD{double invokePrim(double arg0, long arg1, double arg2,
        double arg3);}
static public interface
    DD000{Object invokePrim(double arg0, double arg1, Object arg2,
        Object arg3);}
static public interface
    DD00L{long invokePrim(double arg0, double arg1, Object arg2,
        Object arg3);}
static public interface
    DD00D{double invokePrim(double arg0, double arg1, Object arg2,
        Object arg3);}
static public interface
    DD0L0{Object invokePrim(double arg0, double arg1, Object arg2,
        long arg3);}
static public interface
    DD0LL{long invokePrim(double arg0, double arg1, Object arg2,
        long arg3);}
static public interface
    DD0LD{double invokePrim(double arg0, double arg1, Object arg2,
        long arg3);}
static public interface
    DD0DO{Object invokePrim(double arg0, double arg1, Object arg2,
        double arg3);}
static public interface
    DD0DL{long invokePrim(double arg0, double arg1, Object arg2,
        double arg3);}
static public interface
    DD0DD{double invokePrim(double arg0, double arg1, Object arg2,
        double arg3);}
static public interface
    DDL00{Object invokePrim(double arg0, double arg1, long arg2,
        Object arg3);}
static public interface
    DDL0L{long invokePrim(double arg0, double arg1, long arg2,
        Object arg3);}
static public interface
    DDL0D{double invokePrim(double arg0, double arg1, long arg2,
        Object arg3);}
static public interface
    DDLLO{Object invokePrim(double arg0, double arg1, long arg2,
        long arg3);}
static public interface
    DLLL0{long invokePrim(double arg0, double arg1, long arg2,
        long arg3);}
```

```

static public interface
    DDLLD{double invokePrim(double arg0, double arg1, long arg2,
                           long arg3);}
static public interface
    DDLDO{Object invokePrim(double arg0, double arg1, long arg2,
                           double arg3);}
static public interface
    DDLDL{long invokePrim(double arg0, double arg1, long arg2,
                          double arg3);}
static public interface
    DDLDD{double invokePrim(double arg0, double arg1, long arg2,
                           double arg3);}
static public interface
    DDDOO{Object invokePrim(double arg0, double arg1, double arg2,
                           Object arg3);}
static public interface
    DDDOL{long invokePrim(double arg0, double arg1, double arg2,
                          Object arg3);}
static public interface
    DDDOD{double invokePrim(double arg0, double arg1, double arg2,
                           Object arg3);}
static public interface
    DDDLO{Object invokePrim(double arg0, double arg1, double arg2,
                           long arg3);}
static public interface
    DDDLL{long invokePrim(double arg0, double arg1, double arg2,
                          long arg3);}
static public interface
    DDDLD{double invokePrim(double arg0, double arg1, double arg2,
                           long arg3);}
static public interface
    DDDDO{Object invokePrim(double arg0, double arg1, double arg2,
                           double arg3);}
static public interface
    DDDDL{long invokePrim(double arg0, double arg1, double arg2,
                          double arg3);}
static public interface
    DDDDD{double invokePrim(double arg0, double arg1, double arg2,
                           double arg3);}
}

```

9.36 IKeywordLookup.java

— IKeywordLookup.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Oct 31, 2009 */

package clojure.lang;

public interface IKeywordLookup{
ILookupThunk getLookupThunk(Keyword k);
}
```

9.37 ILookup.java

— ILookup.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Aug 2, 2009 */

package clojure.lang;

public interface ILookup{
Object valAt(Object key);

Object valAt(Object key, Object notFound);
}
```

9.38 ILookupSite.java

— ILookupSite.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Nov 2, 2009 */

package clojure.lang;

public interface ILookupSite{
```

```
ILookupThunk fault(Object target);  
}
```

—————

9.39 ILookupThunk.java

— ILookupThunk.java —

```
/*  
\getchunk{Clojure Copyright}  
*/  
/* rich Nov 2, 2009 */  
  
package clojure.lang;  
  
public interface ILookupThunk{  
  
Object get(Object target);  
}
```

—————

9.40 IMapEntry.java

(Map.Entry [1723])
— IMapEntry.java —

```
/*  
\getchunk{Clojure Copyright}  
*/  
package clojure.lang;  
  
import java.util.Map;  
  
public interface IMapEntry extends Map.Entry{  
Object key();  
  
Object val();  
}
```

—————

9.41 IMeta.java

— IMeta.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Dec 31, 2008 */

package clojure.lang;

public interface IMeta {
    IPersistentMap meta();
}
```

9.42 Indexed.java

(Counted [768])

— Indexed.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich May 24, 2009 */

package clojure.lang;

public interface Indexed extends Counted{
    Object nth(int i);

    Object nth(int i, Object notFound);
}
```

9.43 IndexedSeq.java

(ISeq [805]) (Counted [768])

— IndexedSeq.java —

```
/*
\getchunk{Clojure Copyright}
```

```
*/
package clojure.lang;

public interface IndexedSeq extends ISeq, Counted{
    public int index();
}
```

9.44 IObj.java

(IMeta [799])
— IObj.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

public interface IObj extends IMeta {
    public IObj withMeta(IPersistentMap meta);
}
```

9.45 IPersistentCollection.java

(Seqable [1140])
— IPersistentCollection.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

public interface IPersistentCollection extends Seqable {
    int count();
    IPersistentCollection cons(Object o);
    IPersistentCollection empty();
```

```
boolean equiv(Object o);  
}
```

— — —

9.46 IPersistentList.java

(Sequential [1140]) (IPersistentStack [802])
— IPersistentList.java —

```
/*  
\getchunk{Clojure Copyright}  
*/  
package clojure.lang;  
  
public interface IPersistentList extends Sequential, IPersistentStack{  
}
```

— — —

9.47 IPersistentMap.java

(Iterable [1723]) (Associative [576]) (Counted [768])
— IPersistentMap.java —

```
/*  
\getchunk{Clojure Copyright}  
*/  
package clojure.lang;  
  
public interface IPersistentMap extends Iterable, Associative, Counted{  
  
    IPersistentMap assoc(Object key, Object val);  
  
    IPersistentMap assocEx(Object key, Object val) throws Exception;  
  
    IPersistentMap without(Object key) throws Exception;  
}
```

— — —

9.48 IPersistentSet.java

(IPersistentCollection [800]) (Counted [768])
 — IPersistentSet.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 3, 2008 */

package clojure.lang;

public interface IPersistentSet extends IPersistentCollection, Counted{
    public IPersistentSet disjoin(Object key) throws Exception;
    public boolean contains(Object key);
    public Object get(Object key);
}
```

9.49 IPersistentStack.java

(IPersistentCollection [800])
 — IPersistentStack.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Sep 19, 2007 */

package clojure.lang;

public interface IPersistentStack extends IPersistentCollection{
    Object peek();

    IPersistentStack pop();
}
```

9.50 IPersistentVector.java

(Associative [576]) (Sequential [1140]) (Reversible [1094]) (Indexed [799]) (IPersistentStack [802])
 — IPersistentVector.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

public interface IPersistentVector
    extends Associative, Sequential, IPersistentStack,
           Reversible, Indexed{
    int length();

    IPersistentVector assocN(int i, Object val);

    IPersistentVector cons(Object o);

}

```

—————

9.51 IPromiseImpl.java

— IPromiseImpl.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

public interface IPromiseImpl {
    boolean hasValue();
}

```

—————

9.52 IProxy.java

— IProxy.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Feb 27, 2008 */

package clojure.lang;

```

```

public interface IProxy{

    public void __initClojureFnMappings(IPersistentMap m);
    public void __updateClojureFnMappings(IPersistentMap m);
    public IPersistentMap __getClojureFnMappings();

}

```

9.53 IReduce.java

— IReduce.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jun 11, 2008 */

package clojure.lang;

public interface IReduce{
    Object reduce(IFn f) throws Exception;

    Object reduce(IFn f, Object start) throws Exception;
}

```

9.54 IReference.java

(IMeta [799])

— IReference.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Dec 31, 2008 */

package clojure.lang;

public interface IReference extends IMeta {
    IPersistentMap alterMeta(IFn alter, ISeq args) throws Exception;
    IPersistentMap resetMeta(IPersistentMap m);
}

```

9.55 IRef.java

(IDeref [773])
— IRef.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Nov 18, 2007 */

package clojure.lang;

public interface IRef extends IDeref{

    void setValidator(IFn vf);

    IFn getValidator();

    IPersistentMap getWatches();

    IRef addWatch(Object key, IFn callback);

    IRef removeWatch(Object key);

}
```

9.56 ISeq.java

(IPersistentCollection [800]) (Sequential [1140])
— ISeq.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

/**
 * A persistent, functional, sequence interface
 * <p/>
 * ISeqs are immutable values, i.e. neither first(), nor rest() changes
 * or invalidates the ISeq
 */
```

```
public interface ISeq extends IPersistentCollection, Sequential{
    Object first();
    ISeq next();
    ISeq more();
    ISeq cons(Object o);
}
```

9.57 IteratorSeq.java

(ASeq [571])
— IteratorSeq.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.io.IOException;
import java.io.NotSerializableException;
import java.util.Iterator;

public class IteratorSeq extends ASeq{
    final Iterator iter;
    final State state;

    static class State{
        volatile Object val;
        volatile Object _rest;
    }

    public static IteratorSeq create(Iterator iter){
        if(iter.hasNext())
            return new IteratorSeq(iter);
        return null;
    }

    IteratorSeq(Iterator iter){
        this.iter = iter;
        state = new State();
        this.state.val = state;
        this.state._rest = state;
    }
}
```

```

}

IteratorSeq(IPersistentMap meta, Iterator iter, State state){
    super(meta);
    this.iter = iter;
    this.state = state;
}

public Object first(){
    if(state.val == state)
        synchronized(state)
    {
        if(state.val == state)
            state.val = iter.next();
    }
    return state.val;
}

public ISeq next(){
    if(state._rest == state)
        synchronized(state)
    {
        if(state._rest == state)
        {
            first();
            state._rest = create(iter);
        }
    }
    return (ISeq) state._rest;
}

public IteratorSeq withMeta(IPersistentMap meta){
    return new IteratorSeq(meta, iter, state);
}

private void writeObject (java.io.ObjectOutputStream out)
throws IOException {
    throw new NotSerializableException(getClass().getName());
}
}

```

—————

9.58 ITransientAssociative.java

(ITransientCollection [808]) (ILookup [797])
 — ITransientAssociative.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 17, 2009 */

package clojure.lang;

public interface ITransientAssociative
  extends ITransientCollection, ILookup{

ITransientAssociative assoc(Object key, Object val);
}

```

9.59 ITransientCollection.java

— ITransientCollection.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 17, 2009 */

package clojure.lang;

public interface ITransientCollection{

ITransientCollection conj(Object val);

IPersistentCollection persistent();
}

```

9.60 ITransientMap.java

(ITransientAssociative [807]) (Counted [768])
— ITransientMap.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 17, 2009 */

```

```

package clojure.lang;

public interface ITransientMap extends ITransientAssociative, Counted{
    ITransientMap assoc(Object key, Object val);
    ITransientMap without(Object key);
    IPersistentMap persistent();
}

```

9.61 ITransientSet.java

(ITransientCollection [808]) (Counted [768])
 — **ITransientSet.java** —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 3, 2008 */

package clojure.lang;

public interface ITransientSet extends ITransientCollection, Counted{
    public ITransientSet disjoin(Object key) throws Exception;
    public boolean contains(Object key);
    public Object get(Object key);
}

```

9.62 ITransientVector.java

(ITransientAssociative [807]) (Indexed [799])
 — **ITransientVector.java** —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 17, 2009 */

package clojure.lang;

public interface ITransientVector extends ITransientAssociative, Indexed{

```

```
ITransientVector assocN(int i, Object val);

ITransientVector pop();
}
```

9.63 Keyword.java

(IFn [774]) (Comparable [1723]) (Named [861]) (Serializable [1723])
— Keyword.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 29, 2006 10:39:05 AM */

package clojure.lang;

import java.io.ObjectStreamException;
import java.io.Serializable;
import java.util.concurrent.ConcurrentHashMap;
import java.lang.ref.ReferenceQueue;
import java.lang.ref.SoftReference;

public final class Keyword
    implements IFn, Comparable, Named, Serializable {

    private static ConcurrentHashMap<Symbol, SoftReference<Keyword>> table =
        new ConcurrentHashMap();
    static final ReferenceQueue rq = new ReferenceQueue();
    public final Symbol sym;
    final int hash;

    public static Keyword intern(Symbol sym){
        if(sym.meta() != null)
            sym = (Symbol) sym.withMeta(null);
        Util.clearCache(rq, table);
        Keyword k = new Keyword(sym);
        SoftReference<Keyword> existingRef =
            table.putIfAbsent(sym, new SoftReference<Keyword>(k,rq));
        if(existingRef == null)
            return k;
        Keyword existingk = existingRef.get();
        if(existingk != null)
            return existingk;
    }
}
```

```
//entry died in the interim, do over
table.remove(sym, existingRef);
return intern(sym);
}

public static Keyword intern(String ns, String name){
    return intern(Symbol.intern(ns, name));
}

public static Keyword intern(String nsname){
    return intern(Symbol.intern(nsname));
}

private Keyword(Symbol sym){
    this.sym = sym;
    hash = sym.hashCode() + 0x9e3779b9;
}

public static Keyword find(Symbol sym){
    SoftReference<Keyword> ref = table.get(sym);
    if (ref != null)
        return ref.get();
    else
        return null;
}

public static Keyword find(String ns, String name){
    return find(Symbol.intern(ns, name));
}

public static Keyword find(String nsname){
    return find(Symbol.intern(nsname));
}

public final int hashCode(){
    return hash;
}

public String toString(){
    return ":" + sym;
}

public Object throwArity(){
    throw new IllegalArgumentException(
        "Wrong number of args passed to keyword: " + toString());
}

public Object call() throws Exception{
    return throwArity();
}
```

```
public void run(){
    throw new UnsupportedOperationException();
}

public Object invoke() throws Exception{
    return throwArity();
}

public int compareTo(Object o){
    return sym.compareTo(((Keyword) o).sym);
}

public String getNamespace(){
    return sym.getNamespace();
}

public String getName(){
    return sym.getName();
}

private Object readResolve() throws ObjectStreamException{
    return intern(sym);
}

/**
 * Indexer implements IFn for attr access
 *
 * @param obj - must be IPersistentMap
 * @return the value at the key or nil if not found
 * @throws Exception
 */
final public Object invoke(Object obj) throws Exception{
    if(obj instanceof ILookup)
        return ((ILookup)obj).valAt(this);
    return RT.get(obj, this);
}

final public Object invoke(Object obj, Object notFound)
throws Exception{
    if(obj instanceof ILookup)
        return ((ILookup)obj).valAt(this,notFound);
    return RT.get(obj, this, notFound);
}

public Object invoke(Object arg1, Object arg2, Object arg3)
throws Exception{
    return throwArity();
}
```

```
public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
```

```
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16)
throws Exception{
    return throwArity();
}
```

```
public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20)
throws Exception{
    return throwArity();
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20, Object... args)
```

```

        throws Exception{
            return throwArity();
        }

    public Object applyTo(ISeq arglist) throws Exception{
        return AFn.applyToHelper(this, arglist);
    }

}

```

9.64 KeywordLookupSite.java

(ILookupSite [797]) (ILookupThunk [798])
— KeywordLookupSite.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Nov 2, 2009 */

package clojure.lang;

public final class KeywordLookupSite
    implements ILookupSite, ILookupThunk{

    final Keyword k;

    public KeywordLookupSite(Keyword k){
        this.k = k;
    }

    public ILookupThunk fault(Object target){
        if(target instanceof IKeywordLookup)
        {
            return install(target);
        }
        else if(target instanceof ILookup)
        {
            return ilookupThunk(target.getClass());
        }
        return this;
    }

    public Object get(Object target){

```

```

        if(target instanceof IKeywordLookup || target instanceof ILookup)
            return this;
        return RT.get(target,k);
    }

private ILookupThunk ilookupThunk(final Class c){
    return new ILookupThunk(){
        public Object get(Object target){
            if(target != null && target.getClass() == c)
                return ((ILookup) target).valAt(k);
            return this;
        }
    };
}

private ILookupThunk install(Object target){
    ILookupThunk t = ((IKeywordLookup)target).getLookupThunk(k);
    if(t != null)
        return t;
    return ilookupThunk(target.getClass());
}
}

```

9.65 LazilyPersistentVector.java

— LazilyPersistentVector.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich May 14, 2008 */

package clojure.lang;

import java.util.Collection;

public class LazilyPersistentVector{

static public IPersistentVector createOwning(Object... items){
    if(items.length == 0)
        return PersistentVector.EMPTY;
    else if(items.length <= 32)
        return
            new PersistentVector(items.length, 5,

```

```

        PersistentVector.EMPTY_NODE,items);
    return PersistentVector.create(items);
}

static public IPersistentVector create(Collection coll){
    if(!(coll instanceof ISeq) && coll.size() <= 32)
        return createOwning(coll.toArray());
    return PersistentVector.create(RT.seq(coll));
}
}

```

9.66 LazySeq.java

(Obj [947]) (ISeq [805]) (List [1723])
— LazySeq.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jan 31, 2009 */

package clojure.lang;

import java.util.*;

public final class LazySeq extends Obj implements ISeq, List{

    private IFn fn;
    private Object sv;
    private ISeq s;

    public LazySeq(IFn fn){
        this.fn = fn;
    }

    private LazySeq(IPersistentMap meta, ISeq s){
        super(meta);
        this.fn = null;
        this.s = s;
    }

    public Obj withMeta(IPersistentMap meta){
        return new LazySeq(meta, seq());
    }
}

```

```

final synchronized Object sval(){
    if(fn != null)
    {
        try
        {
            sv = fn.invoke();
            fn = null;
        }
        catch(RuntimeException e)
        {
            throw e;
        }
        catch(Exception e)
        {
            throw new RuntimeException(e);
        }
    }
    if(sv != null)
        return sv;
    return s;
}

final synchronized public ISeq seq(){
    sval();
    if(sv != null)
    {
        Object ls = sv;
        sv = null;
        while(ls instanceof LazySeq)
        {
            ls = ((LazySeq)ls).sval();
        }
        s = RT.seq(ls);
    }
    return s;
}

public int count(){
    int c = 0;
    for(ISeq s = seq(); s != null; s = s.next())
        ++c;
    return c;
}

public Object first(){
    seq();
    if(s == null)
        return null;
    return s.first();
}

```

```

public ISeq next(){
    seq();
    if(s == null)
        return null;
    return s.next();
}

public ISeq more(){
    seq();
    if(s == null)
        return PersistentList.EMPTY;
    return s.more();
}

public ISeq cons(Object o){
    return RT.cons(o, seq());
}

public IPersistentCollection empty(){
    return PersistentList.EMPTY;
}

public boolean equiv(Object o){
    return equals(o);
}

public int hashCode(){
    ISeq s = seq();
    if(s == null)
        return 1;
    return Util.hash(seq());
}

public boolean equals(Object o){
    ISeq s = seq();
    if(s != null)
        return s.equiv(o);
    else
        return (o instanceof Sequential ||
                o instanceof List) &&
                RT.seq(o) == null;
}
}

// java.util.Collection implementation

public Object[] toArray(){
    return RT.seqToArray(seq());
}

```

```
public boolean add(Object o){
    throw new UnsupportedOperationException();
}

public boolean remove(Object o){
    throw new UnsupportedOperationException();
}

public boolean addAll(Collection c){
    throw new UnsupportedOperationException();
}

public void clear(){
    throw new UnsupportedOperationException();
}

public boolean retainAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean removeAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean containsAll(Collection c){
    for(Object o : c)
    {
        if(!contains(o))
            return false;
    }
    return true;
}

public Object[] toArray(Object[] a){
    if(a.length >= count())
    {
        ISeq s = seq();
        for(int i = 0; s != null; ++i, s = s.next())
        {
            a[i] = s.first();
        }
        if(a.length > count())
            a[count()] = null;
    }
    else
        return toArray();
}
```

```
public int size(){
    return count();
}

public boolean isEmpty(){
    return seq() == null;
}

public boolean contains(Object o){
    for(ISeq s = seq(); s != null; s = s.next())
    {
        if(Util.equiv(s.first(), o))
            return true;
    }
    return false;
}

public Iterator iterator(){
    return new SeqIterator(seq());
}

////////// List stuff ///////////
private List reify(){
    return new ArrayList(this);
}

public List subList(int fromIndex, int toIndex){
    return reify().subList(fromIndex, toIndex);
}

public Object set(int index, Object element){
    throw new UnsupportedOperationException();
}

public Object remove(int index){
    throw new UnsupportedOperationException();
}

public int indexOf(Object o){
    ISeq s = seq();
    for(int i = 0; s != null; s = s.next(), i++)
    {
        if(Util.equiv(s.first(), o))
            return i;
    }
    return -1;
}

public int lastIndexOf(Object o){
    return reify().lastIndexOf(o);
}
```

```

}

public ListIterator listIterator(){
    return reify().listIterator();
}

public ListIterator listIterator(int index){
    return reify().listIterator(index);
}

public Object get(int index){
    return RT.nth(this, index);
}

public void add(int index, Object element){
    throw new UnsupportedOperationException();
}

public boolean addAll(int index, Collection c){
    throw new UnsupportedOperationException();
}

}

```

9.67 LineNumberingPushbackReader.java

(PushbackReader [1723])

— LineNumberingPushbackReader.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.io.PushbackReader;
import java.io.Reader;
import java.io.LineNumberReader;
import java.io.IOException;

public class LineNumberingPushbackReader extends PushbackReader{

    // This class is a PushbackReader that wraps a LineNumberReader. The code
    // here to handle line terminators only mentions '\n' because
    // LineNumberReader collapses all occurrences of CR, LF, and CRLF into a

```

```
// single '\n'.
```

```
private static final int newline = (int) '\n';
```

```
private boolean _atLineStart = true;
```

```
private boolean _prev;
```

```
public LineNumberingPushbackReader(Reader r){
```

```
    super(new LineNumberReader(r));
```

```
}
```

```
public int getLineNumber(){
```

```
    return ((LineNumberReader) in).getLineNumber() + 1;
```

```
}
```

```
public int read() throws IOException{
```

```
    int c = super.read();
```

```
    _prev = _atLineStart;
```

```
    _atLineStart = (c == newline) || (c == -1);
```

```
    return c;
```

```
}
```

```
public void unread(int c) throws IOException{
```

```
    super.unread(c);
```

```
    _atLineStart = _prev;
```

```
}
```

```
public String readLine() throws IOException{
```

```
    int c = read();
```

```
    String line;
```

```
    switch (c) {
```

```
    case -1:
```

```
        line = null;
```

```
        break;
```

```
    case newline:
```

```
        line = "";
```

```
        break;
```

```
    default:
```

```
        String first = String.valueOf((char) c);
```

```
        String rest = ((LineNumberReader)in).readLine();
```

```
        line = (rest == null) ? first : first + rest;
```

```
        _prev = false;
```

```
        _atLineStart = true;
```

```
        break;
```

```
}
```

```
    return line;
```

```
}
```

```
public boolean atLineStart(){
```

```
    return _atLineStart;
```

```

}
}
```



9.68 LispReader.java

— LispReader.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.io.*;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.math.BigInteger;
import java.math.BigDecimal;
import java.lang.*;

public class LispReader{

    static final Symbol QUOTE = Symbol.intern("quote");
    static final Symbol THE_VAR = Symbol.intern("var");
    //static Symbol SYNTAX_QUOTE = Symbol.intern(null, "syntax-quote");
    static Symbol UNQUOTE = Symbol.intern("clojure.core", "unquote");
    static Symbol UNQUOTE_SPLICING =
        Symbol.intern("clojure.core", "unquote-splicing");
    static Symbol CONCAT = Symbol.intern("clojure.core", "concat");
    static Symbol SEQ = Symbol.intern("clojure.core", "seq");
    static Symbol LIST = Symbol.intern("clojure.core", "list");
    static Symbol APPLY = Symbol.intern("clojure.core", "apply");
    static Symbol HASHMAP = Symbol.intern("clojure.core", "hash-map");
    static Symbol HASHSET = Symbol.intern("clojure.core", "hash-set");
    static Symbol VECTOR = Symbol.intern("clojure.core", "vector");
    static Symbol WITH_META = Symbol.intern("clojure.core", "with-meta");
    static Symbol META = Symbol.intern("clojure.core", "meta");
    static Symbol DEREF = Symbol.intern("clojure.core", "deref");
    //static Symbol DEREF_BANG = Symbol.intern("clojure.core", "deref!");

    \getchunk{LispReader macros statement}
    \getchunk{LispReader dispatchMacros statement}
    //static Pattern symbolPat =
```

```

// Pattern.compile("[:]?([\\D&&[^:/]][^:/]*)?[\\D&&[^:/]][^:/]*");
static Pattern symbolPat =
    Pattern.compile("[:]?([\\D&&[^/]]*)?([\\D&&[^/]][^/]*)");
//static Pattern varPat =
// Pattern.compile("(\\D&&[^:\\.])[^:\\.]*:(\\D&&[^:\\.])[^:\\.]*");
//static Pattern intPat = Pattern.compile("-+?[0-9]+\\.?");
static Pattern intPat =
    Pattern.compile(
        "(-+?)\\((0|[1-9][0-9]*|0[xX](0-9A-Fa-f+)|0([0-7]+)+|
        |([1-9][0-9]?)[rR](0-9A-Za-z+)|0[0-9]+)(N)?");
static Pattern ratioPat = Pattern.compile("(-+?[0-9]+)/([0-9]+)");
static Pattern floatPat =
    Pattern.compile("(-+?[0-9]+(\\.\\.[0-9]*)([eE]-+?[0-9]+)?)(M)?");
static final Symbol SLASH = Symbol.intern("/");
static final Symbol CLOJURE_SLASH = Symbol.intern("clojure.core","");
//static Pattern accessorPat = Pattern.compile("\\.[a-zA-Z_]\\w*");
//static Pattern instanceMemberPat =
// Pattern.compile("\\.([a-zA-Z_]\\w*)\\.(a-zA-Z_\\w*)");
//static Pattern staticMemberPat =
// Pattern.compile("\\.([a-zA-Z_]\\w*)\\.(a-zA-Z_\\w*)");
//static Pattern classNamePat =
// Pattern.compile("\\.([a-zA-Z_]\\w*)\\.");

//symbol->gensymbol
static Var GENSYM_ENV = Var.create(null).setDynamic();
//sorted-map num->gensymbol
static Var ARG_ENV = Var.create(null).setDynamic();

static
{
\getchunk{LispReader Syntax Macro Table}

\getchunk{LispReader Dispatch Macro Table}
}

static boolean isWhitespace(int ch){
    return Character.isWhitespace(ch) || ch == ',';
}

static void unread(PushbackReader r, int ch) throws IOException{
    if(ch != -1)
        r.unread(ch);
}

public static class ReaderException extends Exception{
    final int line;

    public ReaderException(int line, Throwable cause){
        super(cause);
        this.line = line;
    }
}

```

```

        }
    }

\getchunk{LispReader read method}

static private String readToken(PushbackReader r, char initch)
throws Exception{
    StringBuilder sb = new StringBuilder();
    sb.append(initch);

    for(; ;)
    {
        int ch = r.read();
        if(ch == -1 || isWhitespace(ch) || isTerminatingMacro(ch))
        {
            unread(r, ch);
            return sb.toString();
        }
        sb.append((char) ch);
    }
}

static private Object readNumber(PushbackReader r, char initch)
throws Exception{
    StringBuilder sb = new StringBuilder();
    sb.append(initch);

    for(; ;)
    {
        int ch = r.read();
        if(ch == -1 || isWhitespace(ch) || isMacro(ch))
        {
            unread(r, ch);
            break;
        }
        sb.append((char) ch);
    }

    String s = sb.toString();
    Object n = matchNumber(s);
    if(n == null)
        throw new NumberFormatException("Invalid number: " + s);
    return n;
}

static private int readUnicodeChar(String token,
                                  int offset,
                                  int length,
                                  int base)
throws Exception{

```

```

if(token.length() != offset + length)
    throw new IllegalArgumentException(
        "Invalid unicode character: \\\" + token);
int uc = 0;
for(int i = offset; i < offset + length; ++i)
{
    int d = Character.digit(token.charAt(i), base);
    if(d == -1)
        throw new IllegalArgumentException(
            "Invalid digit: " + (char) d);
    uc = uc * base + d;
}
return (char) uc;
}

static private int readUnicodeChar(PushbackReader r,
                                    int initch,
                                    int base,
                                    int length,
                                    boolean exact)
throws Exception{
    int uc = Character.digit(initch, base);
    if(uc == -1)
        throw new IllegalArgumentException("Invalid digit: " + initch);
    int i = 1;
    for(; i < length; ++i)
    {
        int ch = r.read();
        if(ch == -1 || isWhitespace(ch) || isMacro(ch))
        {
            unread(r, ch);
            break;
        }
        int d = Character.digit(ch, base);
        if(d == -1)
            throw new IllegalArgumentException(
                "Invalid digit: " + (char) ch);
        uc = uc * base + d;
    }
    if(i != length && exact)
        throw new IllegalArgumentException(
            "Invalid character length: " + i + ", should be: " + length);
    return uc;
}

static private Object interpretToken(String s) throws Exception{
    if(s.equals("nil"))
    {
        return null;
    }
}

```

```

        else if(s.equals("true"))
        {
            return RT.T;
        }
        else if(s.equals("false"))
        {
            return RT.F;
        }
        else if(s.equals("/"))
        {
            return SLASH;
        }
        else if(s.equals("clojure.core//"))
        {
            return CLOJURE_SLASH;
        }
    Object ret = null;

    ret = matchSymbol(s);
    if(ret != null)
        return ret;

    throw new Exception("Invalid token: " + s);
}

private static Object matchSymbol(String s){
    Matcher m = symbolPat.matcher(s);
    if(m.matches())
    {
        int gc = m.groupCount();
        String ns = m.group(1);
        String name = m.group(2);
        if(ns != null && ns.endsWith(":/")
           || name.endsWith("::")
           || s.indexOf(":::", 1) != -1)
            return null;
        if(s.startsWith(":::"))
        {
            Symbol ks = Symbol.intern(s.substring(2));
            Namespace kns;
            if(ks.ns != null)
                kns = Compiler.namespaceFor(ks);
            else
                kns = Compiler.currentNS();
            //auto-resolving keyword
            if (kns != null)
                return Keyword.intern(kns.name.name, ks.name);
            else
                return null;
        }
    }
}

```

```

        }
        boolean isKeyword = s.charAt(0) == ':';
        Symbol sym = Symbol.intern(s.substring(isKeyword ? 1 : 0));
        if(isKeyword)
            return Keyword.intern(sym);
        return sym;
    }
    return null;
}

private static Object matchNumber(String s){
    Matcher m = intPat.matcher(s);
    if(m.matches())
    {
        if(m.group(2) != null)
        {
            if(m.group(8) != null)
                return BigInt.ZERO;
            return Numbers.num(0);
        }
        boolean negate = (m.group(1).equals("-"));
        String n;
        int radix = 10;
        if((n = m.group(3)) != null)
            radix = 10;
        else if((n = m.group(4)) != null)
            radix = 16;
        else if((n = m.group(5)) != null)
            radix = 8;
        else if((n = m.group(7)) != null)
            radix = Integer.parseInt(m.group(6));
        if(n == null)
            return null;
        BigInteger bn = new BigInteger(n, radix);
        if(negate)
            bn = bn.negate();
        if(m.group(8) != null)
            return BigInt.fromBigInteger(bn);
        return bn.bitLength() < 64 ?
            Numbers.num(bn.longValue())
            : BigInt.fromBigInteger(bn);
    }
    m = floatPat.matcher(s);
    if(m.matches())
    {
        if(m.group(4) != null)
            return new BigDecimal(m.group(1));
        return Double.parseDouble(s);
    }
}

```

```

m = ratioPat.matcher(s);
if(m.matches())
{
    return
        Numbers.divide(
            Numbers.reduceBigInt(
                BigInt.fromBigInteger(new BigInteger(m.group(1)))),
            Numbers.reduceBigInt(
                BigInt.fromBigInteger(new BigInteger(m.group(2)))));
}
return null;
}

static private IFn getMacro(int ch){
    if(ch < macros.length)
        return macros[ch];
    return null;
}

static private boolean isMacro(int ch){
    return (ch < macros.length && macros[ch] != null);
}

static private boolean isTerminatingMacro(int ch){
    return (ch != '#' && ch != '\'' && isMacro(ch));
}

\getchunk{LispReader RegexReader class}

\getchunk{LispReader StringReader class}

\getchunk{LispReader CommentReader class}

\getchunk{LispReader DiscardReader class}

\getchunk{LispReader WrappingReader class}

public static class DeprecatedWrappingReader extends AFn{
    final Symbol sym;
    final String macro;

    public DeprecatedWrappingReader(Symbol sym, String macro){
        this.sym = sym;
        this.macro = macro;
    }

    public Object invoke(Object reader, Object quote)
        throws Exception{
        System.out.println("WARNING: reader macro " + macro +
            " is deprecated; use " + sym.getName() +

```

```

        " instead");
    PushbackReader r = (PushbackReader) reader;
    Object o = read(r, true, null, true);
    return RT.list(sym, o);
}

\getchunk{LispReader VarReader class}

/*
static class DerefReader extends AFn{

    public Object invoke(Object reader, Object quote)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        int ch = r.read();
        if(ch == -1)
            throw new Exception("EOF while reading character");
        if(ch == '!')
        {
            Object o = read(r, true, null, true);
            return RT.list(DEREF_BANG, o);
        }
        else
        {
            r.unread(ch);
            Object o = read(r, true, null, true);
            return RT.list(DEREF, o);
        }
    }

}
*/
\getchunk{LispReader DispatchReader class}

static Symbol garg(int n){
    return Symbol.intern(null,
        (n == -1 ? "rest" : ("p" + n)) + "__" + RT.nextID() + "#");
}

\getchunk{LispReader FnReader class}

static Symbol registerArg(int n){
    PersistentTreeMap argsyms = (PersistentTreeMap) ARG_ENV.deref();
    if(argsyms == null)
    {
        throw new IllegalStateException("arg literal not in #()");
    }
}
```

```

Symbol ret = (Symbol) argsyms.valAt(n);
if(ret == null)
{
    ret = garg(n);
    ARG_ENV.set(argsyms.assoc(n, ret));
}
return ret;
}

\getchunk{LispReader ArgReader class}

\getchunk{LispReader MetaReader class}

\getchunk{LispReader SyntaxQuoteReader class}

static boolean isUnquoteSplicing(Object form){
    return form instanceof ISeq &&
           Util.equals(RT.first(form),UNQUOTE_SPLICING);
}

static boolean isUnquote(Object form){
    return form instanceof ISeq && Util.equals(RT.first(form),UNQUOTE);
}

\getchunk{LispReader UnquoteReader class}

\getchunk{LispReader CharacterReader class}

\getchunk{LispReader ListReader class}

static class CtorReader extends AFn{
    static final Symbol cls = Symbol.intern("class");

    public Object invoke(Object reader, Object leftangle)
        throws Exception{
        PushbackReader r = (PushbackReader) reader;
        // #<classname>
        // #<classname args*>
        // #<classname/staticMethod args*>
        List list = readDelimitedList('>', r, true);
        if(list.isEmpty())
            throw new Exception(
                "Must supply 'class', classname or classname/staticMethod");
        Symbol s = (Symbol) list.get(0);
        Object[] args = list.subList(1, list.size()).toArray();
        if(s.equals(cls))
        {
            return RT.className(args[0].toString());
        }
    else if(s.ns != null) //static method

```

```

    {
        String classname = s.ns;
        String method = s.name;
        return Reflector.invokeStaticMethod(classname, method, args);
    }
    else
    {
        return
            Reflector.invokeConstructor(RT.classForName(s.name), args);
    }
}

\getchunk{LispReader EvalReader class}

//static class ArgVectorReader extends AFn{
//    public Object invoke(Object reader, Object leftparen)
//        throws Exception{
//        PushbackReader r = (PushbackReader) reader;
//        return ArgVector.create(readDelimitedList('|', r, true));
//    }
//}

\getchunk{LispReader VectorReader class}

\getchunk{LispReader MapReader class}

\getchunk{LispReader SetReader class}

\getchunk{LispReader UnmatchedDelimiterReader class}

\getchunk{LispReader UnreadableReader class}

public static List readDelimitedList(char delim,
                                      PushbackReader r,
                                      boolean isRecursive)
throws Exception
final int firstline =
(r instanceof LineNumberingPushbackReader) ?
((LineNumberingPushbackReader) r).getLineNumber() : -1;

ArrayList a = new ArrayList();

for(; ;)
{
    int ch = r.read();

    while(isWhitespace(ch))

```

```

        ch = r.read();

        if(ch == -1)
        {
            if(firstline < 0)
                throw new Exception("EOF while reading");
            else
                throw new Exception(
                    "EOF while reading, starting at line " + firstline);
        }

        if(ch == delim)
            break;

        IFn macroFn = getMacro(ch);
        if(macroFn != null)
        {
            Object mret = macroFn.invoke(r, (char) ch);
            //no op macros return the reader
            if(mret != r)
                a.add(mret);
        }
        else
        {
            unread(r, ch);

            Object o = read(r, true, null, isRecursive);
            if(o != r)
                a.add(o);
        }
    }

    return a;
}

/*
public static void main(String[] args)
throws Exception{
    //RT.init();
    PushbackReader rdr =
        new PushbackReader( new java.io.StringReader( "(+ 21 21)" ) );
    Object input = LispReader.read(rdr, false, new Object(), false );
    System.out.println(Compiler.eval(input));
}

public static void main(String[] args){
    LineNumberingPushbackReader r =
        new LineNumberingPushbackReader(
            new InputStreamReader(System.in));
}

```

```

OutputStreamWriter w = new OutputStreamWriter(System.out);
Object ret = null;
try
{
    for(; ;)
    {
        ret = LispReader.read(r, true, null, false);
        RT.print(ret, w);
        w.write('\n');
        if(ret != null)
            w.write(ret.getClass().toString());
        w.write('\n');
        w.flush();
    }
}
catch(Exception e)
{
    e.printStackTrace();
}
*/
}

```

9.69 LockingTransaction.java

— LockingTransaction.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 26, 2007 */

package clojure.lang;

import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicLong;
import java.util.concurrent.Callable;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.CountDownLatch;

@SuppressWarnings({"SynchronizeOnNonFinalField"})
public class LockingTransaction{

```

```
public static final int RETRY_LIMIT = 10000;
public static final int LOCK_WAIT_MSECS = 100;
public static final long BARGE_WAIT_NANOS = 10 * 1000000;
//public static int COMMUTE_RETRY_LIMIT = 10;

static final int RUNNING = 0;
static final int COMMITTING = 1;
static final int RETRY = 2;
static final int KILLED = 3;
static final int COMMITTED = 4;

final static ThreadLocal<LockingTransaction> transaction =
    new ThreadLocal<LockingTransaction>();

static class RetryEx extends Error{
}

static class AbortException extends Exception{
}

public static class Info{
    final AtomicInteger status;
    final long startPoint;
    final CountDownLatch latch;

    public Info(int status, long startPoint){
        this.status = new AtomicInteger(status);
        this.startPoint = startPoint;
        this.latch = new CountDownLatch(1);
    }

    public boolean running(){
        int s = status.get();
        return s == RUNNING || s == COMMITTING;
    }
}

static class CFn{
    final IFn fn;
    final ISeq args;

    public CFn(IFn fn, ISeq args){
        this.fn = fn;
        this.args = args;
    }
}
//total order on transactions
```

```

//transactions will consume a point for init, for each retry,
//and on commit if writing
final private static AtomicLong lastPoint = new AtomicLong();

void getReadPoint(){
    readPoint = lastPoint.incrementAndGet();
}

long getCommitPoint(){
    return lastPoint.incrementAndGet();
}

void stop(int status){
    if(info != null)
    {
        synchronized(info)
        {
            info.status.set(status);
            info.latch.countDown();
        }
        info = null;
        vals.clear();
        sets.clear();
        commutes.clear();
        //actions.clear();
    }
}
}

Info info;
long readPoint;
long startPoint;
long startTime;
final RetryEx retryex = new RetryEx();
final ArrayList<Agent.Action> actions = new ArrayList<Agent.Action>();
final HashMap<Ref, Object> vals = new HashMap<Ref, Object>();
final HashSet<Ref> sets = new HashSet<Ref>();
final TreeMap<Ref, ArrayList<CFn>> commutes =
    new TreeMap<Ref, ArrayList<CFn>>();

final HashSet<Ref> ensures = new HashSet<Ref>();    //all hold readLock

void tryWriteLock(Ref ref){
    try
    {
        if(!ref.lock
            .writeLock()
            .tryLock(LOCK_WAIT_MSECS, TimeUnit.MILLISECONDS))
        throw retryex;
    }
}

```

```

        }
    catch(InterruptedException e)
    {
        throw retryex;
    }
}

//returns the most recent val
Object lock(Ref ref){
    //can't upgrade readLock, so release it
    releaseIfEnsured(ref);

    boolean unlocked = true;
    try
    {
        tryWriteLock(ref);
        unlocked = false;

        if(ref.tvals != null && ref.tvals.point > readPoint)
            throw retryex;
        Info refinfo = ref.tinfo;

        //write lock conflict
        if(refinfo != null && refinfo != info && refinfo.running())
        {
            if(!barge(refinfo))
            {
                ref.lock.writeLock().unlock();
                unlocked = true;
                return blockAndBail(refinfo);
            }
        }
        ref.tinfo = info;
        return ref.tvals == null ? null : ref.tvals.val;
    }
    finally
    {
        if(!unlocked)
            ref.lock.writeLock().unlock();
    }
}

private Object blockAndBail(Info refinfo){
//stop prior to blocking
    stop(RETRY);
    try
    {
        refinfo.latch.await(LOCK_WAIT_MSECS, TimeUnit.MILLISECONDS);
    }
    catch(InterruptedException e)
}

```

```

    {
        //ignore
    }
    throw retryex;
}

private void releaseIfEnsured(Ref ref){
    if(ensures.contains(ref))
    {
        ensures.remove(ref);
        ref.lock.readLock().unlock();
    }
}

void abort() throws AbortException{
    stop(KILLED);
    throw new AbortException();
}

private boolean bargeTimeElapsed(){
    return System.nanoTime() - startTime > BARGE_WAIT_NANOS;
}

private boolean barge(Info refinfo){
    boolean barged = false;
    //if this transaction is older
    // try to abort the other
    if(bargeTimeElapsed() && startPoint < refinfo.startPoint)
    {
        barged = refinfo.status.compareAndSet(RUNNING, KILLED);
        if(barged)
            refinfo.latch.countDown();
    }
    return barged;
}

static LockingTransaction getEx(){
    LockingTransaction t = transaction.get();
    if(t == null || t.info == null)
        throw new IllegalStateException("No transaction running");
    return t;
}

static public boolean isRunning(){
    return getRunning() != null;
}

static LockingTransaction getRunning(){
    LockingTransaction t = transaction.get();
    if(t == null || t.info == null)

```

```
        return null;
        return t;
    }

    static public Object runInTransaction(Callable fn) throws Exception{
        LockingTransaction t = transaction.get();
        if(t == null)
            transaction.set(t = new LockingTransaction());

        if(t.info != null)
            return fn.call();

        return t.run(fn);
    }

    static class Notify{
        final public Ref ref;
        final public Object oldval;
        final public Object newval;

        Notify(Ref ref, Object oldval, Object newval){
            this.ref = ref;
            this.oldval = oldval;
            this.newval = newval;
        }
    }

    Object run(Callable fn) throws Exception{
        boolean done = false;
        Object ret = null;
        ArrayList<Ref> locked = new ArrayList<Ref>();
        ArrayList<Notify> notify = new ArrayList<Notify>();

        for(int i = 0; !done && i < RETRY_LIMIT; i++)
        {
            try
            {
                getReadPoint();
                if(i == 0)
                {
                    startPoint = readPoint;
                    startTime = System.nanoTime();
                }
                info = new Info(RUNNING, startPoint);
                ret = fn.call();
                //make sure no one has killed us before this point,
                //and can't from now on
                if(info.status.compareAndSet(RUNNING, COMMITTING))
                {
                    for(Map.Entry<Ref, ArrayList<CFn>> e
```

```

        : commutes.entrySet())
    {
        Ref ref = e.getKey();
        if(sets.contains(ref)) continue;

        boolean wasEnsured = ensures.contains(ref);
        //can't upgrade readLock, so release it
        releaseIfEnsured(ref);
        tryWriteLock(ref);
        locked.add(ref);
        if(wasEnsured && ref.tvals != null &&
           ref.tvals.point > readPoint)
            throw retryex;

        Info refinfo = ref.tinfo;
        if(refinfo != null &&
           refinfo != info &&
           refinfo.running())
        {
            if(!barge(refinfo))
                throw retryex;
        }
        Object val =
            ref.tvals == null ? null : ref.tvals.val;
        vals.put(ref, val);
        for(CFn f : e.getValue())
        {
            vals.put(ref,
                      f.fn.applyTo(
                          RT.cons(vals.get(ref), f.args)));
        }
    }
    for(Ref ref : sets)
    {
        tryWriteLock(ref);
        locked.add(ref);
    }

    //validate and enqueue notifications
    for(Map.Entry<Ref, Object> e : vals.entrySet())
    {
        Ref ref = e.getKey();
        ref.validate(ref.getValidator(), e.getValue());
    }

    //at this point, all values calced, all refs to
    //be written locked
    //no more client code to be called
    long msecs = System.currentTimeMillis();
    long commitPoint = getCommitPoint();
}

```

```

        for(Map.Entry<Ref, Object> e : vals.entrySet())
        {
            Ref ref = e.getKey();
            Object oldval =
                ref.tvals == null ? null : ref.tvals.val;
            Object newval = e.getValue();
            int hcount = ref.histCount();

            if(ref.tvals == null)
            {
                ref.tvals =
                    new Ref.TVal(newval, commitPoint, msecs);
            }
            else if((ref.faults.get() > 0 &&
                     hcount < ref.maxHistory) ||
                     hcount < ref.minHistory)
            {
                ref.tvals =
                    new Ref.TVal(newval,
                                commitPoint,
                                msecs,
                                ref.tvals);
                ref.faults.set(0);
            }
            else
            {
                ref.tvals = ref.tvals.next;
                ref.tvals.val = newval;
                ref.tvals.point = commitPoint;
                ref.tvals.msecs = msecs;
            }
            if(ref.getWatches().count() > 0)
                notify.add(new Notify(ref, oldval, newval));
        }

        done = true;
        info.status.set(COMMITTED);
    }
}
catch(RetryEx retry)
{
    //eat this so we retry rather than fall out
}
finally
{
    for(int k = locked.size() - 1; k >= 0; --k)
    {
        locked.get(k).lock.writeLock().unlock();
    }
    locked.clear();
}

```

```

        for(Ref r : ensures)
        {
            r.lock.readLock().unlock();
        }
    ensures.clear();
    stop(done ? COMMITTED : RETRY);
    try
    {
        if(done) //re-dispatch out of transaction
        {
            for(Notify n : notify)
            {
                n.ref.notifyWatches(n.oldval, n.newval);
            }
            for(Agent.Action action : actions)
            {
                Agent.dispatchAction(action);
            }
        }
    }
    finally
    {
        notify.clear();
        actions.clear();
    }
}
if(!done)
    throw new Exception(
        "Transaction failed after reaching retry limit");
return ret;
}

public void enqueue(Agent.Action action){
    actions.add(action);
}

Object doGet(Ref ref){
    if(!info.running())
        throw retryex;
    if(vals.containsKey(ref))
        return vals.get(ref);
    try
    {
        ref.lock.readLock().lock();
        if(ref.tvals == null)
            throw new IllegalStateException(
                ref.toString() + " is unbound.");
        Ref.TVal ver = ref.tvals;
        do

```

```

        {
            if(ver.point <= readPoint)
                return ver.val;
            } while((ver = ver.prior) != ref.tvals);
        }
    finally
    {
        ref.lock.readLock().unlock();
    }
    //no version of val precedes the read point
    ref.faults.incrementAndGet();
    throw retryex;
}

Object doSet(Ref ref, Object val){
    if(!info.running())
        throw retryex;
    if(commutes.containsKey(ref))
        throw new IllegalStateException("Can't set after commute");
    if(!sets.contains(ref))
    {
        sets.add(ref);
        lock(ref);
    }
    vals.put(ref, val);
    return val;
}

void doEnsure(Ref ref){
    if(!info.running())
        throw retryex;
    if(ensures.contains(ref))
        return;
    ref.lock.readLock().lock();

    //someone completed a write after our snapshot
    if(ref.tvals != null && ref.tvals.point > readPoint) {
        ref.lock.readLock().unlock();
        throw retryex;
    }

    Info refinfo = ref.tinfo;

    //writer exists
    if(refinfo != null && refinfo.running())
    {
        ref.lock.readLock().unlock();

        if(refinfo != info) //not us, ensure is doomed

```

```

        {
            blockAndBail(refinfo);
        }
    }
    else
        ensures.add(ref);
}

Object doCommute(Ref ref, IFn fn, ISeq args) throws Exception{
    if(!info.running())
        throw retryex;
    if(!vals.containsKey(ref))
    {
        Object val = null;
        try
        {
            ref.lock.readLock().lock();
            val = ref.tvals == null ? null : ref.tvals.val;
        }
        finally
        {
            ref.lock.readLock().unlock();
        }
        vals.put(ref, val);
    }
    ArrayList<CFn> fns = commutes.get(ref);
    if(fns == null)
        commutes.put(ref, fns = new ArrayList<CFn>());
    fns.add(new CFn(fn, args));
    Object ret = fn.applyTo(RT.cons(vals.get(ref), args));
    vals.put(ref, ret);
    return ret;
}

/*
//for test
static CyclicBarrier barrier;
static ArrayList<Ref> items;

public static void main(String[] args){
    try
    {
        if(args.length != 4)
            System.err.println(
                "Usage: LockingTransaction nthreads nitems niters ninstances");
        int nthreads = Integer.parseInt(args[0]);
        int nitems = Integer.parseInt(args[1]);
        int niters = Integer.parseInt(args[2]);
        int ninstances = Integer.parseInt(args[3]);
    }
}

```

```
if(items == null)
{
    ArrayList<Ref> temp = new ArrayList(nitems);
    for(int i = 0; i < nitems; i++)
        temp.add(new Ref(0));
    items = temp;
}

class Incr extends AFn{
    public Object invoke(Object arg1) throws Exception{
        Integer i = (Integer) arg1;
        return i + 1;
    }

    public Obj withMeta(IPersistentMap meta){
        throw new UnsupportedOperationException();
    }
}

class Commuter extends AFn implements Callable{
    int niters;
    List<Ref> items;
    Incr incr;

    public Commuter(int niters, List<Ref> items){
        this.niters = niters;
        this.items = items;
        this.incr = new Incr();
    }

    public Object call() throws Exception{
        long nanos = 0;
        for(int i = 0; i < niters; i++)
        {
            long start = System.nanoTime();
            LockingTransaction.runInTransaction(this);
            nanos += System.nanoTime() - start;
        }
        return nanos;
    }

    public Object invoke() throws Exception{
        for(Ref tref : items)
        {
            LockingTransaction.getEx().doCommute(tref, incr);
        }
        return null;
    }
}
```

```

public Obj withMeta(IPersistentMap meta){
    throw new UnsupportedOperationException();

}

class Incrementer extends AFn implements Callable{
    int niters;
    List<Ref> items;

    public Incrementer(int niters, List<Ref> items){
        this.niters = niters;
        this.items = items;
    }

    public Object call() throws Exception{
        long nanos = 0;
        for(int i = 0; i < niters; i++)
        {
            long start = System.nanoTime();
            LockingTransaction.runInTransaction(this);
            nanos += System.nanoTime() - start;
        }
        return nanos;
    }

    public Object invoke() throws Exception{
        for(Ref tref : items)
        {
//            Transaction.get().doTouch(tref);
//            LockingTransaction t = LockingTransaction.getEx();
//            int val = (Integer) t.doGet(tref);
//            t.doSet(tref, val + 1);
//            int val = (Integer) tref.get();
//            tref.set(val + 1);
        }
        return null;
    }

    public Obj withMeta(IPersistentMap meta){
        throw new UnsupportedOperationException();

    }
}

ArrayList<Callable<Long>> tasks = new ArrayList(nthreads);
for(int i = 0; i < nthreads; i++)
{

```

```
ArrayList<Ref> si;
synchronized(items)
{
    si = (ArrayList<Ref>) items.clone();
}
Collections.shuffle(si);
tasks.add(new Incrementer(niters, si));
//tasks.add(new Commuter(niters, si));
}
ExecutorService e = Executors.newFixedThreadPool(nthreads);

if(barrier == null)
    barrier = new CyclicBarrier(ninstances);
System.out.println("waiting for other instances...");
barrier.await();
System.out.println("starting");
long start = System.nanoTime();
List<Future<Long>> results = e.invokeAll(tasks);
long estimatedTime = System.nanoTime() - start;
System.out.printf(
    "nthreads: %d, nitems: %d, niters: %d, time: %d%n",
    nthreads, nitems, niters, estimatedTime / 1000000);
e.shutdown();
for(Future<Long> result : results)
{
    System.out.printf("%d, ", result.get() / 1000000);
}
System.out.println();
System.out.println("waiting for other instances...");
barrier.await();
synchronized(items)
{
    for(Ref item : items)
    {
        System.out.printf("%d, ", (Integer) item.currentVal());
    }
}
System.out.println("\ndone");
System.out.flush();
}
catch(Exception ex)
{
    ex.printStackTrace();
}
}

*/
}
```

9.70 MapEntry.java

```
(AMapEntry [527])
— MapEntry.java —

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.util.Iterator;

public class MapEntry extends AMapEntry{
final Object _key;
final Object _val;

public MapEntry(Object key, Object val){
    this._key = key;
    this._val = val;
}

public Object key(){
    return _key;
}

public Object val(){
    return _val;
}

public Object getKey(){
    return key();
}

public Object getValue(){
    return val();
}
}
```

9.71 MapEquivalence.java

— MapEquivalence.java —

```
/*
```

```
\getchunk{Clojure Copyright}
*/
/* rich Aug 4, 2010 */

package clojure.lang;

//marker interface
public interface MapEquivalence{
}
```

9.72 MethodImplCache.java

— MethodImplCache.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Nov 8, 2009 */

package clojure.lang;

public final class MethodImplCache{

    static public class Entry{
        final public Class c;
        final public IFn fn;

        public Entry(Class c, IFn fn){
            this.c = c;
            this.fn = fn;
        }
    }

    public final IPersistentMap protocol;
    public final Keyword methodk;
    public final int shift;
    public final int mask;
    public final Object[] table;    //[[class, entry, class, entry ...]]

    Entry mre = null;

    public MethodImplCache(IPersistentMap protocol, Keyword methodk){
        this(protocol, methodk, 0, 0, RT.EMPTY_ARRAY);
    }
```

```

public MethodImplCache(IPersistentMap protocol,
                      Keyword methodk,
                      int shift,
                      int mask,
                      Object[] table){
    this.protocol = protocol;
    this.methodk = methodk;
    this.shift = shift;
    this.mask = mask;
    this.table = table;
}

public IFn fnFor(Class c){
    Entry last = mre;
    if(last != null && last.c == c)
        return last.fn;
    return findFnFor(c);
}

IFn findFnFor(Class c){
    int idx = ((Util.hash(c) >> shift) & mask) << 1;
    if(idx < table.length && table[idx] == c)
    {
        Entry e = ((Entry) table[idx + 1]);
        mre = e;
        return e != null ? e.fn : null;
    }
    return null;
}
}

```

9.73 MultiFn.java

(AFn [509])
 — MultiFn.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Sep 13, 2007 */

package clojure.lang;

import java.util.Map;

```

```
public class MultiFn extends AFn{
    final public IFn dispatchFn;
    final public Object defaultDispatchVal;
    final public IRef hierarchy;
    final String name;
    IPersistentMap methodTable;
    IPersistentMap preferTable;
    IPersistentMap methodCache;
    Object cachedHierarchy;

    static final Var assoc = RT.var("clojure.core", "assoc");
    static final Var dissoc = RT.var("clojure.core", "dissoc");
    static final Var isa = RT.var("clojure.core", "isa?");
    static final Var parents = RT.var("clojure.core", "parents");

    public MultiFn(String name,
                    IFn dispatchFn,
                    Object defaultDispatchVal,
                    IRef hierarchy) throws Exception{
        this.name = name;
        this.dispatchFn = dispatchFn;
        this.defaultDispatchVal = defaultDispatchVal;
        this.methodTable = PersistentHashMap.EMPTY;
        this.methodCache = getMethodTable();
        this.preferTable = PersistentHashMap.EMPTY;
        this.hierarchy = hierarchy;
        cachedHierarchy = null;
    }

    synchronized public MultiFn reset(){
        methodTable = methodCache
            = preferTable
            = PersistentHashMap.EMPTY;
        cachedHierarchy = null;
        return this;
    }

    synchronized public MultiFn addMethod(Object dispatchVal,
                                          IFn method)
    throws Exception{
        methodTable = getMethodTable().assoc(dispatchVal, method);
        resetCache();
        return this;
    }

    synchronized public MultiFn removeMethod(Object dispatchVal)
    throws Exception{
        methodTable = getMethodTable().without(dispatchVal);
        resetCache();
    }
}
```

```

        return this;
    }

    synchronized public MultiFn preferMethod(Object dispatchValX,
                                              Object dispatchValY)
    throws Exception{
        if(prefers(dispatchValY, dispatchValX))
            throw new IllegalStateException(
                String.format("Preference conflict in multimethod '%s':"+
                    "%s is already preferred to %s",
                    name, dispatchValY, dispatchValX));
        preferTable =
            getPreferTable().assoc(dispatchValX,
                RT.conj((IPersistentCollection)
                    RT.get(getPreferTable(),
                        dispatchValX,
                        PersistentHashSet.EMPTY),
                        dispatchValY));
        resetCache();
        return this;
    }

    private boolean prefers(Object x, Object y) throws Exception{
        IPersistentSet xprefs =
            (IPersistentSet) getPreferTable().valAt(x);
        if(xprefs != null && xprefs.contains(y))
            return true;
        for(ISeq ps = RT.seq(parents.invoke(y));
            ps != null;
            ps = ps.next())
        {
            if(prefers(x, ps.first()))
                return true;
        }
        for(ISeq ps = RT.seq(parents.invoke(x));
            ps != null;
            ps = ps.next())
        {
            if(prefers(ps.first(), y))
                return true;
        }
        return false;
    }

    private boolean isA(Object x, Object y) throws Exception{
        return RT.booleanCast(isa.invoke(hierarchy.deref(), x, y));
    }

    private boolean dominates(Object x, Object y) throws Exception{
        return prefers(x, y) || isA(x, y);
    }
}

```

```

}

private IPersistentMap resetCache() throws Exception{
    methodCache = getMethodTable();
    cachedHierarchy = hierarchy.deref();
    return methodCache;
}

synchronized public IFn getMethod(Object dispatchVal) throws Exception{
    if(cachedHierarchy != hierarchy.deref())
        resetCache();
    IFn targetFn = (IFn) methodCache.getValueAt(dispatchVal);
    if(targetFn != null)
        return targetFn;
    targetFn = findAndCacheBestMethod(dispatchVal);
    if(targetFn != null)
        return targetFn;
    targetFn = (IFn) getMethodTable().valAt(defaultDispatchVal);
    return targetFn;
}

private IFn getFn(Object dispatchVal) throws Exception{
    IFn targetFn = getMethod(dispatchVal);
    if(targetFn == null)
        throw new IllegalArgumentException(
            String.format("No method in multimethod '%s' "+
                "for dispatch value: %s",
                name, dispatchVal));
    return targetFn;
}

private IFn findAndCacheBestMethod(Object dispatchVal)
throws Exception{
    Map.Entry bestEntry = null;
    for(Object o : getMethodTable())
    {
        Map.Entry e = (Map.Entry) o;
        if(isA(dispatchVal, e.getKey()))
        {
            if(bestEntry == null ||
                dominates(e.getKey(), bestEntry.getKey()))
                bestEntry = e;
            if(!dominates(bestEntry.getKey(), e.getKey()))
                throw new IllegalArgumentException(
                    String.format(
                        "Multiple methods in multimethod '%s' match "+
                        "dispatch value: %s -> %s and %s, and "+
                        "neither is preferred",
                        name, dispatchVal, e.getKey(), bestEntry.getKey()));
        }
    }
}

```

```

        }
        if(bestEntry == null)
            return null;
        //ensure basis has stayed stable throughout, else redo
        if(cachedHierarchy == hierarchy.deref())
        {
            //place in cache
            methodCache =
                methodCache.assoc(dispatchVal, bestEntry.getValue());
            return (IFn) bestEntry.getValue();
        }
    else
    {
        resetCache();
        return findAndCacheBestMethod(dispatchVal);
    }
}

public Object invoke()
throws Exception{
    return getFn(dispatchFn.invoke()).invoke();
}

public Object invoke(Object arg1)
throws Exception{
    return getFn(dispatchFn.invoke(arg1)).invoke(arg1);
}

public Object invoke(Object arg1, Object arg2)
throws Exception{
    return getFn(dispatchFn.invoke(arg1, arg2)).invoke(arg1, arg2);
}

public Object invoke(Object arg1, Object arg2, Object arg3)
throws Exception{
    return getFn(dispatchFn.invoke(arg1, arg2, arg3))
        .invoke(arg1, arg2, arg3);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4)
throws Exception{
    return getFn(dispatchFn.invoke(arg1, arg2, arg3, arg4))
        .invoke(arg1, arg2, arg3, arg4);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4, Object arg5)
throws Exception{
    return getFn(dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5));
}

```

```
        .invoke(arg1, arg2, arg3, arg4, arg5);
    }

    public Object invoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6)
    throws Exception{
        return getFn(
            dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6))
        .invoke(arg1, arg2, arg3, arg4, arg5, arg6);
    }

    public Object invoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7)
    throws Exception{
        return getFn(
            dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7))
        .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7);
    }

    public Object invoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8)
    throws Exception{
        return getFn(
            dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8))
        .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8);
    }

    public Object invoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9)
    throws Exception{
        return getFn(
            dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                             arg9))
        .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9);
    }

    public Object invoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9,
                         Object arg10)
    throws Exception{
        return getFn(
            dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                             arg9, arg10))
        .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
                 arg10);
    }
}
```

```

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12, arg13))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12, arg13);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12, arg13, arg14))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12, arg13, arg14);
}

```

```
public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12, arg13, arg14, arg15))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12, arg13, arg14, arg15);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12, arg13, arg14, arg15,
                          arg16))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12, arg13, arg14, arg15, arg16);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12, arg13, arg14, arg15,
                          arg16, arg17))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12, arg13, arg14, arg15, arg16, arg17);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18)
```

```

throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12, arg13, arg14, arg15,
                          arg16, arg17, arg18))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12, arg13, arg14, arg15, arg16, arg17,
            arg18);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12, arg13, arg14, arg15,
                          arg16, arg17, arg18, arg19))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12, arg13, arg14, arg15, arg16,
            arg17, arg18, arg19);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12, arg13, arg14, arg15,
                          arg16, arg17, arg18, arg19, arg20))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12, arg13, arg14, arg15, arg16,
            arg17, arg18, arg19, arg20);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19,
                     Object arg20)

```

```

        Object arg19, Object arg20, Object... args)
throws Exception{
    return getFn(
        dispatchFn.invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
                          arg9, arg10, arg11, arg12, arg13, arg14, arg15,
                          arg16, arg17, arg18, arg19, arg20, args))
    .invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
            arg10, arg11, arg12, arg13, arg14, arg15, arg16,
            arg17, arg18, arg19, arg20, args);
}

public IPersistentMap getMethodTable() {
    return methodTable;
}

public IPersistentMap getPreferTable() {
    return preferTable;
}
}

```

—————

9.74 Named.java

— Named.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Sep 20, 2007 */

package clojure.lang;

public interface Named{
String getNamespace();

String getName();
}

```

—————

9.75 Namespace.java

(AReference [552]) (Serializable [1723])
 — Namespace.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Jan 23, 2008 */

package clojure.lang;

import java.io.ObjectStreamException;
import java.io.Serializable;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicReference;

public class Namespace extends AReference implements Serializable {
    final public Symbol name;
    transient final AtomicReference<IPersistentMap> mappings =
        new AtomicReference<IPersistentMap>();
    transient final AtomicReference<IPersistentMap> aliases =
        new AtomicReference<IPersistentMap>();

    final static ConcurrentHashMap<Symbol, Namespace> namespaces =
        new ConcurrentHashMap<Symbol, Namespace>();

    public String toString(){
        return name.toString();
    }

    Namespace(Symbol name){
        super(name.meta());
        this.name = name;
        mappings.set(RT.DEFAULT_IMPORTS);
        aliases.set(RT.map());
    }

    public static ISeq all(){
        return RT.seq(namespaces.values());
    }

    public Symbol getName(){
        return name;
    }

    public IPersistentMap getMappings(){
        return mappings.get();
    }

    public Var intern(Symbol sym){
        if(sym.ns != null)
            {
                throw new IllegalArgumentException(
                    "Can't intern namespace-qualified symbol");
            }
        
```

```

        }
        IPersistentMap map = getMappings();
        Object o;
        Var v = null;
        while((o = map.valAt(sym)) == null)
        {
            if(v == null)
                v = new Var(this, sym);
            IPersistentMap newMap = map.assoc(sym, v);
            mappings.compareAndSet(map, newMap);
            map = getMappings();
        }
        if(o instanceof Var && ((Var) o).ns == this)
            return (Var) o;

        if(v == null)
            v = new Var(this, sym);

        warnOrFailOnReplace(sym, o, v);

        while(!mappings.compareAndSet(map, map.assoc(sym, v)))
            map = getMappings();

        return v;
    }

    private void warnOrFailOnReplace(Symbol sym, Object o, Object v){
        if (o instanceof Var)
        {
            Namespace ns = ((Var)o).ns;
            if (ns == this)
                return;
            if (ns != RT.CLOJURE_NS)
                throw new IllegalStateException(
                    sym + " already refers to: " + o +
                    " in namespace: " + name);
        }
        RT.errPrintWriter().println("WARNING: " + sym +
            " already refers to: " + o + " in namespace: " + name
            + ", being replaced by: " + v);
    }

    Object reference(Symbol sym, Object val){
        if(sym.ns != null)
        {
            throw new IllegalArgumentException(
                "Can't intern namespace-qualified symbol");
        }
        IPersistentMap map = getMappings();

```

```

Object o;
while((o = map.valAt(sym)) == null)
{
    IPersistentMap newMap = map.assoc(sym, val);
    mappings.compareAndSet(map, newMap);
    map = getMappings();
}
if(o == val)
    return o;

warnOrFailOnReplace(sym, o, val);

while(!mappings.compareAndSet(map, map.assoc(sym, val)))
    map = getMappings();

return val;
}

public static boolean
areDifferentInstancesOfSameClassName(Class cls1,
                                     Class cls2) {
    return (cls1 != cls2) && (cls1.getName().equals(cls2.getName()));
}

Class referenceClass(Symbol sym, Class val){
    if(sym.ns != null)
    {
        throw new IllegalArgumentException(
            "Can't intern namespace-qualified symbol");
    }
    IPersistentMap map = getMappings();
    Class c = (Class) map.valAt(sym);
    while((c == null) ||
          (areDifferentInstancesOfSameClassName(c, val)))
    {
        IPersistentMap newMap = map.assoc(sym, val);
        mappings.compareAndSet(map, newMap);
        map = getMappings();
        c = (Class) map.valAt(sym);
    }
    if(c == val)
        return c;

    throw new IllegalStateException(sym + " already refers to: " +
                                   c + " in namespace: " + name);
}

public void unmap(Symbol sym) throws Exception{
    if(sym.ns != null)

```

```
{  
    throw new IllegalArgumentException(  
        "Can't unintern namespace-qualified symbol");  
}  
IPersistentMap map = getMappings();  
while(map.containsKey(sym))  
{  
    IPersistentMap newMap = map.without(sym);  
    mappings.compareAndSet(map, newMap);  
    map = getMappings();  
}  
}  
  
public Class importClass(Symbol sym, Class c){  
    return referenceClass(sym, c);  
}  
  
public Class importClass(Class c){  
    String n = c.getName();  
    return importClass(  
        Symbol.intern(n.substring(n.lastIndexOf('.') + 1)), c);  
}  
  
public Var refer(Symbol sym, Var var){  
    return (Var) reference(sym, var);  
}  
  
public static Namespace findOrCreate(Symbol name){  
    Namespace ns = namespaces.get(name);  
    if(ns != null)  
        return ns;  
    Namespace newns = new Namespace(name);  
    ns = namespaces.putIfAbsent(name, newns);  
    return ns == null ? newns : ns;  
}  
  
public static Namespace remove(Symbol name){  
    if(name.equals(RT.CLOJURE_NS.name))  
        throw new IllegalArgumentException(  
            "Cannot remove clojure namespace");  
    return namespaces.remove(name);  
}  
  
public static Namespace find(Symbol name){  
    return namespaces.get(name);  
}  
  
public Object getMapping(Symbol name){
```

```

        return mappings.get().valAt(name);
    }

    public Var findInternedVar(Symbol symbol){
        Object o = mappings.get().valAt(symbol);
        if(o != null && o instanceof Var && ((Var) o).ns == this)
            return (Var) o;
        return null;
    }

    public IPersistentMap getAliases(){
        return aliases.get();
    }

    public Namespace lookupAlias(Symbol alias){
        IPersistentMap map = getAliases();
        return (Namespace) map.valAt(alias);
    }

    public void addAlias(Symbol alias, Namespace ns){
        if (alias == null || ns == null)
            throw new NullPointerException("Expecting Symbol + Namespace");
        IPersistentMap map = getAliases();
        while(!map.containsKey(alias))
        {
            IPersistentMap newMap = map.assoc(alias, ns);
            aliases.compareAndSet(map, newMap);
            map = getAliases();
        }
        // you can rebind an alias, but only to the
        // initially-aliased namespace.
        if(!map.valAt(alias).equals(ns))
            throw new IllegalStateException(
                "Alias " + alias + " already exists in namespace "
                + name + ", aliasing " + map.valAt(alias));
    }

    public void removeAlias(Symbol alias) throws Exception{
        IPersistentMap map = getAliases();
        while(map.containsKey(alias))
        {
            IPersistentMap newMap = map.without(alias);
            aliases.compareAndSet(map, newMap);
            map = getAliases();
        }
    }

    private Object readResolve() throws ObjectStreamException {
        // ensures that serialized namespaces are "deserialized" to the

```

```
// namespace in the present runtime
    return findOrCreate(name);
}
}
```

—————

9.76 Numbers.java

— Numbers.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 31, 2008 */

package clojure.lang;

import java.math.BigInteger;
import java.math.BigDecimal;
import java.math.MathContext;

public class Numbers{

    static interface Ops{
        Ops combine(Ops y);

        Ops opsWith(LongOps x);

        Ops opsWith(DoubleOps x);

        Ops opsWith(RatioOps x);

        Ops opsWith(BigIntOps x);

        Ops opsWith(BigDecimalOps x);

        public boolean isZero(Number x);

        public boolean isPos(Number x);

        public boolean isNeg(Number x);

        public Number add(Number x, Number y);
        public Number addP(Number x, Number y);

        public Number multiply(Number x, Number y);
    }
}
```

```
public Number multiplyP(Number x, Number y);

public Number divide(Number x, Number y);

public Number quotient(Number x, Number y);

public Number remainder(Number x, Number y);

public boolean equiv(Number x, Number y);

public boolean lt(Number x, Number y);

public Number negate(Number x);
public Number negateP(Number x);

public Number inc(Number x);
public Number incP(Number x);

public Number dec(Number x);
public Number decP(Number x);
}

static abstract class OpsP implements Ops{
    public Number addP(Number x, Number y){
        return add(x, y);
    }

    public Number multiplyP(Number x, Number y){
        return multiply(x, y);
    }

    public Number negateP(Number x){
        return negate(x);
    }

    public Number incP(Number x){
        return inc(x);
    }

    public Number decP(Number x){
        return dec(x);
    }
}

static interface BitOps{
    BitOps combine(BitOps y);

    BitOps bitOpsWith(LongBitOps x);

    BitOps bitOpsWith(BigIntBitOps x);
}
```

```
public Number not(Number x);

public Number and(Number x, Number y);

public Number or(Number x, Number y);

public Number xor(Number x, Number y);

public Number andNot(Number x, Number y);

public Number clearBit(Number x, int n);

public Number setBit(Number x, int n);

public Number flipBit(Number x, int n);

public boolean testBit(Number x, int n);

public Number shiftLeft(Number x, int n);

public Number shiftRight(Number x, int n);

}

static public boolean isZero(Object x){
    return ops(x).isZero((Number)x);
}

static public boolean isPos(Object x){
    return ops(x).isPos((Number)x);
}

static public boolean isNeg(Object x){
    return ops(x).isNeg((Number)x);
}

static public Number minus(Object x){
    return ops(x).negate((Number)x);
}

static public Number minusP(Object x){
    return ops(x).negateP((Number)x);
}

static public Number inc(Object x){
    return ops(x).inc((Number)x);
}

static public Number incP(Object x){
```

```

        return ops(x).incP((Number)x);
    }

    static public Number dec(Object x){
        return ops(x).dec((Number)x);
    }

    static public Number decP(Object x){
        return ops(x).decP((Number)x);
    }

    static public Number add(Object x, Object y){
        return ops(x).combine(ops(y)).add((Number)x, (Number)y);
    }

    static public Number addP(Object x, Object y){
        return ops(x).combine(ops(y)).addP((Number)x, (Number)y);
    }

    static public Number minus(Object x, Object y){
        Ops yops = ops(y);
        return ops(x).combine(yops).add((Number)x, yops.negate((Number)y));
    }

    static public Number minusP(Object x, Object y){
        Ops yops = ops(y);
        return ops(x).combine(yops).addP((Number)x, yops.negateP((Number)y));
    }

    static public Number multiply(Object x, Object y){
        return ops(x).combine(ops(y)).multiply((Number)x, (Number)y);
    }

    static public Number multiplyP(Object x, Object y){
        return ops(x).combine(ops(y)).multiplyP((Number)x, (Number)y);
    }

    static public Number divide(Object x, Object y){
        Ops yops = ops(y);
        if(yops.isZero((Number)y))
            throw new ArithmeticException("Divide by zero");
        return ops(x).combine(yops).divide((Number)x, (Number)y);
    }

    static public Number quotient(Object x, Object y){
        Ops yops = ops(y);
        if(yops.isZero((Number) y))
            throw new ArithmeticException("Divide by zero");
        return ops(x).combine(yops).quotient((Number)x, (Number)y);
    }
}

```

```
static public Number remainder(Object x, Object y){  
    Ops yops = ops(y);  
    if(yops.isZero((Number) y))  
        throw new ArithmeticException("Divide by zero");  
    return ops(x).combine(yops).remainder((Number)x, (Number)y);  
}  
  
static public double quotient(double n, double d){  
    if(d == 0)  
        throw new ArithmeticException("Divide by zero");  
  
    double q = n / d;  
    if(q <= Long.MAX_VALUE && q >= Long.MIN_VALUE)  
    {  
        return (double)(long) q;  
    }  
    else  
    { //bigint quotient  
        return new BigDecimal(q).toBigInteger().doubleValue();  
    }  
}  
  
static public double remainder(double n, double d){  
    if(d == 0)  
        throw new ArithmeticException("Divide by zero");  
  
    double q = n / d;  
    if(q <= Long.MAX_VALUE && q >= Long.MIN_VALUE)  
    {  
        return (n - ((long) q) * d);  
    }  
    else  
    { //bigint quotient  
        Number bq = new BigDecimal(q).toBigInteger();  
        return (n - bq.doubleValue() * d);  
    }  
}  
  
static public boolean equiv(Object x, Object y){  
    return equiv((Number) x, (Number) y);  
}  
  
static public boolean equiv(Number x, Number y){  
    return ops(x).combine(ops(y)).equiv(x, y);  
}  
  
static public boolean equal(Number x, Number y){  
    return category(x) == category(y)  
        && ops(x).combine(ops(y)).equiv(x, y);  
}
```

```
}

static public boolean lt(Object x, Object y){
    return ops(x).combine(ops(y)).lt((Number)x, (Number)y);
}

static public boolean lte(Object x, Object y){
    return !ops(x).combine(ops(y)).lt((Number)y, (Number)x);
}

static public boolean gt(Object x, Object y){
    return ops(x).combine(ops(y)).lt((Number)y, (Number)x);
}

static public boolean gte(Object x, Object y){
    return !ops(x).combine(ops(y)).lt((Number)x, (Number)y);
}

static public int compare(Number x, Number y){
    Ops ops = ops(x).combine(ops(y));
    if(ops.lt(x, y))
        return -1;
    else if(ops.lt(y, x))
        return 1;
    return 0;
}

static BigInt toBigInt(Object x){
    if(x instanceof BigInt)
        return (BigInt) x;
    if(x instanceof BigInteger)
        return BigInt.fromBigInteger((BigInteger) x);
    else
        return BigInt.fromLong(((Number) x).longValue());
}

static BigInteger toBigInteger(Object x){
    if(x instanceof BigInteger)
        return (BigInteger) x;
    else if(x instanceof BigInt)
        return ((BigInt) x).toBigInteger();
    else
        return BigInteger.valueOf(((Number) x).longValue());
}

static BigDecimal toBigDecimal(Object x){
    if(x instanceof BigDecimal)
        return (BigDecimal) x;
    else if(x instanceof BigInt)
    {
```

```

        BigInt bi = (BigInt) x;
        if(bi.bipart == null)
            return BigDecimal.valueOf(bi.lpart);
        else
            return new BigDecimal(bi.bipart);
    }
    else if(x instanceof BigInteger)
        return new BigDecimal((BigInteger) x);
    else if(x instanceof Double)
        return new BigDecimal(((Number) x).doubleValue());
    else if(x instanceof Float)
        return new BigDecimal(((Number) x).doubleValue());
    else if(x instanceof Ratio)
    {
        Ratio r = (Ratio)x;
        return (BigDecimal)divide(new BigDecimal(r.numerator),
                               r.denominator);
    }
    else
        return BigDecimal.valueOf(((Number) x).longValue());
}

static public Ratio toRatio(Object x){
    if(x instanceof Ratio)
        return (Ratio) x;
    else if(x instanceof BigDecimal)
    {
        BigDecimal bx = (BigDecimal) x;
        BigInteger bv = bx.unscaledValue();
        int scale = bx.scale();
        if(scale < 0)
            return
                new Ratio(bv.multiply(BigInteger.TEN.pow(-scale)),
                          BigInteger.ONE);
        else
            return new Ratio(bv, BigInteger.TEN.pow(scale));
    }
    return new Ratio(toBigInteger(x), BigInteger.ONE);
}

static public Number rationalize(Number x){
    if(x instanceof Float || x instanceof Double)
        return rationalize(BigDecimal.valueOf(x.doubleValue()));
    else if(x instanceof BigDecimal)
    {
        BigDecimal bx = (BigDecimal) x;
        BigInteger bv = bx.unscaledValue();
        int scale = bx.scale();
        if(scale < 0)
            return

```

```

        BigInt.fromBigInteger(
            bv.multiply(BigInteger.TEN.pow(-scale)));
    else
        return divide(bv, BigInteger.TEN.pow(scale));
    }
    return x;
}

//static Number box(int val){
//    return Integer.valueOf(val);
//}

//static Number box(long val){
//    return Long.valueOf(val);
//}

//static Double box(double val){
//    return Double.valueOf(val);
//}

//static Double box(float val){
//    return Double.valueOf((double) val);
//}

static public Number reduceBigInt(BigInt val){
    if(val.bipart == null)
        return num(val.lpart);
    else
        return val.bipart;
}

static public Number divide(BigInteger n, BigInteger d){
    if(d.equals(BigInteger.ZERO))
        throw new ArithmeticException("Divide by zero");
    BigInteger gcd = n.gcd(d);
    if(gcd.equals(BigInteger.ZERO))
        return BigInt.ZERO;
    n = n.divide(gcd);
    d = d.divide(gcd);
    if(d.equals(BigInteger.ONE))
        return BigInt.fromBigInteger(n);
    else if(d.equals(BigInteger.ONE.negate()))
        return BigInt.fromBigInteger(n.negate());
    return new Ratio((d.signum() < 0 ? n.negate() : n),
                    (d.signum() < 0 ? d.negate() : d));
}

static public Number not(Object x){
    return bitOps(x).not((Number)x);
}

```

```
static public Number and(Object x, Object y){
    return bitOps(x).combine(bitOps(y)).and((Number)x, (Number)y);
}

static public Number or(Object x, Object y){
    return bitOps(x).combine(bitOps(y)).or((Number)x, (Number)y);
}

static public Number xor(Object x, Object y){
    return bitOps(x).combine(bitOps(y)).xor((Number)x, (Number)y);
}

static public Number andNot(Number x, Number y){
    return bitOps(x).combine(bitOps(y)).andNot(x, y);
}

static public Number clearBit(Number x, int n){
    if(n < 0)
        throw new ArithmeticException("Negative bit index");
    return bitOps(x).clearBit(x, n);
}

static public Number setBit(Number x, int n){
    if(n < 0)
        throw new ArithmeticException("Negative bit index");
    return bitOps(x).setBit(x, n);
}

static public Number flipBit(Number x, int n){
    if(n < 0)
        throw new ArithmeticException("Negative bit index");
    return bitOps(x).flipBit(x, n);
}

static public boolean testBit(Number x, int n){
    if(n < 0)
        throw new ArithmeticException("Negative bit index");
    return bitOps(x).testBit(x, n);
}

static public Number shiftLeft(Object x, Object n){
    return bitOps(x).shiftLeft((Number)x, ((Number)n).intValue());
}

static public int shiftLeftInt(int x, int n){
    return x << n;
}
```

```
static public long shiftLeft(long x, int n){
    if(n < 0)
        return shiftRight(x, -n);
    return x << n;
}

static public Number shiftRight(Object x, Object n){
    return bitOps(x).shiftRight((Number)x, ((Number)n).intValue());
}

static public int shiftRightInt(int x, int n){
    return x >> n;
}

static public long shiftRight(long x, int n){
    if(n < 0)
        return shiftLeft(x, -n);
    return x >> n;
}

final static class LongOps implements Ops{
    public Ops combine(Ops y){
        return y.opsWith(this);
    }

    final public Ops opsWith(LongOps x){
        return this;
    }

    final public Ops opsWith(DoubleOps x){
        return DOUBLE_OPS;
    }

    final public Ops opsWith(RatioOps x){
        return RATIO_OPS;
    }

    final public Ops opsWith(BigIntOps x){
        return BIGINT_OPS;
    }

    final public Ops opsWith(BigDecimalOps x){
        return BIGDECIMAL_OPS;
    }

    public boolean isZero(Number x){
        return x.longValue() == 0;
    }

    public boolean isPos(Number x){
```

```

        return x.longValue() > 0;
    }

    public boolean isNeg(Number x){
        return x.longValue() < 0;
    }

    final public Number add(Number x, Number y){
        return num(Numbers.add(x.longValue(),y.longValue()));
    }

    final public Number addP(Number x, Number y){
        long lx = x.longValue(), ly = y.longValue();
        long ret = lx + ly;
        if ((ret ^ lx) < 0 && (ret ^ ly) < 0)
            return BIGINT_OPS.add(x, y);
        return num(ret);
    }

    final public Number multiply(Number x, Number y){
        return num(Numbers.multiply(x.longValue(), y.longValue()));
    }

    final public Number multiplyP(Number x, Number y){
        long lx = x.longValue(), ly = y.longValue();
        long ret = lx * ly;
        if (ly != 0 && ret/ly != lx)
            return BIGINT_OPS.multiply(x, y);
        return num(ret);
    }

    static long gcd(long u, long v){
        while(v != 0)
        {
            long r = u % v;
            u = v;
            v = r;
        }
        return u;
    }

    public Number divide(Number x, Number y){
        long n = x.longValue();
        long val = y.longValue();
        long gcd = gcd(n, val);
        if(gcd == 0)
            return num(0);

        n = n / gcd;
        long d = val / gcd;
        if(d == 1)

```

```

        return num(n);
    if(d < 0)
    {
        n = -n;
        d = -d;
    }
    return new Ratio(BigInteger.valueOf(n), BigInteger.valueOf(d));
}

public Number quotient(Number x, Number y){
    return num(x.longValue() / y.longValue());
}

public Number remainder(Number x, Number y){
    return num(x.longValue() % y.longValue());
}

public boolean equiv(Number x, Number y){
    return x.longValue() == y.longValue();
}

public boolean lt(Number x, Number y){
    return x.longValue() < y.longValue();
}

//public Number subtract(Number x, Number y);
final public Number negate(Number x){
    long val = x.longValue();
    return num(Numbers.minus(val));
}

final public Number negateP(Number x){
    long val = x.longValue();
    if(val > Long.MIN_VALUE)
        return num(-val);
    return BigInt.fromBigInteger(BigInteger.valueOf(val).negate());
}
public Number inc(Number x){
    long val = x.longValue();
    return num(Numbers.inc(val));
}

public Number incP(Number x){
    long val = x.longValue();
    if(val < Long.MAX_VALUE)
        return num(val + 1);
    return BIGINT_OPS.inc(x);
}

public Number dec(Number x){

```

```
    long val = x.longValue();
    return num(Numbers.dec(val));
}

public Number decP(Number x){
    long val = x.longValue();
    if(val > Long.MIN_VALUE)
        return num(val - 1);
    return BIGINT_OPS.dec(x);
}
}

final static class DoubleOps extends OpsP{
    public Ops combine(Ops y){
        return y.opsWith(this);
    }

    final public Ops opsWith(LongOps x){
        return this;
    }

    final public Ops opsWith(DoubleOps x){
        return this;
    }

    final public Ops opsWith(RatioOps x){
        return this;
    }

    final public Ops opsWith(BigIntOps x){
        return this;
    }

    final public Ops opsWith(BigDecimalOps x){
        return this;
    }

    public boolean isZero(Number x){
        return x.doubleValue() == 0;
    }

    public boolean isPos(Number x){
        return x.doubleValue() > 0;
    }

    public boolean isNeg(Number x){
        return x.doubleValue() < 0;
    }

    final public Number add(Number x, Number y){
```

```

        return Double.valueOf(x.doubleValue() + y.doubleValue());
    }

    final public Number multiply(Number x, Number y){
        return Double.valueOf(x.doubleValue() * y.doubleValue());
    }

    public Number divide(Number x, Number y){
        return Double.valueOf(x.doubleValue() / y.doubleValue());
    }

    public Number quotient(Number x, Number y){
        return Numbers.quotient(x.doubleValue(), y.doubleValue());
    }

    public Number remainder(Number x, Number y){
        return Numbers.remainder(x.doubleValue(), y.doubleValue());
    }

    public boolean equiv(Number x, Number y){
        return x.doubleValue() == y.doubleValue();
    }

    public boolean lt(Number x, Number y){
        return x.doubleValue() < y.doubleValue();
    }

    //public Number subtract(Number x, Number y);
    final public Number negate(Number x){
        return Double.valueOf(-x.doubleValue());
    }

    public Number inc(Number x){
        return Double.valueOf(x.doubleValue() + 1);
    }

    public Number dec(Number x){
        return Double.valueOf(x.doubleValue() - 1);
    }
}

final static class RatioOps extends OpsP{
    public Ops combine(Ops y){
        return y.opsWith(this);
    }

    final public Ops opsWith(LongOps x){
        return this;
    }
}

```

```
final public Ops opsWith(DoubleOps x){
    return DOUBLE_OPS;
}

final public Ops opsWith(RatioOps x){
    return this;
}

final public Ops opsWith(BigIntOps x){
    return this;
}

final public Ops opsWith(BigDecimalOps x){
    return BIGDECIMAL_OPS;
}

public boolean isZero(Number x){
    Ratio r = (Ratio) x;
    return r.numerator.signum() == 0;
}

public boolean isPos(Number x){
    Ratio r = (Ratio) x;
    return r.numerator.signum() > 0;
}

public boolean isNeg(Number x){
    Ratio r = (Ratio) x;
    return r.numerator.signum() < 0;
}

static Number normalizeRet(Number ret, Number x, Number y){
//    if(ret instanceof BigInteger &&
//        !(x instanceof BigInteger ||
//        y instanceof BigInteger))
//    {
//        return reduceBigInt((BigInteger) ret);
//    }
    return ret;
}

final public Number add(Number x, Number y){
    Ratio rx = toRatio(x);
    Ratio ry = toRatio(y);
    Number ret = divide(ry.numerator.multiply(rx.denominator)
        .add(rx.numerator.multiply(ry.denominator))
        , ry.denominator.multiply(rx.denominator));
    return normalizeRet(ret, x, y);
}
```

```

final public Number multiply(Number x, Number y){
    Ratio rx = toRatio(x);
    Ratio ry = toRatio(y);
    Number ret = Numbers.divide(ry.numerator.multiply(rx.numerator)
        , ry.denominator.multiply(rx.denominator));
    return normalizeRet(ret, x, y);
}

public Number divide(Number x, Number y){
    Ratio rx = toRatio(x);
    Ratio ry = toRatio(y);
    Number ret = Numbers.divide(ry.denominator.multiply(rx.numerator)
        , ry.numerator.multiply(rx.denominator));
    return normalizeRet(ret, x, y);
}

public Number quotient(Number x, Number y){
    Ratio rx = toRatio(x);
    Ratio ry = toRatio(y);
    BigInteger q = rx.numerator.multiply(ry.denominator).divide(
        rx.denominator.multiply(ry.numerator));
    return normalizeRet(BigInt.fromBigInteger(q), x, y);
}

public Number remainder(Number x, Number y){
    Ratio rx = toRatio(x);
    Ratio ry = toRatio(y);
    BigInteger q = rx.numerator.multiply(ry.denominator).divide(
        rx.denominator.multiply(ry.numerator));
    Number ret = Numbers.minus(x, Numbers.multiply(q, y));
    return normalizeRet(ret, x, y);
}

public boolean equiv(Number x, Number y){
    Ratio rx = toRatio(x);
    Ratio ry = toRatio(y);
    return rx.numerator.equals(ry.numerator)
        && rx.denominator.equals(ry.denominator);
}

public boolean lt(Number x, Number y){
    Ratio rx = toRatio(x);
    Ratio ry = toRatio(y);
    return Numbers.lt(
        rx.numerator.multiply(ry.denominator),
        ry.numerator.multiply(rx.denominator));
}

//public Number subtract(Number x, Number y);
final public Number negate(Number x){
}

```

```
    Ratio r = (Ratio) x;
    return new Ratio(r.numerator.negate(), r.denominator);
}

public Number inc(Number x){
    return Numbers.add(x, 1);
}

public Number dec(Number x){
    return Numbers.add(x, -1);
}

}

final static class BigIntOps extends OpsP{
    public Ops combine(Ops y){
        return y.opsWith(this);
    }

    final public Ops opsWith(LongOps x){
        return this;
    }

    final public Ops opsWith(DoubleOps x){
        return DOUBLE_OPS;
    }

    final public Ops opsWith(RatioOps x){
        return RATIO_OPS;
    }

    final public Ops opsWith(BigIntOps x){
        return this;
    }

    final public Ops opsWith(BigDecimalOps x){
        return BIGDECIMAL_OPS;
    }

    public boolean isZero(Number x){
        BigInt bx = toBigInt(x);
        if(bx.bipart == null)
            return bx.lpart == 0;
        return bx.bipart.signum() == 0;
    }

    public boolean isPos(Number x){
        BigInt bx = toBigInt(x);
        if(bx.bipart == null)
            return bx.lpart > 0;
```

```

        return bx.bipart.signum() > 0;
    }

public boolean isNeg(Number x){
    BigInt bx = toBigInt(x);
    if(bx.bipart == null)
        return bx.lpart < 0;
    return bx.bipart.signum() < 0;
}

final public Number add(Number x, Number y){
    return
        BigInt.fromBigInteger(toBigInteger(x).add(toBigInteger(y)));
}

final public Number multiply(Number x, Number y){
    return
        BigInt.fromBigInteger(
            toBigInteger(x).multiply(toBigInteger(y)));
}

public Number divide(Number x, Number y){
    return Numbers.divide(toBigInteger(x), toBigInteger(y));
}

public Number quotient(Number x, Number y){
    return
        BigInt.fromBigInteger(toBigInteger(x).divide(toBigInteger(y)));
}

public Number remainder(Number x, Number y){
    return
        BigInt.fromBigInteger(
            toBigInteger(x).remainder(toBigInteger(y)));
}

public boolean equiv(Number x, Number y){
    return toBigInt(x).equals(toBigInt(y));
}

public boolean lt(Number x, Number y){
    return toBigInteger(x).compareTo(toBigInteger(y)) < 0;
}

//public Number subtract(Number x, Number y);
final public Number negate(Number x){
    return BigInt.fromBigInteger(toBigInteger(x).negate());
}

public Number inc(Number x){
}

```

```
    BigInteger bx = toBigInteger(x);
    return BigInt.fromBigInteger(bx.add(BigInteger.ONE));
}

public Number dec(Number x){
    BigInteger bx = toBigInteger(x);
    return BigInt.fromBigInteger(bx.subtract(BigInteger.ONE));
}
}

final static class BigDecimalOps extends OpsP{
    final static Var MATH_CONTEXT = RT.MATH_CONTEXT;

    public Ops combine(Ops y){
        return y.opsWith(this);
    }

    final public Ops opsWith(LongOps x){
        return this;
    }

    final public Ops opsWith(DoubleOps x){
        return DOUBLE_OPS;
    }

    final public Ops opsWith(RatioOps x){
        return this;
    }

    final public Ops opsWith(BigIntOps x){
        return this;
    }

    final public Ops opsWith(BigDecimalOps x){
        return this;
    }

    public boolean isZero(Number x){
        BigDecimal bx = (BigDecimal) x;
        return bx.signum() == 0;
    }

    public boolean isPos(Number x){
        BigDecimal bx = (BigDecimal) x;
        return bx.signum() > 0;
    }

    public boolean isNeg(Number x){
        BigDecimal bx = (BigDecimal) x;
```

```

        return bx.signum() < 0;
    }

final public Number add(Number x, Number y){
    MathContext mc = (MathContext) MATH_CONTEXT.deref();
    return mc == null
        ? toBigDecimal(x).add(toBigDecimal(y))
        : toBigDecimal(x).add(toBigDecimal(y), mc);
}

final public Number multiply(Number x, Number y){
    MathContext mc = (MathContext) MATH_CONTEXT.deref();
    return mc == null
        ? toBigDecimal(x).multiply(toBigDecimal(y))
        : toBigDecimal(x).multiply(toBigDecimal(y), mc);
}

public Number divide(Number x, Number y){
    MathContext mc = (MathContext) MATH_CONTEXT.deref();
    return mc == null
        ? toBigDecimal(x).divide(toBigDecimal(y))
        : toBigDecimal(x).divide(toBigDecimal(y), mc);
}

public Number quotient(Number x, Number y){
    MathContext mc = (MathContext) MATH_CONTEXT.deref();
    return
        mc == null
        ? toBigDecimal(x).divideToIntegralValue(toBigDecimal(y))
        : toBigDecimal(x).divideToIntegralValue(toBigDecimal(y), mc);
}

public Number remainder(Number x, Number y){
    MathContext mc = (MathContext) MATH_CONTEXT.deref();
    return mc == null
        ? toBigDecimal(x).remainder(toBigDecimal(y))
        : toBigDecimal(x).remainder(toBigDecimal(y), mc);
}

public boolean equiv(Number x, Number y){
    return toBigDecimal(x).equals(toBigDecimal(y));
}

public boolean lt(Number x, Number y){
    return toBigDecimal(x).compareTo(toBigDecimal(y)) < 0;
}

//public Number subtract(Number x, Number y);
final public Number negate(Number x){
    MathContext mc = (MathContext) MATH_CONTEXT.deref();
}

```

```
        return mc == null
            ? ((BigDecimal) x).negate()
            : ((BigDecimal) x).negate(mc);
    }

    public Number inc(Number x){
        MathContext mc = (MathContext) MATH_CONTEXT.deref();
        BigDecimal bx = (BigDecimal) x;
        return mc == null
            ? bx.add(BigDecimal.ONE)
            : bx.add(BigDecimal.ONE, mc);
    }

    public Number dec(Number x){
        MathContext mc = (MathContext) MATH_CONTEXT.deref();
        BigDecimal bx = (BigDecimal) x;
        return mc == null
            ? bx.subtract(BigDecimal.ONE)
            : bx.subtract(BigDecimal.ONE, mc);
    }
}

final static class LongBitOps implements BitOps{
    public BitOps combine(BitOps y){
        return y.bitOpsWith(this);
    }

    final public BitOps bitOpsWith(LongBitOps x){
        return this;
    }

    final public BitOps bitOpsWith(BigIntBitOps x){
        return BIGINT_BITOPS;
    }

    public Number not(Number x){
        return num(~x.longValue());
    }

    public Number and(Number x, Number y){
        return num(x.longValue() & y.longValue());
    }

    public Number or(Number x, Number y){
        return num(x.longValue() | y.longValue());
    }

    public Number xor(Number x, Number y){
        return num(x.longValue() ^ y.longValue());
    }
}
```

```

public Number andNot(Number x, Number y){
    return num(x.longValue() & ~y.longValue());
}

public Number clearBit(Number x, int n){
    if(n < 63)
        return (num(x.longValue() & ~(1L << n)));
    else
        return BigInt.fromBigInteger(toBigInteger(x).clearBit(n));
}

public Number setBit(Number x, int n){
    if(n < 63)
        return num(x.longValue() | (1L << n));
    else
        return BigInt.fromBigInteger(toBigInteger(x).setBit(n));
}

public Number flipBit(Number x, int n){
    if(n < 63)
        return num(x.longValue() ^ (1L << n));
    else
        return BigInt.fromBigInteger(toBigInteger(x).flipBit(n));
}

public boolean testBit(Number x, int n){
    if(n < 64)
        return (x.longValue() & (1L << n)) != 0;
    else
        return toBigInteger(x).testBit(n);
}

public Number shiftLeft(Number x, int n){
    if(n < 0)
        return shiftRight(x, -n);
    return num(Numbers.shiftLeft(x.longValue(), n));
}

public Number shiftRight(Number x, int n){
    if(n < 0)
        return shiftLeft(x, -n);
    return num(x.longValue() >> n);
}

final static class BigIntBitOps implements BitOps{
    public BitOps combine(BitOps y){
        return y.bitOpsWith(this);
    }
}

```

```
final public BitOps bitOpsWith(LongBitOps x){
    return this;
}

final public BitOps bitOpsWith(BigIntBitOps x){
    return this;
}

public Number not(Number x){
    return BigInt.fromBigInteger(toBigInteger(x).not());
}

public Number and(Number x, Number y){
    return
        BigInt.fromBigInteger(toBigInteger(x).and(toBigInteger(y)));
}

public Number or(Number x, Number y){
    return
        BigInt.fromBigInteger(toBigInteger(x).or(toBigInteger(y)));
}

public Number xor(Number x, Number y){
    return
        BigInt.fromBigInteger(toBigInteger(x).xor(toBigInteger(y)));
}

public Number andNot(Number x, Number y){
    return
        BigInt.fromBigInteger(toBigInteger(x).andNot(toBigInteger(y)));
}

public Number clearBit(Number x, int n){
    return BigInt.fromBigInteger(toBigInteger(x).clearBit(n));
}

public Number setBit(Number x, int n){
    return BigInt.fromBigInteger(toBigInteger(x).setBit(n));
}

public Number flipBit(Number x, int n){
    return BigInt.fromBigInteger(toBigInteger(x).flipBit(n));
}

public boolean testBit(Number x, int n){
    return toBigInteger(x).testBit(n);
}

public Number shiftLeft(Number x, int n){
```

```

        return BigInt.fromBigInteger(toBigInteger(x).shiftLeft(n));
    }

    public Number shiftRight(Number x, int n){
        return BigInt.fromBigInteger(toBigInteger(x).shiftRight(n));
    }
}

static final LongOps LONG_OPS = new LongOps();
static final DoubleOps DOUBLE_OPS = new DoubleOps();
static final RatioOps RATIO_OPS = new RatioOps();
static final BigIntOps BIGINT_OPS = new BigIntOps();
static final BigDecimalOps BIGDECIMAL_OPS = new BigDecimalOps();

static final LongBitOps LONG_BITOPS = new LongBitOps();
static final BigIntBitOps BIGINT_BITOPS = new BigIntBitOps();

static public enum Category {INTEGER, FLOATING, DECIMAL, RATIO};

static Ops ops(Object x){
    Class xc = x.getClass();

    if(xc == Integer.class)
        return LONG_OPS;
    else if(xc == Double.class)
        return DOUBLE_OPS;
    else if(xc == Long.class)
        return LONG_OPS;
    else if(xc == Float.class)
        return DOUBLE_OPS;
    else if(xc == BigInt.class)
        return BIGINT_OPS;
    else if(xc == BigInteger.class)
        return BIGINT_OPS;
    else if(xc == Ratio.class)
        return RATIO_OPS;
    else if(xc == BigDecimal.class)
        return BIGDECIMAL_OPS;
    else
        return LONG_OPS;
}

static Category category(Object x){
    Class xc = x.getClass();

    if(xc == Integer.class)
        return Category.INTEGER;
    else if(xc == Double.class)
        return Category.FLOATING;
    else if(xc == Long.class)

```

```

        return Category.INTEGER;
    else if(xc == Float.class)
        return Category.FLOATING;
    else if(xc == BigInt.class)
        return Category.INTEGER;
    else if(xc == Ratio.class)
        return Category.RATIO;
    else if(xc == BigDecimal.class)
        return Category.DECIMAL;
    else
        return Category.INTEGER;
}

static BitOps bitOps(Object x){
    Class xc = x.getClass();

    if(xc == Long.class)
        return LONG_BITOPS;
    else if(xc == Integer.class)
        return LONG_BITOPS;
    else if(xc == BigInt.class)
        return BIGINT_BITOPS;
    else if(xc == BigInteger.class)
        return BIGINT_BITOPS;
    else if(xc == Double.class ||
            xc == Float.class ||
            xc == BigDecimalOps.class ||
            xc == Ratio.class)
        throw new ArithmeticException(
            "bit operation on non integer type: " + xc);
    else
        return LONG_BITOPS;
}

static public float[] float_array(int size, Object init){
    float[] ret = new float[size];
    if(init instanceof Number)
    {
        float f = ((Number) init).floatValue();
        for(int i = 0; i < ret.length; i++)
            ret[i] = f;
    }
    else
    {
        ISeq s = RT.seq(init);
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = ((Number) s.first()).floatValue();
    }
    return ret;
}

```

```

static public float[] float_array(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new float[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = RT.count(s);
        float[] ret = new float[size];
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = ((Number) s.first()).floatValue();
        return ret;
    }
}

static public double[] double_array(int size, Object init){
    double[] ret = new double[size];
    if(init instanceof Number)
    {
        double f = ((Number) init).doubleValue();
        for(int i = 0; i < ret.length; i++)
            ret[i] = f;
    }
    else
    {
        ISeq s = RT.seq(init);
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = ((Number) s.first()).doubleValue();
    }
    return ret;
}

static public double[] double_array(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new double[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = RT.count(s);
        double[] ret = new double[size];
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = ((Number) s.first()).doubleValue();
        return ret;
    }
}

static public int[] int_array(int size, Object init){
    int[] ret = new int[size];
    if(init instanceof Number)
    {

```

```
int f = ((Number) init).intValue();
for(int i = 0; i < ret.length; i++)
    ret[i] = f;
}
else
{
    ISeq s = RT.seq(init);
    for(int i = 0; i < size && s != null; i++, s = s.next())
        ret[i] = ((Number) s.first()).intValue();
}
return ret;
}

static public int[] int_array(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new int[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = RT.count(s);
        int[] ret = new int[size];
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = ((Number) s.first()).intValue();
        return ret;
    }
}

static public long[] long_array(int size, Object init){
    long[] ret = new long[size];
    if(init instanceof Number)
    {
        long f = ((Number) init).longValue();
        for(int i = 0; i < ret.length; i++)
            ret[i] = f;
    }
    else
    {
        ISeq s = RT.seq(init);
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = ((Number) s.first()).longValue();
    }
    return ret;
}

static public long[] long_array(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new long[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
```

```

        int size = RT.count(s);
        long[] ret = new long[size];
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = ((Number) s.first()).longValue();
        return ret;
    }
}

static public short[] short_array(int size, Object init){
    short[] ret = new short[size];
    if(init instanceof Short)
    {
        short s = (Short) init;
        for(int i = 0; i < ret.length; i++)
            ret[i] = s;
    }
    else
    {
        ISeq s = RT.seq(init);
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = (Short) s.first();
    }
    return ret;
}

static public short[] short_array(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new short[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = RT.count(s);
        short[] ret = new short[size];
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = (Short) s.first();
        return ret;
    }
}

static public char[] char_array(int size, Object init){
    char[] ret = new char[size];
    if(init instanceof Character)
    {
        char c = (Character) init;
        for(int i = 0; i < ret.length; i++)
            ret[i] = c;
    }
    else
    {
        ISeq s = RT.seq(init);

```

```
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = (Character) s.first();
    }
    return ret;
}

static public char[] char_array(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new char[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = RT.count(s);
        char[] ret = new char[size];
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = (Character) s.first();
        return ret;
    }
}

static public byte[] byte_array(int size, Object init){
    byte[] ret = new byte[size];
    if(init instanceof Byte)
    {
        byte b = (Byte) init;
        for(int i = 0; i < ret.length; i++)
            ret[i] = b;
    }
    else
    {
        ISeq s = RT.seq(init);
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = (Byte) s.first();
    }
    return ret;
}

static public byte[] byte_array(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new byte[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = RT.count(s);
        byte[] ret = new byte[size];
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = (Byte)s.first();
        return ret;
    }
}
```

```
static public boolean[] boolean_array(int size, Object init){
    boolean[] ret = new boolean[size];
    if(init instanceof Boolean)
    {
        boolean b = (Boolean) init;
        for(int i = 0; i < ret.length; i++)
            ret[i] = b;
    }
    else
    {
        ISeq s = RT.seq(init);
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = (Boolean)s.first();
    }
    return ret;
}

static public boolean[] boolean_array(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new boolean[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = RT.count(s);
        boolean[] ret = new boolean[size];
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = (Boolean)s.first();
        return ret;
    }
}

static public boolean[] booleans(Object array){
    return (boolean[]) array;
}

static public byte[] bytes(Object array){
    return (byte[]) array;
}

static public char[] chars(Object array){
    return (char[]) array;
}

static public short[] shorts(Object array){
    return (short[]) array;
}

static public float[] floats(Object array){
    return (float[]) array;
```

```
}

static public double[] doubles(Object array){
    return (double[]) array;
}

static public int[] ints(Object array){
    return (int[]) array;
}

static public long[] longs(Object array){
    return (long[]) array;
}

static public Number num(Object x){
    return (Number) x;
}

static public Number num(float x){
    return Double.valueOf(x);
}

static public Number num(double x){
    return Double.valueOf(x);
}

static public double add(double x, double y){
    return x + y;
}

static public double addP(double x, double y){
    return x + y;
}

static public double minus(double x, double y){
    return x - y;
}

static public double minusP(double x, double y){
    return x - y;
}

static public double minus(double x){
    return -x;
}

static public double minusP(double x){
    return -x;
}
```

```
static public double inc(double x){
    return x + 1;
}

static public double incP(double x){
    return x + 1;
}

static public double dec(double x){
    return x - 1;
}

static public double decP(double x){
    return x - 1;
}

static public double multiply(double x, double y){
    return x * y;
}

static public double multiplyP(double x, double y){
    return x * y;
}

static public double divide(double x, double y){
    return x / y;
}

static public boolean equiv(double x, double y){
    return x == y;
}

static public boolean lt(double x, double y){
    return x < y;
}

static public boolean lte(double x, double y){
    return x <= y;
}

static public boolean gt(double x, double y){
    return x > y;
}

static public boolean gte(double x, double y){
    return x >= y;
}

static public boolean isPos(double x){
    return x > 0;
```

```
}

static public boolean isNeg(double x){
    return x < 0;
}

static public boolean isZero(double x){
    return x == 0;
}

static int throwIntOverflow(){
    throw new ArithmeticException("integer overflow");
}

//static public Number num(int x){
//    return Integer.valueOf(x);
//}

static public int unchecked_int_add(int x, int y){
    return x + y;
}

static public int unchecked_int_subtract(int x, int y){
    return x - y;
}

static public int unchecked_int_negate(int x){
    return -x;
}

static public int unchecked_int_inc(int x){
    return x + 1;
}

static public int unchecked_int_dec(int x){
    return x - 1;
}

static public int unchecked_int_multiply(int x, int y){
    return x * y;
}

//static public int add(int x, int y){
//    int ret = x + y;
//    if ((ret ^ x) < 0 && (ret ^ y) < 0)
//        return throwIntOverflow();
//    return ret;
//}

//static public int not(int x){
```

```
//      return ~x;
//}

static public long not(long x){
    return ~x;
}
//static public int and(int x, int y){
//    return x & y;
//}

static public long and(long x, long y){
    return x & y;
}

//static public int or(int x, int y){
//    return x | y;
//}

static public long or(long x, long y){
    return x | y;
}

//static public int xor(int x, int y){
//    return x ^ y;
//}

static public long xor(long x, long y){
    return x ^ y;
}

//static public int minus(int x, int y){
//    int ret = x - y;
//    if (((ret ^ x) < 0 && (ret ^ ~y) < 0))
//        return throwIntOverflow();
//    return ret;
//}

//static public int minus(int x){
//    if(x == Integer.MIN_VALUE)
//        return throwIntOverflow();
//    return -x;
//}

//static public int inc(int x){
//    if(x == Integer.MAX_VALUE)
//        return throwIntOverflow();
//    return x + 1;
//}

//static public int dec(int x){
```

```
//      if(x == Integer.MIN_VALUE)
//          return throwIntOverflow();
//      return x - 1;
//}

//static public int multiply(int x, int y){
//    int ret = x * y;
//    if (y != 0 && ret/y != x)
//        return throwIntOverflow();
//    return ret;
//}

static public int unchecked_int_divide(int x, int y){
    return x / y;
}

static public int unchecked_int_remainder(int x, int y){
    return x % y;
}

//static public boolean equiv(int x, int y){
//    return x == y;
//}

//static public boolean lt(int x, int y){
//    return x < y;
//}

//static public boolean lte(int x, int y){
//    return x <= y;
//}

//static public boolean gt(int x, int y){
//    return x > y;
//}

//static public boolean gte(int x, int y){
//    return x >= y;
//}

//static public boolean isPos(int x){
//    return x > 0;
//}

//static public boolean isNeg(int x){
//    return x < 0;
//}

//static public boolean isZero(int x){
//    return x == 0;
```

```
//}

static public Number num(long x){
    return Long.valueOf(x);
}

static public long
    unchecked_add(long x, long y){return x + y;}
static public long
    unchecked_minus(long x, long y){return x - y;}
static public long
    unchecked_multiply(long x, long y){return x * y;}
static public long
    unchecked_minus(long x){return -x;}
static public long
    unchecked_inc(long x){return x + 1;}
static public long
    unchecked_dec(long x){return x - 1;}

static public Number
    unchecked_add(Object x, Object y){return add(x,y);}
static public Number
    unchecked_minus(Object x, Object y){return minus(x,y);}
static public Number
    unchecked_multiply(Object x, Object y){return multiply(x,y);}
static public Number
    unchecked_minus(Object x){return minus(x);}
static public Number
    unchecked_inc(Object x){return inc(x);}
static public Number
    unchecked_dec(Object x){return dec(x);}

static public double
    unchecked_add(double x, double y){return add(x,y);}
static public double
    unchecked_minus(double x, double y){return minus(x,y);}
static public double
    unchecked_multiply(double x, double y){return multiply(x,y);}
static public double
    unchecked_minus(double x){return minus(x);}
static public double
    unchecked_inc(double x){return inc(x);}
static public double
    unchecked_dec(double x){return dec(x);}

static public double
    unchecked_add(double x, Object y){return add(x,y);}
static public double
    unchecked_minus(double x, Object y){return minus(x,y);}
static public double
```

```
    unchecked_multiply(double x, Object y){return multiply(x,y);}
static public double
    unchecked_add(Object x, double y){return add(x,y);}
static public double
    unchecked_minus(Object x, double y){return minus(x,y);}
static public double
    unchecked_multiply(Object x, double y){return multiply(x,y);}

static public double
    unchecked_add(double x, long y){return add(x,y);}
static public double
    unchecked_minus(double x, long y){return minus(x,y);}
static public double
    unchecked_multiply(double x, long y){return multiply(x,y);}
static public double
    unchecked_add(long x, double y){return add(x,y);}
static public double
    unchecked_minus(long x, double y){return minus(x,y);}
static public double
    unchecked_multiply(long x, double y){return multiply(x,y);}

static public Number
    unchecked_add(long x, Object y){return add(x,y);}
static public Number
    unchecked_minus(long x, Object y){return minus(x,y);}
static public Number
    unchecked_multiply(long x, Object y){return multiply(x,y);}
static public Number
    unchecked_add(Object x, long y){return add(x,y);}
static public Number
    unchecked_minus(Object x, long y){return minus(x,y);}
static public Number
    unchecked_multiply(Object x, long y){return multiply(x,y);}

static public Number
    quotient(double x, Object y){return quotient((Object)x,y);}
static public Number
    quotient(Object x, double y){return quotient(x,(Object)y);}
static public Number
    quotient(long x, Object y){return quotient((Object)x,y);}
static public Number
    quotient(Object x, long y){return quotient(x,(Object)y);}
static public double
    quotient(double x, long y){return quotient(x,(double)y);}
static public double
    quotient(long x, double y){return quotient((double)x,y);}

static public Number
    remainder(double x, Object y){return remainder((Object)x,y);}
static public Number
```

```

        remainder(Object x, double y){return remainder(x,(Object)y);}
static public Number
    remainder(long x, Object y){return remainder((Object)x,y);}
static public Number
    remainder(Object x, long y){return remainder(x,(Object)y);}
static public double
    remainder(double x, long y){return remainder(x,(double)y);}
static public double
    remainder(long x, double y){return remainder((double)x,y);}

static public long add(long x, long y){
    long ret = x + y;
    if ((ret ^ x) < 0 && (ret ^ y) < 0)
        return throwIntOverflow();
    return ret;
}

static public Number addP(long x, long y){
    long ret = x + y;
    if ((ret ^ x) < 0 && (ret ^ y) < 0)
        return addP((Number)x,(Number)y);
    return num(ret);
}

static public long minus(long x, long y){
    long ret = x - y;
    if (((ret ^ x) < 0 && (ret ^ ~y) < 0))
        return throwIntOverflow();
    return ret;
}

static public Number minusP(long x, long y){
    long ret = x - y;
    if (((ret ^ x) < 0 && (ret ^ ~y) < 0))
        return minusP((Number)x,(Number)y);
    return num(ret);
}

static public long minus(long x){
    if(x == Long.MIN_VALUE)
        return throwIntOverflow();
    return -x;
}

static public Number minusP(long x){
    if(x == Long.MIN_VALUE)
        return BigInt.fromBigInteger(BigInteger.valueOf(x).negate());
    return num(-x);
}

```

```
static public long inc(long x){
    if(x == Long.MAX_VALUE)
        return throwIntOverflow();
    return x + 1;
}

static public Number incP(long x){
    if(x == Long.MAX_VALUE)
        return BIGINT_OPS.inc(x);
    return num(x + 1);
}

static public long dec(long x){
    if(x == Long.MIN_VALUE)
        return throwIntOverflow();
    return x - 1;
}

static public Number decP(long x){
    if(x == Long.MIN_VALUE)
        return BIGINT_OPS.dec(x);
    return num(x - 1);
}

static public long multiply(long x, long y){
    long ret = x * y;
    if (y != 0 && ret/y != x)
        return throwIntOverflow();
    return ret;
}

static public Number multiplyP(long x, long y){
    long ret = x * y;
    if (y != 0 && ret/y != x)
        return multiplyP((Number)x,(Number)y);
    return num(ret);
}

static public long quotient(long x, long y){
    return x / y;
}

static public long remainder(long x, long y){
    return x % y;
}

static public boolean equiv(long x, long y){
    return x == y;
}
```

```
static public boolean lt(long x, long y){
    return x < y;
}

static public boolean lte(long x, long y){
    return x <= y;
}

static public boolean gt(long x, long y){
    return x > y;
}

static public boolean gte(long x, long y){
    return x >= y;
}

static public boolean isPos(long x){
    return x > 0;
}

static public boolean isNeg(long x){
    return x < 0;
}

static public boolean isZero(long x){
    return x == 0;
}

/*
static public class F{
    static public float add(float x, float y){
        return x + y;
    }

    static public float subtract(float x, float y){
        return x - y;
    }

    static public float negate(float x){
        return -x;
    }

    static public float inc(float x){
        return x + 1;
    }

    static public float dec(float x){
        return x - 1;
    }
}
```

```
static public float multiply(float x, float y){  
    return x * y;  
}  
  
static public float divide(float x, float y){  
    return x / y;  
}  
  
static public boolean equiv(float x, float y){  
    return x == y;  
}  
  
static public boolean lt(float x, float y){  
    return x < y;  
}  
  
static public boolean lte(float x, float y){  
    return x <= y;  
}  
  
static public boolean gt(float x, float y){  
    return x > y;  
}  
  
static public boolean gte(float x, float y){  
    return x >= y;  
}  
  
static public boolean pos(float x){  
    return x > 0;  
}  
  
static public boolean neg(float x){  
    return x < 0;  
}  
  
static public boolean zero(float x){  
    return x == 0;  
}  
  
static public float aget(float[] xs, int i){  
    return xs[i];  
}  
  
static public float aset(float[] xs, int i, float v){  
    xs[i] = v;  
    return v;  
}
```

```

static public int alength(float[] xs){
    return xs.length;
}

static public float[] aclone(float[] xs){
    return xs.clone();
}

static public float[] vec(int size, Object init){
    float[] ret = new float[size];
    if(init instanceof Number)
    {
        float f = ((Number) init).floatValue();
        for(int i = 0; i < ret.length; i++)
            ret[i] = f;
    }
    else
    {
        ISeq s = RT.seq(init);
        for(int i = 0; i < size && s != null; i++, s = s.rest())
            ret[i] = ((Number) s.first()).floatValue();
    }
    return ret;
}

static public float[] vec(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new float[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = s.count();
        float[] ret = new float[size];
        for(int i = 0; i < size && s != null; i++, s = s.rest())
            ret[i] = ((Number) s.first()).intValue();
        return ret;
    }
}

static public float[] vsadd(float[] x, float y){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] += y;
    return xs;
}

static public float[] vssub(float[] x, float y){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)

```

```
        xs[i] -= y;
    return xs;
}

static public float[] vsdiv(float[] x, float y){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] /= y;
    return xs;
}

static public float[] vsmul(float[] x, float y){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] *= y;
    return xs;
}

static public float[] svdiv(float y, float[] x){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = y / xs[i];
    return xs;
}

static public float[] vsmuladd(float[] x, float y, float[] zs){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y + zs[i];
    return xs;
}

static public float[] vsmulsub(float[] x, float y, float[] zs){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y - zs[i];
    return xs;
}

static public float[] vsmulsadd(float[] x, float y, float z){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y + z;
    return xs;
}

static public float[] vsmulssub(float[] x, float y, float z){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y - z;
```

```
        return xs;
    }

    static public float[] vabs(float[] x){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = Math.abs(xs[i]);
        return xs;
    }

    static public float[] vnegabs(float[] x){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = -Math.abs(xs[i]);
        return xs;
    }

    static public float[] vneg(float[] x){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = -xs[i];
        return xs;
    }

    static public float[] vsqr(float[] x){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] *= xs[i];
        return xs;
    }

    static public float[] vsignedsqr(float[] x){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] *= Math.abs(xs[i]);
        return xs;
    }

    static public float[] vclip(float[] x, float low, float high){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
        {
            if(xs[i] < low)
                xs[i] = low;
            else if(xs[i] > high)
                xs[i] = high;
        }
        return xs;
    }
```

```

static public IPersistentVector vclipcounts(float[] x,
                                             float low,
                                             float high){
    final float[] xs = x.clone();
    int lowc = 0;
    int highc = 0;

    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < low)
        {
            ++lowc;
            xs[i] = low;
        }
        else if(xs[i] > high)
        {
            ++highc;
            xs[i] = high;
        }
    }
    return RT.vector(xs, lowc, highc);
}

static public float[] vthresh(float[] x,
                             float thresh,
                             float otherwise){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < thresh)
            xs[i] = otherwise;
    }
    return xs;
}

static public float[] vreverse(float[] x){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[xs.length - i - 1];
    return xs;
}

static public float[] vrunningsum(float[] x){
    final float[] xs = x.clone();
    for(int i = 1; i < xs.length; i++)
        xs[i] = xs[i - 1] + xs[i];
    return xs;
}

static public float[] vsort(float[] x){

```

```
final float[] xs = x.clone();
Arrays.sort(xs);
return xs;
}

static public float vdot(float[] xs, float[] ys){
    float ret = 0;
    for(int i = 0; i < xs.length; i++)
        ret += xs[i] * ys[i];
    return ret;
}

static public float vmax(float[] xs){
    if(xs.length == 0)
        return 0;
    float ret = xs[0];
    for(int i = 0; i < xs.length; i++)
        ret = Math.max(ret, xs[i]);
    return ret;
}

static public float vmin(float[] xs){
    if(xs.length == 0)
        return 0;
    float ret = xs[0];
    for(int i = 0; i < xs.length; i++)
        ret = Math.min(ret, xs[i]);
    return ret;
}

static public float vmean(float[] xs){
    if(xs.length == 0)
        return 0;
    return vsum(xs) / xs.length;
}

static public double vrms(float[] xs){
    if(xs.length == 0)
        return 0;
    float ret = 0;
    for(int i = 0; i < xs.length; i++)
        ret += xs[i] * xs[i];
    return Math.sqrt(ret / xs.length);
}

static public float vsum(float[] xs){
    float ret = 0;
    for(int i = 0; i < xs.length; i++)
        ret += xs[i];
    return ret;
}
```

```
}

static public boolean vequiv(float[] xs, float[] ys){
    return Arrays.equals(xs, ys);
}

static public float[] vadd(float[] x, float[] ys){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] += ys[i];
    return xs;
}

static public float[] vsub(float[] x, float[] ys){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] -= ys[i];
    return xs;
}

static public float[] vaddmul(float[] x, float[] ys, float[] zs){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] + ys[i]) * zs[i];
    return xs;
}

static public float[] vsubmul(float[] x, float[] ys, float[] zs){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] - ys[i]) * zs[i];
    return xs;
}

static public float[] vaddsmul(float[] x, float[] ys, float z){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] + ys[i]) * z;
    return xs;
}

static public float[] vsubsmul(float[] x, float[] ys, float z){
    final float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] - ys[i]) * z;
    return xs;
}

static public float[] vmulsadd(float[] x, float[] ys, float z){
    final float[] xs = x.clone();
```

```

        for(int i = 0; i < xs.length; i++)
            xs[i] = (xs[i] * ys[i]) + z;
        return xs;
    }

    static public float[] vdiv(float[] x, float[] ys){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] /= ys[i];
        return xs;
    }

    static public float[] vmul(float[] x, float[] ys){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] *= ys[i];
        return xs;
    }

    static public float[] vmuladd(float[] x, float[] ys, float[] zs){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = (xs[i] * ys[i]) + zs[i];
        return xs;
    }

    static public float[] vmulsub(float[] x, float[] ys, float[] zs){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = (xs[i] * ys[i]) - zs[i];
        return xs;
    }

    static public float[] vmax(float[] x, float[] ys){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = Math.max(xs[i], ys[i]);
        return xs;
    }

    static public float[] vmin(float[] x, float[] ys){
        final float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = Math.min(xs[i], ys[i]);
        return xs;
    }

    static public float[] vmap(IFn fn, float[] x) throws Exception{
        float[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)

```

```
        xs[i] = ((Number) fn.invoke(xs[i])).floatValue();
    return xs;
}

static public float[] vmap(IFn fn,
                           float[] x,
                           float[] ys) throws Exception{
    float[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = ((Number) fn.invoke(xs[i], ys[i])).floatValue();
    return xs;
}

static public class D{
    static public double add(double x, double y){
        return x + y;
    }

    static public double subtract(double x, double y){
        return x - y;
    }

    static public double negate(double x){
        return -x;
    }

    static public double inc(double x){
        return x + 1;
    }

    static public double dec(double x){
        return x - 1;
    }

    static public double multiply(double x, double y){
        return x * y;
    }

    static public double divide(double x, double y){
        return x / y;
    }

    static public boolean equiv(double x, double y){
        return x == y;
    }

    static public boolean lt(double x, double y){
        return x < y;
    }
}
```

```
}

static public boolean lte(double x, double y){
    return x <= y;
}

static public boolean gt(double x, double y){
    return x > y;
}

static public boolean gte(double x, double y){
    return x >= y;
}

static public boolean pos(double x){
    return x > 0;
}

static public boolean neg(double x){
    return x < 0;
}

static public boolean zero(double x){
    return x == 0;
}

static public double aget(double[] xs, int i){
    return xs[i];
}

static public double aset(double[] xs, int i, double v){
    xs[i] = v;
    return v;
}

static public int alength(double[] xs){
    return xs.length;
}

static public double[] aclone(double[] xs){
    return xs.clone();
}

static public double[] vec(int size, Object init){
    double[] ret = new double[size];
    if(init instanceof Number)
    {
        double f = ((Number) init).doubleValue();
        for(int i = 0; i < ret.length; i++)
            ret[i] = f;
    }
}
```

```

        }
    else
    {
        ISeq s = RT.seq(init);
        for(int i = 0; i < size && s != null; i++, s = s.rest())
            ret[i] = ((Number) s.first()).doubleValue();
    }
    return ret;
}

static public double[] vec(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new double[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = s.count();
        double[] ret = new double[size];
        for(int i = 0; i < size && s != null; i++, s = s.rest())
            ret[i] = ((Number) s.first()).intValue();
        return ret;
    }
}

static public double[] vsadd(double[] x, double y){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] += y;
    return xs;
}

static public double[] vssub(double[] x, double y){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] -= y;
    return xs;
}

static public double[] vsdiv(double[] x, double y){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] /= y;
    return xs;
}

static public double[] vsmul(double[] x, double y){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] *= y;
    return xs;
}

```

```
}

static public double[] svdiv(double y, double[] x){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = y / xs[i];
    return xs;
}

static public double[] vsmuladd(double[] x, double y, double[] zs){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y + zs[i];
    return xs;
}

static public double[] vsmulsub(double[] x, double y, double[] zs){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y - zs[i];
    return xs;
}

static public double[] vsmulsadd(double[] x, double y, double z){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y + z;
    return xs;
}

static public double[] vsmulssub(double[] x, double y, double z){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y - z;
    return xs;
}

static public double[] vabs(double[] x){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = Math.abs(xs[i]);
    return xs;
}

static public double[] vnegabs(double[] x){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = -Math.abs(xs[i]);
    return xs;
}
```

```
static public double[] vneg(double[] x){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = -xs[i];
    return xs;
}

static public double[] vsqr(double[] x){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] *= xs[i];
    return xs;
}

static public double[] vsignedsqr(double[] x){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] *= Math.abs(xs[i]);
    return xs;
}

static public double[] vclip(double[] x, double low, double high){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < low)
            xs[i] = low;
        else if(xs[i] > high)
            xs[i] = high;
    }
    return xs;
}

static public IPersistentVector vclipcounts(double[] x,
                                            double low,
                                            double high){
    final double[] xs = x.clone();
    int lowc = 0;
    int highc = 0;

    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < low)
        {
            ++lowc;
            xs[i] = low;
        }
        else if(xs[i] > high)
        {
```

```

        ++highc;
        xs[i] = high;
    }
}
return RT.vector(xs, lowc, highc);
}

static public double[] vthresh(double[] x,
                               double thresh,
                               double otherwise){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < thresh)
            xs[i] = otherwise;
    }
    return xs;
}

static public double[] vreverse(double[] x){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[xs.length - i - 1];
    return xs;
}

static public double[] vrrunningsum(double[] x){
    final double[] xs = x.clone();
    for(int i = 1; i < xs.length; i++)
        xs[i] = xs[i - 1] + xs[i];
    return xs;
}

static public double[] vsort(double[] x){
    final double[] xs = x.clone();
    Arrays.sort(xs);
    return xs;
}

static public double vdot(double[] xs, double[] ys){
    double ret = 0;
    for(int i = 0; i < xs.length; i++)
        ret += xs[i] * ys[i];
    return ret;
}

static public double vmax(double[] xs){
    if(xs.length == 0)
        return 0;
    double ret = xs[0];

```

```
    for(int i = 0; i < xs.length; i++)
        ret = Math.max(ret, xs[i]);
    return ret;
}

static public double vmin(double[] xs){
    if(xs.length == 0)
        return 0;
    double ret = xs[0];
    for(int i = 0; i < xs.length; i++)
        ret = Math.min(ret, xs[i]);
    return ret;
}

static public double vmean(double[] xs){
    if(xs.length == 0)
        return 0;
    return vsum(xs) / xs.length;
}

static public double vrms(double[] xs){
    if(xs.length == 0)
        return 0;
    double ret = 0;
    for(int i = 0; i < xs.length; i++)
        ret += xs[i] * xs[i];
    return Math.sqrt(ret / xs.length);
}

static public double vsum(double[] xs){
    double ret = 0;
    for(int i = 0; i < xs.length; i++)
        ret += xs[i];
    return ret;
}

static public boolean vequiv(double[] xs, double[] ys){
    return Arrays.equals(xs, ys);
}

static public double[] vadd(double[] x, double[] ys){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] += ys[i];
    return xs;
}

static public double[] vsub(double[] x, double[] ys){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
```

```

        xs[i] -= ys[i];
    return xs;
}

static public double[] vaddmul(double[] x,
                               double[] ys,
                               double[] zs){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] + ys[i]) * zs[i];
    return xs;
}

static public double[] vsubmul(double[] x, double[] ys, double[] zs){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] - ys[i]) * zs[i];
    return xs;
}

static public double[] vaddsmul(double[] x, double[] ys, double z){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] + ys[i]) * z;
    return xs;
}

static public double[] vsbsmulp(double[] x, double[] ys, double z){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] - ys[i]) * z;
    return xs;
}

static public double[] vmulsadd(double[] x, double[] ys, double z){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] * ys[i]) + z;
    return xs;
}

static public double[] vdiv(double[] x, double[] ys){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] /= ys[i];
    return xs;
}

static public double[] vmul(double[] x, double[] ys){
    final double[] xs = x.clone();

```

```
    for(int i = 0; i < xs.length; i++)
        xs[i] *= ys[i];
    return xs;
}

static public double[] vmuladd(double[] x, double[] ys, double[] zs){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] * ys[i]) + zs[i];
    return xs;
}

static public double[] vmulsub(double[] x, double[] ys, double[] zs){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] * ys[i]) - zs[i];
    return xs;
}

static public double[] vmax(double[] x, double[] ys){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = Math.max(xs[i], ys[i]);
    return xs;
}

static public double[] vmin(double[] x, double[] ys){
    final double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = Math.min(xs[i], ys[i]);
    return xs;
}

static public double[] vmap(IFn fn, double[] x) throws Exception{
    double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = ((Number) fn.invoke(xs[i])).doubleValue();
    return xs;
}

static public double[] vmap(IFn fn,
                           double[] x,
                           double[] ys) throws Exception{
    double[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = ((Number) fn.invoke(xs[i], ys[i])).doubleValue();
    return xs;
}
}
```

```
static public class I{
    static public int add(int x, int y){
        return x + y;
    }

    static public int subtract(int x, int y){
        return x - y;
    }

    static public int negate(int x){
        return -x;
    }

    static public int inc(int x){
        return x + 1;
    }

    static public int dec(int x){
        return x - 1;
    }

    static public int multiply(int x, int y){
        return x * y;
    }

    static public int divide(int x, int y){
        return x / y;
    }

    static public boolean equiv(int x, int y){
        return x == y;
    }

    static public boolean lt(int x, int y){
        return x < y;
    }

    static public boolean lte(int x, int y){
        return x <= y;
    }

    static public boolean gt(int x, int y){
        return x > y;
    }

    static public boolean gte(int x, int y){
        return x >= y;
    }

    static public boolean pos(int x){
```

```
        return x > 0;
    }

    static public boolean neg(int x){
        return x < 0;
    }

    static public boolean zero(int x){
        return x == 0;
    }

    static public int aget(int[] xs, int i){
        return xs[i];
    }

    static public int aset(int[] xs, int i, int v){
        xs[i] = v;
        return v;
    }

    static public int alength(int[] xs){
        return xs.length;
    }

    static public int[] aclone(int[] xs){
        return xs.clone();
    }

    static public int[] vec(int size, Object init){
        int[] ret = new int[size];
        if(init instanceof Number)
        {
            int f = ((Number) init).intValue();
            for(int i = 0; i < ret.length; i++)
                ret[i] = f;
        }
        else
        {
            ISeq s = RT.seq(init);
            for(int i = 0; i < size && s != null; i++, s = s.rest())
                ret[i] = ((Number) s.first()).intValue();
        }
        return ret;
    }

    static public int[] vec(Object sizeOrSeq){
        if(sizeOrSeq instanceof Number)
            return new int[((Number) sizeOrSeq).intValue()];
        else
    }
```

```

ISeq s = RT.seq(sizeOrSeq);
int size = s.count();
int[] ret = new int[size];
for(int i = 0; i < size && s != null; i++, s = s.rest())
    ret[i] = ((Number) s.first()).intValue();
return ret;
}

static public int[] vsadd(int[] x, int y){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] += y;
    return xs;
}

static public int[] vssub(int[] x, int y){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] -= y;
    return xs;
}

static public int[] vsdiv(int[] x, int y){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] /= y;
    return xs;
}

static public int[] vsmul(int[] x, int y){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] *= y;
    return xs;
}

static public int[] svdiv(int y, int[] x){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = y / xs[i];
    return xs;
}

static public int[] vsmuladd(int[] x, int y, int[] zs){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y + zs[i];
    return xs;
}

```

```
static public int[] vsmulsub(int[] x, int y, int[] zs){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y - zs[i];
    return xs;
}

static public int[] vsmulsadd(int[] x, int y, int z){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y + z;
    return xs;
}

static public int[] vsmulssub(int[] x, int y, int z){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y - z;
    return xs;
}

static public int[] vabs(int[] x){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = Math.abs(xs[i]);
    return xs;
}

static public int[] vnegabs(int[] x){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = -Math.abs(xs[i]);
    return xs;
}

static public int[] vneg(int[] x){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = -xs[i];
    return xs;
}

static public int[] vsqr(int[] x){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] *= xs[i];
    return xs;
}
```

```

static public int[] vsignedsqr(int[] x){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] *= Math.abs(xs[i]);
    return xs;
}

static public int[] vclip(int[] x, int low, int high){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < low)
            xs[i] = low;
        else if(xs[i] > high)
            xs[i] = high;
    }
    return xs;
}

static public IPersistentVector vclipcounts(int[] x,
                                            int low,
                                            int high){
    final int[] xs = x.clone();
    int lowc = 0;
    int highc = 0;

    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < low)
        {
            ++lowc;
            xs[i] = low;
        }
        else if(xs[i] > high)
        {
            ++highc;
            xs[i] = high;
        }
    }
    return RT.vector(xs, lowc, highc);
}

static public int[] vthresh(int[] x, int thresh, int otherwise){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < thresh)
            xs[i] = otherwise;
    }
    return xs;
}

```

```
}

static public int[] vreverse(int[] x){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[xs.length - i - 1];
    return xs;
}

static public int[] vrundingsum(int[] x){
    final int[] xs = x.clone();
    for(int i = 1; i < xs.length; i++)
        xs[i] = xs[i - 1] + xs[i];
    return xs;
}

static public int[] vsort(int[] x){
    final int[] xs = x.clone();
    Arrays.sort(xs);
    return xs;
}

static public int vdot(int[] xs, int[] ys){
    int ret = 0;
    for(int i = 0; i < xs.length; i++)
        ret += xs[i] * ys[i];
    return ret;
}

static public int vmax(int[] xs){
    if(xs.length == 0)
        return 0;
    int ret = xs[0];
    for(int i = 0; i < xs.length; i++)
        ret = Math.max(ret, xs[i]);
    return ret;
}

static public int vmin(int[] xs){
    if(xs.length == 0)
        return 0;
    int ret = xs[0];
    for(int i = 0; i < xs.length; i++)
        ret = Math.min(ret, xs[i]);
    return ret;
}

static public double vmean(int[] xs){
    if(xs.length == 0)
        return 0;
```

```
        return vsum(xs) / (double) xs.length;
    }

    static public double vrms(int[] xs){
        if(xs.length == 0)
            return 0;
        int ret = 0;
        for(int i = 0; i < xs.length; i++)
            ret += xs[i] * xs[i];
        return Math.sqrt(ret / (double) xs.length);
    }

    static public int vsum(int[] xs){
        int ret = 0;
        for(int i = 0; i < xs.length; i++)
            ret += xs[i];
        return ret;
    }

    static public boolean vequiv(int[] xs, int[] ys){
        return Arrays.equals(xs, ys);
    }

    static public int[] vadd(int[] x, int[] ys){
        final int[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] += ys[i];
        return xs;
    }

    static public int[] vsub(int[] x, int[] ys){
        final int[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] -= ys[i];
        return xs;
    }

    static public int[] vaddmul(int[] x, int[] ys, int[] zs){
        final int[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = (xs[i] + ys[i]) * zs[i];
        return xs;
    }

    static public int[] vsubmul(int[] x, int[] ys, int[] zs){
        final int[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = (xs[i] - ys[i]) * zs[i];
        return xs;
    }
```

```
static public int[] vaddsmul(int[] x, int[] ys, int z){  
    final int[] xs = x.clone();  
    for(int i = 0; i < xs.length; i++)  
        xs[i] = (xs[i] + ys[i]) * z;  
    return xs;  
}  
  
static public int[] vsubsmul(int[] x, int[] ys, int z){  
    final int[] xs = x.clone();  
    for(int i = 0; i < xs.length; i++)  
        xs[i] = (xs[i] - ys[i]) * z;  
    return xs;  
}  
  
static public int[] vmulsadd(int[] x, int[] ys, int z){  
    final int[] xs = x.clone();  
    for(int i = 0; i < xs.length; i++)  
        xs[i] = (xs[i] * ys[i]) + z;  
    return xs;  
}  
  
static public int[] vdiv(int[] x, int[] ys){  
    final int[] xs = x.clone();  
    for(int i = 0; i < xs.length; i++)  
        xs[i] /= ys[i];  
    return xs;  
}  
  
static public int[] vmul(int[] x, int[] ys){  
    final int[] xs = x.clone();  
    for(int i = 0; i < xs.length; i++)  
        xs[i] *= ys[i];  
    return xs;  
}  
  
static public int[] vmuladd(int[] x, int[] ys, int[] zs){  
    final int[] xs = x.clone();  
    for(int i = 0; i < xs.length; i++)  
        xs[i] = (xs[i] * ys[i]) + zs[i];  
    return xs;  
}  
  
static public int[] vmulsub(int[] x, int[] ys, int[] zs){  
    final int[] xs = x.clone();  
    for(int i = 0; i < xs.length; i++)  
        xs[i] = (xs[i] * ys[i]) - zs[i];  
    return xs;  
}
```

```
static public int[] vmax(int[] x, int[] ys){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = Math.max(xs[i], ys[i]);
    return xs;
}

static public int[] vmin(int[] x, int[] ys){
    final int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = Math.min(xs[i], ys[i]);
    return xs;
}

static public int[] vmap(IFn fn, int[] x) throws Exception{
    int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = ((Number) fn.invoke(xs[i])).intValue();
    return xs;
}

static public int[] vmap(IFn fn, int[] x, int[] ys) throws Exception{
    int[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = ((Number) fn.invoke(xs[i], ys[i])).intValue();
    return xs;
}

static public class L{
    static public long add(long x, long y){
        return x + y;
    }

    static public long subtract(long x, long y){
        return x - y;
    }

    static public long negate(long x){
        return -x;
    }

    static public long inc(long x){
        return x + 1;
    }

    static public long dec(long x){
        return x - 1;
    }
}
```

```
static public long multiply(long x, long y){
    return x * y;
}

static public long divide(long x, long y){
    return x / y;
}

static public boolean equiv(long x, long y){
    return x == y;
}

static public boolean lt(long x, long y){
    return x < y;
}

static public boolean lte(long x, long y){
    return x <= y;
}

static public boolean gt(long x, long y){
    return x > y;
}

static public boolean gte(long x, long y){
    return x >= y;
}

static public boolean pos(long x){
    return x > 0;
}

static public boolean neg(long x){
    return x < 0;
}

static public boolean zero(long x){
    return x == 0;
}

static public long aget(long[] xs, int i){
    return xs[i];
}

static public long aset(long[] xs, int i, long v){
    xs[i] = v;
    return v;
}
```

```

static public int alength(long[] xs){
    return xs.length;
}

static public long[] aclone(long[] xs){
    return xs.clone();
}

static public long[] vec(int size, Object init){
    long[] ret = new long[size];
    if(init instanceof Number)
    {
        long f = ((Number) init).longValue();
        for(int i = 0; i < ret.length; i++)
            ret[i] = f;
    }
    else
    {
        ISeq s = RT.seq(init);
        for(int i = 0; i < size && s != null; i++, s = s.rest())
            ret[i] = ((Number) s.first()).longValue();
    }
    return ret;
}

static public long[] vec(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new long[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = s.count();
        long[] ret = new long[size];
        for(int i = 0; i < size && s != null; i++, s = s.rest())
            ret[i] = ((Number) s.first()).intValue();
        return ret;
    }
}

static public long[] vsadd(long[] x, long y){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] += y;
    return xs;
}

static public long[] vssub(long[] x, long y){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)

```

```
        xs[i] -= y;
    return xs;
}

static public long[] vsdiv(long[] x, long y){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] /= y;
    return xs;
}

static public long[] vsmul(long[] x, long y){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] *= y;
    return xs;
}

static public long[] svdiv(long y, long[] x){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = y / xs[i];
    return xs;
}

static public long[] vsmuladd(long[] x, long y, long[] zs){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y + zs[i];
    return xs;
}

static public long[] vsmulsub(long[] x, long y, long[] zs){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y - zs[i];
    return xs;
}

static public long[] vsmulsadd(long[] x, long y, long z){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y + z;
    return xs;
}

static public long[] vsmulssub(long[] x, long y, long z){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[i] * y - z;
```

```
        return xs;
    }

    static public long[] vabs(long[] x){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = Math.abs(xs[i]);
        return xs;
    }

    static public long[] vnegabs(long[] x){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = -Math.abs(xs[i]);
        return xs;
    }

    static public long[] vneg(long[] x){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = -xs[i];
        return xs;
    }

    static public long[] vsqr(long[] x){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] *= xs[i];
        return xs;
    }

    static public long[] vsignedsqr(long[] x){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] *= Math.abs(xs[i]);
        return xs;
    }

    static public long[] vclip(long[] x, long low, long high){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
        {
            if(xs[i] < low)
                xs[i] = low;
            else if(xs[i] > high)
                xs[i] = high;
        }
        return xs;
    }
```

```

static public IPersistentVector vclipcounts(long[] x,
                                             long low,
                                             long high){
    final long[] xs = x.clone();
    int lowc = 0;
    int highc = 0;

    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < low)
        {
            ++lowc;
            xs[i] = low;
        }
        else if(xs[i] > high)
        {
            ++highc;
            xs[i] = high;
        }
    }
    return RT.vector(xs, lowc, highc);
}

static public long[] vthresh(long[] x, long thresh, long otherwise){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
    {
        if(xs[i] < thresh)
            xs[i] = otherwise;
    }
    return xs;
}

static public long[] vreverse(long[] x){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = xs[xs.length - i - 1];
    return xs;
}

static public long[] vrundingsum(long[] x){
    final long[] xs = x.clone();
    for(int i = 1; i < xs.length; i++)
        xs[i] = xs[i - 1] + xs[i];
    return xs;
}

static public long[] vsort(long[] x){
    final long[] xs = x.clone();
    Arrays.sort(xs);
}

```

```
        return xs;
    }

    static public long vdot(long[] xs, long[] ys){
        long ret = 0;
        for(int i = 0; i < xs.length; i++)
            ret += xs[i] * ys[i];
        return ret;
    }

    static public long vmax(long[] xs){
        if(xs.length == 0)
            return 0;
        long ret = xs[0];
        for(int i = 0; i < xs.length; i++)
            ret = Math.max(ret, xs[i]);
        return ret;
    }

    static public long vmin(long[] xs){
        if(xs.length == 0)
            return 0;
        long ret = xs[0];
        for(int i = 0; i < xs.length; i++)
            ret = Math.min(ret, xs[i]);
        return ret;
    }

    static public double vmean(long[] xs){
        if(xs.length == 0)
            return 0;
        return vsum(xs) / (double) xs.length;
    }

    static public double vrms(long[] xs){
        if(xs.length == 0)
            return 0;
        long ret = 0;
        for(int i = 0; i < xs.length; i++)
            ret += xs[i] * xs[i];
        return Math.sqrt(ret / (double) xs.length);
    }

    static public long vsum(long[] xs){
        long ret = 0;
        for(int i = 0; i < xs.length; i++)
            ret += xs[i];
        return ret;
    }
```

```
static public boolean vequiv(long[] xs, long[] ys){
    return Arrays.equals(xs, ys);
}

static public long[] vadd(long[] x, long[] ys){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] += ys[i];
    return xs;
}

static public long[] vsub(long[] x, long[] ys){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] -= ys[i];
    return xs;
}

static public long[] vaddmul(long[] x, long[] ys, long[] zs){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] + ys[i]) * zs[i];
    return xs;
}

static public long[] vsbmul(long[] x, long[] ys, long[] zs){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] - ys[i]) * zs[i];
    return xs;
}

static public long[] vaddsmul(long[] x, long[] ys, long z){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] + ys[i]) * z;
    return xs;
}

static public long[] vsubsmul(long[] x, long[] ys, long z){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] - ys[i]) * z;
    return xs;
}

static public long[] vmulsadd(long[] x, long[] ys, long z){
    final long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = (xs[i] * ys[i]) + z;
```

```

        return xs;
    }

    static public long[] vdiv(long[] x, long[] ys){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] /= ys[i];
        return xs;
    }

    static public long[] vmul(long[] x, long[] ys){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] *= ys[i];
        return xs;
    }

    static public long[] vmuladd(long[] x, long[] ys, long[] zs){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = (xs[i] * ys[i]) + zs[i];
        return xs;
    }

    static public long[] vmulsub(long[] x, long[] ys, long[] zs){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = (xs[i] * ys[i]) - zs[i];
        return xs;
    }

    static public long[] vmax(long[] x, long[] ys){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = Math.max(xs[i], ys[i]);
        return xs;
    }

    static public long[] vmin(long[] x, long[] ys){
        final long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = Math.min(xs[i], ys[i]);
        return xs;
    }

    static public long[] vmap(IFn fn, long[] x) throws Exception{
        long[] xs = x.clone();
        for(int i = 0; i < xs.length; i++)
            xs[i] = ((Number) fn.invoke(xs[i])).longValue();
        return xs;
    }
}

```

```
}

static public long[] vmap(IFn fn, long[] x, long[] ys)
throws Exception{
    long[] xs = x.clone();
    for(int i = 0; i < xs.length; i++)
        xs[i] = ((Number) fn.invoke(xs[i], ys[i])).longValue();
    return xs;
}

}

/*
//overload resolution
/**/

static public Number add(long x, Object y){
    return add((Object)x,y);
}

static public Number add(Object x, long y){
    return add(x,(Object)y);
}

static public Number addP(long x, Object y){
    return addP((Object)x,y);
}

static public Number addP(Object x, long y){
    return addP(x,(Object)y);
}

static public double add(double x, Object y){
    return add(x,((Number)y).doubleValue());
}

static public double add(Object x, double y){
    return add(((Number)x).doubleValue(),y);
}

static public double add(double x, long y){
    return x + y;
}

static public double add(long x, double y){
    return x + y;
}

static public double addP(double x, Object y){
```

```
        return addP(x,((Number)y).doubleValue());
    }

    static public double addP(Object x, double y){
        return addP(((Number)x).doubleValue(),y);
    }

    static public double addP(double x, long y){
        return x + y;
    }

    static public double addP(long x, double y){
        return x + y;
    }

    static public Number minus(long x, Object y){
        return minus((Object)x,y);
    }

    static public Number minus(Object x, long y){
        return minus(x,(Object)y);
    }

    static public Number minusP(long x, Object y){
        return minusP((Object)x,y);
    }

    static public Number minusP(Object x, long y){
        return minusP(x,(Object)y);
    }

    static public double minus(double x, Object y){
        return minus(x,((Number)y).doubleValue());
    }

    static public double minus(Object x, double y){
        return minus(((Number)x).doubleValue(),y);
    }

    static public double minus(double x, long y){
        return x - y;
    }

    static public double minus(long x, double y){
        return x - y;
    }

    static public double minusP(double x, Object y){
        return minus(x,((Number)y).doubleValue());
    }
```

```
static public double minusP(Object x, double y){
    return minus(((Number)x).doubleValue(),y);
}

static public double minusP(double x, long y){
    return x - y;
}

static public double minusP(long x, double y){
    return x - y;
}

static public Number multiply(long x, Object y){
    return multiply((Object)x,y);
}

static public Number multiply(Object x, long y){
    return multiply(x,(Object)y);
}

static public Number multiplyP(long x, Object y){
    return multiplyP((Object)x,y);
}

static public Number multiplyP(Object x, long y){
    return multiplyP(x,(Object)y);
}

static public double multiply(double x, Object y){
    return multiply(x,((Number)y).doubleValue());
}

static public double multiply(Object x, double y){
    return multiply(((Number)x).doubleValue(),y);
}

static public double multiply(double x, long y){
    return x * y;
}

static public double multiply(long x, double y){
    return x * y;
}

static public double multiplyP(double x, Object y){
    return multiplyP(x,((Number)y).doubleValue());
}

static public double multiplyP(Object x, double y){
```

```
        return multiplyP((Number)x).doubleValue(),y);
    }

    static public double multiplyP(double x, long y){
        return x * y;
    }

    static public double multiplyP(long x, double y){
        return x * y;
    }

    static public Number divide(long x, Object y){
        return divide((Object)x,y);
    }

    static public Number divide(Object x, long y){
        return divide(x,(Object)y);
    }

    static public double divide(double x, Object y){
        return x / ((Number)y).doubleValue();
    }

    static public double divide(Object x, double y){
        return ((Number)x).doubleValue() / y;
    }

    static public double divide(double x, long y){
        return x / y;
    }

    static public double divide(long x, double y){
        return x / y;
    }

    static public boolean lt(long x, Object y){
        return lt((Object)x,y);
    }

    static public boolean lt(Object x, long y){
        return lt(x,(Object)y);
    }

    static public boolean lt(double x, Object y){
        return x < ((Number)y).doubleValue();
    }

    static public boolean lt(Object x, double y){
        return ((Number)x).doubleValue() < y;
    }
```

```
static public boolean lt(double x, long y){
    return x < y;
}

static public boolean lt(long x, double y){
    return x < y;
}

static public boolean lte(long x, Object y){
    return lte((Object)x,y);
}

static public boolean lte(Object x, long y){
    return lte(x,(Object)y);
}

static public boolean lte(double x, Object y){
    return x <= ((Number)y).doubleValue();
}

static public boolean lte(Object x, double y){
    return ((Number)x).doubleValue() <= y;
}

static public boolean lte(double x, long y){
    return x <= y;
}

static public boolean lte(long x, double y){
    return x <= y;
}

static public boolean gt(long x, Object y){
    return gt((Object)x,y);
}

static public boolean gt(Object x, long y){
    return gt(x,(Object)y);
}

static public boolean gt(double x, Object y){
    return x > ((Number)y).doubleValue();
}

static public boolean gt(Object x, double y){
    return ((Number)x).doubleValue() > y;
}

static public boolean gt(double x, long y){
```

```
        return x > y;
    }

    static public boolean gt(long x, double y){
        return x > y;
    }

    static public boolean gte(long x, Object y){
        return gte((Object)x,y);
    }

    static public boolean gte(Object x, long y){
        return gte(x,(Object)y);
    }

    static public boolean gte(double x, Object y){
        return x >= ((Number)y).doubleValue();
    }

    static public boolean gte(Object x, double y){
        return ((Number)x).doubleValue() >= y;
    }

    static public boolean gte(double x, long y){
        return x >= y;
    }

    static public boolean gte(long x, double y){
        return x >= y;
    }

    static public boolean equiv(long x, Object y){
        return equiv((Object)x,y);
    }

    static public boolean equiv(Object x, long y){
        return equiv(x,(Object)y);
    }

    static public boolean equiv(double x, Object y){
        return x == ((Number)y).doubleValue();
    }

    static public boolean equiv(Object x, double y){
        return ((Number)x).doubleValue() == y;
    }

    static public boolean equiv(double x, long y){
        return x == y;
    }
```

```
static public boolean equiv(long x, double y){  
    return x == y;  
}  
  
}
```

—————

9.77 Obj.java

(IObj [800]) (Serializable [1723])
— Obj.java —

```
/*  
\getchunk{Clojure Copyright}  
*/  
/* rich Mar 25, 2006 3:44:58 PM */  
  
package clojure.lang;  
  
import java.io.Serializable;  
  
public abstract class Obj implements IObj, Serializable {  
  
    final IPersistentMap _meta;  
  
    public Obj(IPersistentMap meta){  
        this._meta = meta;  
    }  
  
    public Obj(){  
        _meta = null;  
    }  
  
    final public IPersistentMap meta(){  
        return _meta;  
    }  
  
    abstract public Obj withMeta(IPersistentMap meta);  
}
```

—————

9.78 PersistentArrayMap.java

```
(IObj [800]) (IEitableCollection [774])
— PersistentArrayMap.java —

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.io.Serializable;
import java.util.Arrays;
import java.util.Iterator;
import java.util.Map;

/**
 * Simple implementation of persistent map on an array
 * <p/>
 * Note that instances of this class are constant values
 * i.e. add/remove etc return new values
 * <p/>
 * Copies array on every change, so only appropriate for
 * _very_small_ maps
 * <p/>
 * null keys and values are ok, but you won't be able to distinguish
 * a null value via valAt - use contains/entryAt
 */

public class PersistentArrayMap
    extends APersistentMap implements IObj, IEitableCollection {

    final Object[] array;
    static final int HASHTABLE_THRESHOLD = 16;

    public static final PersistentArrayMap EMPTY = new PersistentArrayMap();
    private final IPersistentMap _meta;

    static public IPersistentMap create(Map other) {
        ITransientMap ret = EMPTY.asTransient();
        for(Object o : other.entrySet())
        {
            Map.Entry e = (Entry) o;
            ret = ret.assoc(e.getKey(), e.getValue());
        }
        return ret.persistent();
    }

    protected PersistentArrayMap(){
        this.array = new Object[] {};
    }
}
```

```
        this._meta = null;
    }

    public PersistentArrayMap withMeta(IPersistentMap meta){
        return new PersistentArrayMap(meta, array);
    }

    PersistentArrayMap create(Object... init){
        return new PersistentArrayMap(meta(), init);
    }

    IPersistentMap createHT(Object[] init){
        return PersistentHashMap.create(meta(), init);
    }

    static public PersistentArrayMap createWithCheck(Object[] init){
        for(int i=0;i< init.length;i += 2)
        {
            for(int j=i+2;j<init.length;j += 2)
            {
                if(equalKey(init[i],init[j]))
                    throw new IllegalArgumentException("Duplicate key: " + init[i]);
            }
        }
        return new PersistentArrayMap(init);
    }

    /**
     * This ctor captures/aliases the passed array, so do not modify later
     *
     * @param init {key1,val1,key2,val2,...}
     */
    public PersistentArrayMap(Object[] init){
        this.array = init;
        this._meta = null;
    }

    public PersistentArrayMap(IPersistentMap meta, Object[] init){
        this._meta = meta;
        this.array = init;
    }

    public int count(){
        return array.length / 2;
    }

    public boolean containsKey(Object key){
        return indexOf(key) >= 0;
    }
```

```

public IMapEntry entryAt(Object key){
    int i = indexOf(key);
    if(i >= 0)
        return new MapEntry(array[i],array[i+1]);
    return null;
}

public IPersistentMap assocEx(Object key, Object val) throws Exception{
    int i = indexOf(key);
    Object[] newArray;
    if(i >= 0)
    {
        throw new Exception("Key already present");
    }
    else //didn't have key, grow
    {
        if(array.length > HASHTABLE_THRESHOLD)
            return createHT(array).assocEx(key, val);
        newArray = new Object[array.length + 2];
        if(array.length > 0)
            System.arraycopy(array, 0, newArray, 2, array.length);
        newArray[0] = key;
        newArray[1] = val;
    }
    return create(newArray);
}

public IPersistentMap assoc(Object key, Object val){
    int i = indexOf(key);
    Object[] newArray;
    if(i >= 0) //already have key, same-sized replacement
    {
        if(array[i + 1] == val) //no change, no op
            return this;
        newArray = array.clone();
        newArray[i + 1] = val;
    }
    else //didn't have key, grow
    {
        if(array.length > HASHTABLE_THRESHOLD)
            return createHT(array).assoc(key, val);
        newArray = new Object[array.length + 2];
        if(array.length > 0)
            System.arraycopy(array, 0, newArray, 2, array.length);
        newArray[0] = key;
        newArray[1] = val;
    }
    return create(newArray);
}

```

```

public IPersistentMap without(Object key){
    int i = indexOf(key);
    if(i >= 0) //have key, will remove
    {
        int newlen = array.length - 2;
        if(newlen == 0)
            return empty();
        Object[] newArray = new Object[newlen];
        for(int s = 0, d = 0; s < array.length; s += 2)
        {
            if(!equalKey(array[s], key)) //skip removal key
            {
                newArray[d] = array[s];
                newArray[d + 1] = array[s + 1];
                d += 2;
            }
        }
        return create(newArray);
    }
    //don't have key, no op
    return this;
}

public IPersistentMap empty(){
    return (IPersistentMap) EMPTY.withMeta(meta());
}

final public Object valAt(Object key, Object notFound){
    int i = indexOf(key);
    if(i >= 0)
        return array[i + 1];
    return notFound;
}

public Object valAt(Object key){
    return valAt(key, null);
}

public int capacity(){
    return count();
}

private int indexOf(Object key){
    for(int i = 0; i < array.length; i += 2)
    {
        if(equalKey(array[i], key))
            return i;
    }
    return -1;
}

```

```
static boolean equalKey(Object k1, Object k2){
    return Util.equiv(k1, k2);
}

public Iterator iterator(){
    return new Iter(array);
}

public ISeq seq(){
    if(array.length > 0)
        return new Seq(array, 0);
    return null;
}

public IPersistentMap meta(){
    return _meta;
}

static class Seq extends ASeq implements Counted{
    final Object[] array;
    final int i;

    Seq(Object[] array, int i){
        this.array = array;
        this.i = i;
    }

    public Seq(IPersistentMap meta, Object[] array, int i){
        super(meta);
        this.array = array;
        this.i = i;
    }

    public Object first(){
        return new MapEntry(array[i],array[i+1]);
    }

    public ISeq next(){
        if(i + 2 < array.length)
            return new Seq(array, i + 2);
        return null;
    }

    public int count(){
        return (array.length - i) / 2;
    }

    public Obj withMeta(IPersistentMap meta){
        return new Seq(meta, array, i);
    }
}
```

```

        }
    }

static class Iter implements Iterator{
    Object[] array;
    int i;

    //for iterator
    Iter(Object[] array){
        this(array, -2);
    }

    //for entryAt
    Iter(Object[] array, int i){
        this.array = array;
        this.i = i;
    }

    public boolean hasNext(){
        return i < array.length - 2;
    }

    public Object next(){
        i += 2;
        return new MapEntry(array[i],array[i+1]);
    }

    public void remove(){
        throw new UnsupportedOperationException();
    }
}

public ITransientMap asTransient(){
    return new TransientArrayMap(array);
}

static final class TransientArrayMap extends ATransientMap {
    int len;
    final Object[] array;
    Thread owner;

    public TransientArrayMap(Object[] array){
        this.owner = Thread.currentThread();
        this.array =
            new Object[Math.max(HASHTABLE_THRESHOLD, array.length)];
        System.arraycopy(array, 0, this.array, 0, array.length);
        this.len = array.length;
    }
}

```

```

private int indexOf(Object key){
    for(int i = 0; i < len; i += 2)
    {
        if(equalKey(array[i], key))
            return i;
    }
    return -1;
}

ITransientMap doAssoc(Object key, Object val){
    int i = indexOf(key);
    if(i >= 0) //already have key,
    {
        if(array[i + 1] != val) //no change, no op
            array[i + 1] = val;
    }
    else //didn't have key, grow
    {
        if(len >= array.length)
            return
                PersistentHashMap.create(array)
                    .asTransient()
                    .assoc(key, val);
        array[len++] = key;
        array[len++] = val;
    }
    return this;
}

ITransientMap doWithout(Object key) {
    int i = indexOf(key);
    if(i >= 0) //have key, will remove
    {
        if (len >= 2)
        {
            array[i] = array[len - 2];
            array[i + 1] = array[len - 1];
        }
        len -= 2;
    }
    return this;
}

Object doValAt(Object key, Object notFound) {
    int i = indexOf(key);
    if (i >= 0)
        return array[i + 1];
    return notFound;
}

```

```

int doCount() {
    return len / 2;
}

IPersistentMap doPersistent(){
    ensureEditable();
    owner = null;
    Object[] a = new Object[len];
    System.arraycopy(array,0,a,0,len);
    return new PersistentArrayMap(a);
}

void ensureEditable(){
    if(owner == Thread.currentThread())
        return;
    if(owner != null)
        throw new IllegalAccessError(
            "Transient used by non-owner thread");
    throw new IllegalAccessError(
        "Transient used after persistent! call");
}
}
}

```

9.79 PersistentHashMap.java

(INode [58])
— PersistentHashMap ArrayNode class —

```

final static class ArrayNode implements INode{
    int count;
    final INode[] array;
    final AtomicReference<Thread> edit;

    ArrayNode(AtomicReference<Thread> edit, int count, INode[] array){
        this.array = array;
        this.edit = edit;
        this.count = count;
    }

    public INode assoc(int shift, int hash, Object key,
                      Object val, Box addedLeaf){
        int idx = mask(hash, shift);
        INode node = array[idx];
        if(node == null)
            return

```

```

        new ArrayNode(null, count + 1,
                      cloneAndSet(array,
                                  idx,
                                  BitmapIndexedNode.EMPTY
                                  .assoc(shift + 5,
                                         hash,
                                         key,
                                         val,
                                         addedLeaf)));
    INode n = node.assoc(shift + 5, hash, key, val, addedLeaf);
    if(n == node)
        return this;
    return new ArrayNode(null, count, cloneAndSet(array, idx, n));
}

public INode without(int shift, int hash, Object key){
    int idx = mask(hash, shift);
    INode node = array[idx];
    if(node == null)
        return this;
    INode n = node.without(shift + 5, hash, key);
    if(n == node)
        return this;
    if (n == null) {
        if (count <= 8) // shrink
            return pack(null, idx);
        return
            new ArrayNode(null, count - 1,
                          cloneAndSet(array, idx, n));
    } else
        return
            new ArrayNode(null, count,
                          cloneAndSet(array, idx, n));
}

public IMapEntry find(int shift, int hash, Object key){
    int idx = mask(hash, shift);
    INode node = array[idx];
    if(node == null)
        return null;
    return node.find(shift + 5, hash, key);
}

\getchunk{ArrayNode find Object method}

public ISeq nodeSeq(){
    return Seq.create(array);
}

private ArrayNode ensureEditable(AtomicReference<Thread> edit){

```

```

        if(this.edit == edit)
            return this;
        return new ArrayNode(edit, count, this.array.clone());
    }

    private ArrayNode editAndSet(AtomicReference<Thread> edit,
                                int i, INode n){
        ArrayNode editable = ensureEditable(edit);
        editable.array[i] = n;
        return editable;
    }

    private INode pack(AtomicReference<Thread> edit, int idx) {
        Object[] newArray = new Object[2*(count - 1)];
        int j = 1;
        int bitmap = 0;
        for(int i = 0; i < idx; i++) {
            if (array[i] != null) {
                newArray[j] = array[i];
                bitmap |= 1 << i;
                j += 2;
            }
        }
        for(int i = idx + 1; i < array.length; i++)
            if (array[i] != null) {
                newArray[j] = array[i];
                bitmap |= 1 << i;
                j += 2;
            }
        return new BitmapIndexedNode(edit, bitmap, newArray);
    }

    public INode assoc(AtomicReference<Thread> edit, int shift,
                      int hash, Object key, Object val,
                      Box addedLeaf){
        int idx = mask(hash, shift);
        INode node = array[idx];
        if(node == null) {
            ArrayNode editable =
                editAndSet(edit, idx,
                           BitmapIndexedNode.EMPTY
                           .assoc(edit,
                                  shift + 5,
                                  hash,
                                  key,
                                  val,
                                  addedLeaf));
            editable.count++;
            return editable;
        }
    }
}

```

```

INode n = node.assoc(edit, shift + 5, hash, key, val, addedLeaf);
if(n == node)
    return this;
return editAndSet(edit, idx, n);
}

public INode without(AtomicReference<Thread> edit,
                     int shift,
                     int hash,
                     Object key,
                     Box removedLeaf){
    int idx = mask(hash, shift);
    INode node = array[idx];
    if(node == null)
        return this;
    INode n = node.without(edit, shift + 5, hash, key, removedLeaf);
    if(n == node)
        return this;
    if(n == null) {
        if (count <= 8) // shrink
            return pack(edit, idx);
        ArrayNode editable = editAndSet(edit, idx, n);
        editable.count--;
        return editable;
    }
    return editAndSet(edit, idx, n);
}

static class Seq extends ASeq {
    final INode[] nodes;
    final int i;
    final ISeq s;

    static ISeq create(INode[] nodes) {
        return create(null, nodes, 0, null);
    }

    private static ISeq create(IPersistentMap meta,
                               INode[] nodes, int i, ISeq s) {
        if (s != null)
            return new Seq(meta, nodes, i, s);
        for(int j = i; j < nodes.length; j++)
            if (nodes[j] != null) {
                ISeq ns = nodes[j].nodeSeq();
                if (ns != null)
                    return new Seq(meta, nodes, j + 1, ns);
            }
        return null;
    }
}

```

```

private Seq(IPersistentMap meta, INode[] nodes, int i, ISeq s) {
    super(meta);
    this.nodes = nodes;
    this.i = i;
    this.s = s;
}

public Obj withMeta(IPersistentMap meta) {
    return new Seq(meta, nodes, i, s);
}

public Object first() {
    return s.first();
}

public ISeq next() {
    return create(null, nodes, i, s.next());
}

}

}

```

(INode [58])

— PersistentHashMap BitmapIndexedNode class —

```

final static class BitmapIndexedNode implements INode{
    static final BitmapIndexedNode EMPTY =
        new BitmapIndexedNode(null, 0, new Object[0]);

    int bitmap;
    Object[] array;
    final AtomicReference<Thread> edit;

    final int index(int bit){
        return Integer.bitCount(bitmap & (bit - 1));
    }

    BitmapIndexedNode(AtomicReference<Thread> edit,
                      int bitmap, Object[] array){
        this.bitmap = bitmap;
        this.array = array;
        this.edit = edit;
    }

    public INode assoc(int shift, int hash, Object key,
                      Object val, Box addedLeaf){
        int bit = bitpos(hash, shift);
        int idx = index(bit);

```

```

if((bitmap & bit) != 0) {
    Object keyOrNull = array[2*idx];
    Object valOrNode = array[2*idx+1];
    if(keyOrNull == null) {
        INode n = ((INode) valOrNode).assoc(shift + 5,
                                              hash,
                                              key,
                                              val,
                                              addedLeaf);
        if(n == valOrNode)
            return this;
        return
            new BitmapIndexedNode(null, bitmap,
                                  cloneAndSet(array, 2*idx+1, n));
    }
    if(Util.equiv(key, keyOrNull)) {
        if(val == valOrNode)
            return this;
        return
            new BitmapIndexedNode(null, bitmap,
                                  cloneAndSet(array,
                                              2*idx+1,
                                              val));
    }
    addedLeaf.val = addedLeaf;
    return new BitmapIndexedNode(null, bitmap,
                                 cloneAndSet(array,
                                             2*idx, null,
                                             2*idx+1, createNode(shift + 5,
                                                               keyOrNull,
                                                               valOrNode,
                                                               hash,
                                                               key,
                                                               val)));
} else {
    int n = Integer.bitCount(bitmap);
    if(n >= 16) {
        INode[] nodes = new INode[32];
        int jdx = mask(hash, shift);
        nodes[jdx] = EMPTY.assoc(shift + 5,
                                  hash,
                                  key,
                                  val,
                                  addedLeaf);
        int j = 0;
        for(int i = 0; i < 32; i++)
            if(((bitmap >>> i) & 1) != 0) {
                if (array[j] == null)
                    nodes[i] = (INode) array[j+1];
                else

```

```

        nodes[i] =
            EMPTY.assoc(shift + 5,
                         Util.hash(array[j]),
                         array[j],
                         array[j+1],
                         addedLeaf);
        j += 2;
    }
    return new ArrayNode(null, n + 1, nodes);
} else {
    Object[] newArray = new Object[2*(n+1)];
    System.arraycopy(array, 0, newArray, 0, 2*idx);
    newArray[2*idx] = key;
    addedLeaf.val = addedLeaf;
    newArray[2*idx+1] = val;
    System.arraycopy(array, 2*idx, newArray,
                     2*(idx+1), 2*(n-idx));
    return
        new BitmapIndexedNode(null, bitmap | bit, newArray);
}
}

public INode without(int shift, int hash, Object key){
    int bit = bitpos(hash, shift);
    if((bitmap & bit) == 0)
        return this;
    int idx = index(bit);
    Object keyOrNull = array[2*idx];
    Object valOrNode = array[2*idx+1];
    if(keyOrNull == null) {
        INode n =
            ((INode) valOrNode).without(shift + 5, hash, key);
        if (n == valOrNode)
            return this;
        if (n != null)
            return
                new BitmapIndexedNode(null, bitmap,
                                      cloneAndSet(array, 2*idx+1, n));
        if (bitmap == bit)
            return null;
        return
            new BitmapIndexedNode(null, bitmap ^ bit,
                                  removePair(array, idx));
    }
    if(Util.equiv(key, keyOrNull))
        // TODO: collapse
        return new BitmapIndexedNode(null,
                                     bitmap ^ bit,
                                     removePair(array, idx));
}

```

```

        return this;
    }

    public IMapEntry find(int shift, int hash, Object key){
        int bit = bitpos(hash, shift);
        if((bitmap & bit) == 0)
            return null;
        int idx = index(bit);
        Object keyOrNull = array[2*idx];
        Object valOrNode = array[2*idx+1];
        if(keyOrNull == null)
            return ((INode) valOrNode).find(shift + 5, hash, key);
        if(Util.equiv(key, keyOrNull))
            return new MapEntry(keyOrNull, valOrNode);
        return null;
    }

    public Object find(int shift, int hash,
                      Object key, Object notFound){
        int bit = bitpos(hash, shift);
        if((bitmap & bit) == 0)
            return notFound;
        int idx = index(bit);
        Object keyOrNull = array[2*idx];
        Object valOrNode = array[2*idx+1];
        if(keyOrNull == null)
            return
                ((INode) valOrNode).find(shift + 5, hash, key, notFound);
        if(Util.equiv(key, keyOrNull))
            return valOrNode;
        return notFound;
    }

    public ISeq nodeSeq(){
        return NodeSeq.create(array);
    }

    private BitmapIndexedNode
    ensureEditable(AtomicReference<Thread> edit){
        if(this.edit == edit)
            return this;
        int n = Integer.bitCount(bitmap);
        Object[] newArray =
            new Object[n >= 0 ? 2*(n+1) : 4]; // make room for next assoc
        System.arraycopy(array, 0, newArray, 0, 2*n);
        return new BitmapIndexedNode(edit, bitmap, newArray);
    }

    private BitmapIndexedNode
    editAndSet(AtomicReference<Thread> edit, int i, Object a) {

```

```

        BitmapIndexedNode editable = ensureEditable(edit);
        editable.array[i] = a;
        return editable;
    }

    private BitmapIndexedNode
    editAndSet(AtomicReference<Thread> edit, int i,
               Object a, int j, Object b) {
        BitmapIndexedNode editable = ensureEditable(edit);
        editable.array[i] = a;
        editable.array[j] = b;
        return editable;
    }

    private BitmapIndexedNode
    editAndRemovePair(AtomicReference<Thread> edit,
                      int bit, int i) {
        if (bitmap == bit)
            return null;
        BitmapIndexedNode editable = ensureEditable(edit);
        editable.bitmap ^= bit;
        System.arraycopy(editable.array, 2*(i+1), editable.array,
                        2*i, editable.array.length - 2*(i+1));
        editable.array[editable.array.length - 2] = null;
        editable.array[editable.array.length - 1] = null;
        return editable;
    }

    public INode assoc(AtomicReference<Thread> edit, int shift,
                       int hash, Object key, Object val,
                       Box addedLeaf){
        int bit = bitpos(hash, shift);
        int idx = index(bit);
        if((bitmap & bit) != 0) {
            Object keyOrNull = array[2*idx];
            Object valOrNode = array[2*idx+1];
            if(keyOrNull == null) {
                INode n =
                    ((INode) valOrNode).assoc(edit,
                                              shift + 5,
                                              hash,
                                              key,
                                              val,
                                              addedLeaf);
                if(n == valOrNode)
                    return this;
                return editAndSet(edit, 2*idx+1, n);
            }
            if(Util.equiv(key, keyOrNull)) {
                if(val == valOrNode)

```

```

        return this;
        return editAndSet(edit, 2*idx+1, val);
    }
addedLeaf.val = addedLeaf;
return editAndSet(edit, 2*idx, null, 2*idx+1,
                 createNode(edit, shift + 5, keyOrNull,
                             valOrNode, hash, key, val));
} else {
    int n = Integer.bitCount(bitmap);
    if(n*2 < array.length) {
        addedLeaf.val = addedLeaf;
        BitmapIndexedNode editable = ensureEditable(edit);
        System.arraycopy(editable.array, 2*idx,
                        editable.array, 2*(idx+1),
                        2*(n-idx));
        editable.array[2*idx] = key;
        editable.array[2*idx+1] = val;
        editable.bitmap |= bit;
        return editable;
    }
    if(n >= 16) {
        INode[] nodes = new INode[32];
        int jdx = mask(hash, shift);
        nodes[jdx] = EMPTY.assoc(edit, shift + 5, hash, key,
                                 val, addedLeaf);
        int j = 0;
        for(int i = 0; i < 32; i++) {
            if(((bitmap >>> i) & 1) != 0) {
                if (array[j] == null)
                    nodes[i] = (INode) array[j+1];
                else
                    nodes[i] =
                        EMPTY.assoc(edit,
                                    shift + 5,
                                    Util.hash(array[j]),
                                    array[j],
                                    array[j+1],
                                    addedLeaf);
                j += 2;
            }
        }
        return new ArrayNode(edit, n + 1, nodes);
    } else {
        Object[] newArray = new Object[2*(n+4)];
        System.arraycopy(array, 0, newArray, 0, 2*idx);
        newArray[2*idx] = key;
        addedLeaf.val = addedLeaf;
        newArray[2*idx+1] = val;
        System.arraycopy(array, 2*idx, newArray,
                        2*(idx+1), 2*(n-idx));
        BitmapIndexedNode editable = ensureEditable(edit);
    }
}

```

```

        editable.array = newArray;
        editable.bitmap |= bit;
        return editable;
    }
}
}

public INode without(AtomicReference<Thread> edit, int shift,
                     int hash, Object key, Box removedLeaf){
    int bit = bitpos(hash, shift);
    if((bitmap & bit) == 0)
        return this;
    int idx = index(bit);
    Object keyOrNull = array[2*idx];
    Object valOrNode = array[2*idx+1];
    if(keyOrNull == null) {
        INode n =
            ((INode) valOrNode).without(edit,
                                         shift + 5,
                                         hash,
                                         key,
                                         removedLeaf);
        if (n == valOrNode)
            return this;
        if (n != null)
            return editAndSet(edit, 2*idx+1, n);
        if (bitmap == bit)
            return null;
        removedLeaf.val = removedLeaf;
        return editAndRemovePair(edit, bit, idx);
    }
    if(Util.equiv(key, keyOrNull)) {
        removedLeaf.val = removedLeaf;
        // TODO: collapse
        return editAndRemovePair(edit, bit, idx);
    }
    return this;
}
}

```

(INode [58])

— PersistentHashMap HashCollisionNode class —

```

final static class HashCollisionNode implements INode{

    final int hash;
    int count;
    Object[] array;
}

```

```

final AtomicReference<Thread> edit;

HashCollisionNode(AtomicReference<Thread> edit, int hash,
                  int count, Object... array){
    this.edit = edit;
    this.hash = hash;
    this.count = count;
    this.array = array;
}

public INode assoc(int shift, int hash, Object key,
                   Object val, Box addedLeaf){
    if(hash == this.hash) {
        int idx = findIndex(key);
        if(idx != -1) {
            if(array[idx + 1] == val)
                return this;
            return
                new HashCollisionNode(null, hash, count,
                                      cloneAndSet(array,
                                                  idx + 1,
                                                  val));
        }
        Object[] newArray = new Object[array.length + 2];
        System.arraycopy(array, 0, newArray, 0, array.length);
        newArray[array.length] = key;
        newArray[array.length + 1] = val;
        addedLeaf.val = addedLeaf;
        return
            new HashCollisionNode(edit, hash, count + 1, newArray);
    }
    // nest it in a bitmap node
    return
        new BitmapIndexedNode(null, bitpos(this.hash, shift),
                              new Object[] {null, this})
            .assoc(shift, hash, key, val, addedLeaf);
}

public INode without(int shift, int hash, Object key){
    int idx = findIndex(key);
    if(idx == -1)
        return this;
    if(count == 1)
        return null;
    return
        new HashCollisionNode(null, hash, count - 1,
                              removePair(array, idx/2));
}

public IMapEntry find(int shift, int hash, Object key){

```

```
int idx = findIndex(key);
if(idx < 0)
    return null;
if(Util.equiv(key, array[idx]))
    return new MapEntry(array[idx], array[idx+1]);
return null;
}

public Object find(int shift, int hash, Object key, Object notFound){
    int idx = findIndex(key);
    if(idx < 0)
        return notFound;
    if(Util.equiv(key, array[idx]))
        return array[idx+1];
    return notFound;
}

public ISeq nodeSeq(){
    return NodeSeq.create(array);
}

public int findIndex(Object key){
    for(int i = 0; i < 2*count; i+=2)
    {
        if(Util.equiv(key, array[i]))
            return i;
    }
    return -1;
}

private HashCollisionNode
ensureEditable(AtomicReference<Thread> edit){
    if(this.edit == edit)
        return this;
    return new HashCollisionNode(edit, hash, count, array);
}

private HashCollisionNode
ensureEditable(AtomicReference<Thread> edit, int count,
Object[] array){
    if(this.edit == edit) {
        this.array = array;
        this.count = count;
        return this;
    }
    return new HashCollisionNode(edit, hash, count, array);
}

private HashCollisionNode
editAndSet(AtomicReference<Thread> edit, int i, Object a) {
```

```

HashCollisionNode editable = ensureEditable(edit);
editable.array[i] = a;
return editable;
}

private HashCollisionNode
editAndSet(AtomicReference<Thread> edit, int i, Object a,
           int j, Object b) {
    HashCollisionNode editable = ensureEditable(edit);
    editable.array[i] = a;
    editable.array[j] = b;
    return editable;
}

public INode assoc(AtomicReference<Thread> edit, int shift,
                   int hash, Object key, Object val,
                   Box addedLeaf){
    if(hash == this.hash) {
        int idx = findIndex(key);
        if(idx != -1) {
            if(array[idx + 1] == val)
                return this;
            return editAndSet(edit, idx+1, val);
        }
        if (array.length > 2*count) {
            addedLeaf.val = addedLeaf;
            HashCollisionNode editable =
                editAndSet(edit, 2*count, key, 2*count+1, val);
            editable.count++;
            return editable;
        }
        Object[] newArray = new Object[array.length + 2];
        System.arraycopy(array, 0, newArray, 0, array.length);
        newArray[array.length] = key;
        newArray[array.length + 1] = val;
        addedLeaf.val = addedLeaf;
        return ensureEditable(edit, count + 1, newArray);
    }
    // nest it in a bitmap node
    return
        new BitmapIndexedNode(edit,
                             bitpos(this.hash, shift),
                             new Object[] {null, this, null, null})
        .assoc(edit, shift, hash, key, val, addedLeaf);
}

public INode without(AtomicReference<Thread> edit, int shift,
                     int hash, Object key, Box removedLeaf){
    int idx = findIndex(key);

```

```

        if(idx == -1)
            return this;
        if(count == 1)
            return null;
        HashCollisionNode editable = ensureEditable(edit);
        editable.array[idx] = editable.array[2*count-2];
        editable.array[idx+1] = editable.array[2*count-1];
        editable.array[2*count-2] = editable.array[2*count-1] = null;
        editable.count--;
        return editable;
    }
}

```

(IEitableCollection [774]) (IObj [800])
— PersistentHashMap.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.io.Serializable;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.concurrent.atomic.AtomicReference;

/*
A persistent rendition of Phil Bagwell's Hash Array Mapped Trie

Uses path copying for persistence
HashCollision leaves vs. extended hashing
Node polymorphism vs. conditionals
No sub-tree pools or root-resizing
Any errors are my own
*/

```

```

public class PersistentHashMap
    extends APersistentMap implements IEitableCollection, IObj {

    final int count;
    final INode root;
    final boolean hasNull;
    final Object nullValue;
    final IPersistentMap _meta;

    final public static PersistentHashMap EMPTY =
        new PersistentHashMap(0, null, false, null);

```

```

final private static Object NOT_FOUND = new Object();

static public IPersistentMap create(Map other){
    ITransientMap ret = EMPTY.asTransient();
    for(Object o : other.entrySet())
    {
        Map.Entry e = (Entry) o;
        ret = ret.assoc(e.getKey(), e.getValue());
    }
    return ret.persistent();
}

/*
 * @param init {key1,val1,key2,val2,...}
 */
public static PersistentHashMap create(Object... init){
    ITransientMap ret = EMPTY.asTransient();
    for(int i = 0; i < init.length; i += 2)
    {
        ret = ret.assoc(init[i], init[i + 1]);
    }
    return (PersistentHashMap) ret.persistent();
}

public static PersistentHashMap createWithCheck(Object... init){
    ITransientMap ret = EMPTY.asTransient();
    for(int i = 0; i < init.length; i += 2)
    {
        ret = ret.assoc(init[i], init[i + 1]);
        if(ret.count() != i/2 + 1)
            throw new IllegalArgumentException(
                "Duplicate key: " + init[i]);
    }
    return (PersistentHashMap) ret.persistent();
}

static public PersistentHashMap create(ISeq items){
    ITransientMap ret = EMPTY.asTransient();
    for(; items != null; items = items.next().next())
    {
        if(items.next() == null)
            throw new IllegalArgumentException(String.format(
                "No value supplied for key: %s", items.first()));
        ret = ret.assoc(items.first(), RT.second(items));
    }
    return (PersistentHashMap) ret.persistent();
}

static public PersistentHashMap createWithCheck(ISeq items){
    ITransientMap ret = EMPTY.asTransient();

```

```

        for(int i=0; items != null; items = items.next().next(), ++i)
        {
            if(items.next() == null)
                throw new IllegalArgumentException(String.format(
                    "No value supplied for key: %s", items.first()));
            ret = ret.assoc(items.first(), RT.second(items));
            if(ret.count() != i + 1)
                throw new IllegalArgumentException(
                    "Duplicate key: " + items.first());
        }
        return (PersistentHashMap) ret.persistent();
    }

/*
 * @param init {key1,val1,key2,val2,...}
 */
public static PersistentHashMap create(IPersistentMap meta,
                                         Object... init){
    return create(init).withMeta(meta);
}

PersistentHashMap(int count,
                  INode root,
                  boolean hasNull,
                  Object nullValue){
    this.count = count;
    this.root = root;
    this.hasNull = hasNull;
    this.nullValue = nullValue;
    this._meta = null;
}

public PersistentHashMap(IPersistentMap meta,
                        int count,
                        INode root,
                        boolean hasNull,
                        Object nullValue){
    this._meta = meta;
    this.count = count;
    this.root = root;
    this.hasNull = hasNull;
    this.nullValue = nullValue;
}

public boolean containsKey(Object key){
    if(key == null)
        return hasNull;
    return (root != null)
        ? root.find(0, Util.hash(key), key, NOT_FOUND) != NOT_FOUND
        : false;
}

```

```

}

public IMapEntry entryAt(Object key){
    if(key == null)
        return hasNull ? new MapEntry(null, nullValue) : null;
    return (root != null) ? root.find(0, Util.hash(key), key) : null;
}

public IPersistentMap assoc(Object key, Object val){
    if(key == null) {
        if(hasNull && val == nullValue)
            return this;
        return new PersistentHashMap(meta(),
                                      hasNull ? count : count + 1,
                                      root, true, val);
    }
    Box addedLeaf = new Box(null);
    INode newroot = (root == null ? BitmapIndexedNode.EMPTY : root)
                    .assoc(0, Util.hash(key), key, val, addedLeaf);
    if(newroot == root)
        return this;
    return new PersistentHashMap(meta(),
                                 addedLeaf.val == null
                                 ? count
                                 : count + 1,
                                 newroot, hasNull, nullValue);
}

public Object valAt(Object key, Object notFound){
    if(key == null)
        return hasNull ? nullValue : notFound;
    return root != null
        ? root.find(0, Util.hash(key), key, notFound)
        : notFound;
}

public Object valAt(Object key){
    return valAt(key, null);
}

public IPersistentMap assocEx(Object key, Object val) throws Exception{
    if(containsKey(key))
        throw new Exception("Key already present");
    return assoc(key, val);
}

public IPersistentMap without(Object key){
    if(key == null)
        return
            hasNull

```

```

    ? new PersistentHashMap(meta(), count - 1, root, false, null)
      : this;
    if(root == null)
      return this;
    INode newroot = root.without(0, Util.hash(key), key);
    if(newroot == root)
      return this;
    return
      new PersistentHashMap(meta(), count - 1, newroot,
                           hasNull, nullValue);
  }

  public Iterator iterator(){
    return new SeqIterator(seq());
  }

  public int count(){
    return count;
  }

  public ISeq seq(){
    ISeq s = root != null ? root.nodeSeq() : null;
    return hasNull ? new Cons(new MapEntry(null, nullValue), s) : s;
  }

  public IPersistentCollection empty(){
    return EMPTY.withMeta(meta());
  }

  \getchunk{PersistentHashMap mask method}

  public PersistentHashMap withMeta(IPersistentMap meta){
    return new PersistentHashMap(meta, count, root, hasNull, nullValue);
  }

  public TransientHashMap asTransient() {
    return new TransientHashMap(this);
  }

  public IPersistentMap meta(){
    return _meta;
  }

  static final class TransientHashMap extends ATransientMap {
    AtomicReference<Thread> edit;
    INode root;
    int count;
    boolean hasNull;
    Object nullValue;
    final Box leafFlag = new Box(null);
  }
}

```

```

TransientHashMap(PersistentHashMap m) {
    this(new AtomicReference<Thread>(Thread.currentThread()),
          m.root, m.count, m.hasNull, m.nullValue);
}

TransientHashMap(AtomicReference<Thread> edit, INode root,
                int count, boolean hasNull, Object nullValue) {
    this.edit = edit;
    this.root = root;
    this.count = count;
    this.hasNull = hasNull;
    this.nullValue = nullValue;
}

ITransientMap doAssoc(Object key, Object val) {
    if (key == null) {
        if (this.nullValue != val)
            this.nullValue = val;
        if (!hasNull) {
            this.count++;
            this.hasNull = true;
        }
        return this;
    }
    // Box leafFlag = new Box(null);
    leafFlag.val = null;
    INode n = (root == null ? BitmapIndexedNode.EMPTY : root)
        .assoc(edit, 0, Util.hash(key), key, val, leafFlag);
    if (n != this.root)
        this.root = n;
    if(leafFlag.val != null) this.count++;
    return this;
}

ITransientMap doWithout(Object key) {
    if (key == null) {
        if (!hasNull) return this;
        hasNull = false;
        nullValue = null;
        this.count--;
        return this;
    }
    if (root == null) return this;
    // Box leafFlag = new Box(null);
    leafFlag.val = null;
    INode n = root.without(edit, 0, Util.hash(key), key, leafFlag);
    if (n != root)
        this.root = n;
}

```

```
    if(leafFlag.val != null) this.count--;
    return this;
}

IPersistentMap doPersistent() {
    edit.set(null);
    return new PersistentHashMap(count, root, hasNull, nullValue);
}

Object doValAt(Object key, Object notFound) {
    if (key == null)
        if (hasNull)
            return nullValue;
        else
            return notFound;
    if (root == null)
        return null;
    return root.find(0, Util.hash(key), key, notFound);
}

int doCount() {
    return count;
}

void ensureEditable(){
    Thread owner = edit.get();
    if(owner == Thread.currentThread())
        return;
    if(owner != null)
        throw new IllegalAccessError(
            "Transient used by non-owner thread");
    throw new IllegalAccessError(
        "Transient used after persistent! call");
}
}

\getchunk{PersistentHashMap INode interface}

\getchunk{PersistentHashMap ArrayNode class}

\getchunk{PersistentHashMap BitmapIndexedNode class}

\getchunk{PersistentHashMap HashCollisionNode class}

/*
public static void main(String[] args){
    try
    {
        ArrayList words = new ArrayList();
        Scanner s = new Scanner(new File(args[0]));
        for (String word : s)
            words.add(word);
        PersistentHashMap<String, String> map =
            new PersistentHashMap<String, String>(words);
        System.out.println("map = " + map);
        map.ensureEditable();
        map.put("world", "hello");
        System.out.println("map = " + map);
        map.doPersistent();
        System.out.println("map = " + map);
    }
}
```

```

s.useDelimiter(Pattern.compile("\\W"));
while(s.hasNext())
{
    String word = s.next();
    words.add(word);
}
System.out.println("words: " + words.size());
IPersistentMap map = PersistentHashMap.EMPTY;
//IPersistentMap map = new PersistentTreeMap();
//Map ht = new Hashtable();
Map ht = new HashMap();
Random rand;

System.out.println("Building map");
long startTime = System.nanoTime();
for(Object word5 : words)
{
    map = map.assoc(word5, word5);
}
rand = new Random(42);
IPersistentMap snapshotMap = map;
for(int i = 0; i < words.size() / 200; i++)
{
    map = map.without(words.get(rand.nextInt(words.size() / 2)));
}
long estimatedTime = System.nanoTime() - startTime;
System.out.println("count = " + map.count() +
                   ", time: " + estimatedTime / 1000000);

System.out.println("Building ht");
startTime = System.nanoTime();
for(Object word1 : words)
{
    ht.put(word1, word1);
}
rand = new Random(42);
for(int i = 0; i < words.size() / 200; i++)
{
    ht.remove(words.get(rand.nextInt(words.size() / 2)));
}
estimatedTime = System.nanoTime() - startTime;
System.out.println("count = " + ht.size() +
                   ", time: " + estimatedTime / 1000000);

System.out.println("map lookup");
startTime = System.nanoTime();
int c = 0;
for(Object word2 : words)
{
    if(!map.contains(word2))

```

```

        ++c;
    }
estimatedTime = System.nanoTime() - startTime;
System.out.println("notfound = " + c +
                   ", time: " + estimatedTime / 1000000);
System.out.println("ht lookup");
startTime = System.nanoTime();
c = 0;
for(Object word3 : words)
{
    if(!ht.containsKey(word3))
        ++c;
}
estimatedTime = System.nanoTime() - startTime;
System.out.println("notfound = " + c +
                   ", time: " + estimatedTime / 1000000);
System.out.println("snapshotMap lookup");
startTime = System.nanoTime();
c = 0;
for(Object word4 : words)
{
    if(!snapshotMap.contains(word4))
        ++c;
}
estimatedTime = System.nanoTime() - startTime;
System.out.println("notfound = " + c +
                   ", time: " + estimatedTime / 1000000);
}
catch(FileNotFoundException e)
{
    e.printStackTrace();
}

}

*/
private static INode[] cloneAndSet(INode[] array, int i, INode a) {
    INode[] clone = array.clone();
    clone[i] = a;
    return clone;
}

private static Object[] cloneAndSet(Object[] array, int i, Object a) {
    Object[] clone = array.clone();
    clone[i] = a;
    return clone;
}

private static Object[] cloneAndSet(Object[] array, int i,
                                   Object a, int j, Object b) {

```

```

Object[] clone = array.clone();
clone[i] = a;
clone[j] = b;
return clone;
}

private static Object[] removePair(Object[] array, int i) {
    Object[] newArray = new Object[array.length - 2];
    System.arraycopy(array, 0, newArray, 0, 2*i);
    System.arraycopy(array, 2*(i+1), newArray,
                     2*i, newArray.length - 2*i);
    return newArray;
}

private static INode createNode(int shift, Object key1, Object val1,
                               int key2hash, Object key2,
                               Object val2) {
    int key1hash = Util.hash(key1);
    if(key1hash == key2hash)
        return
            new HashCollisionNode(null, key1hash, 2,
                                  new Object[] {key1, val1, key2, val2});
    Box _ = new Box(null);
    AtomicReference<Thread> edit = new AtomicReference<Thread>();
    return BitmapIndexedNode.EMPTY
        .assoc(edit, shift, key1hash, key1, val1, _)
        .assoc(edit, shift, key2hash, key2, val2, _);
}

private static INode createNode(AtomicReference<Thread> edit,
                               int shift, Object key1, Object val1,
                               int key2hash, Object key2,
                               Object val2) {
    int key1hash = Util.hash(key1);
    if(key1hash == key2hash)
        return
            new HashCollisionNode(null, key1hash, 2,
                                  new Object[] {key1, val1, key2, val2});
    Box _ = new Box(null);
    return BitmapIndexedNode.EMPTY
        .assoc(edit, shift, key1hash, key1, val1, _)
        .assoc(edit, shift, key2hash, key2, val2, _);
}

private static int bitpos(int hash, int shift){
    return 1 << mask(hash, shift);
}

static final class NodeSeq extends ASeq {
    final Object[] array;
}

```

```

final int i;
final ISeq s;

NodeSeq(Object[] array, int i) {
    this(null, array, i, null);
}

static ISeq create(Object[] array) {
    return create(array, 0, null);
}

private static ISeq create(Object[] array, int i, ISeq s) {
    if(s != null)
        return new NodeSeq(null, array, i, s);
    for(int j = i; j < array.length; j+=2) {
        if(array[j] != null)
            return new NodeSeq(null, array, j, null);
        INode node = (INode) array[j+1];
        if (node != null) {
            ISeq nodeSeq = node.nodeSeq();
            if(nodeSeq != null)
                return new NodeSeq(null, array, j + 2, nodeSeq);
        }
    }
    return null;
}

NodeSeq(IPersistentMap meta, Object[] array, int i, ISeq s) {
    super(meta);
    this.array = array;
    this.i = i;
    this.s = s;
}

public Obj withMeta(IPersistentMap meta) {
    return new NodeSeq(meta, array, i, s);
}

public Object first() {
    if(s != null)
        return s.first();
    return new MapEntry(array[i], array[i+1]);
}

public ISeq next() {
    if(s != null)
        return create(array, i, s.next());
    return create(array, i + 2, null);
}
}

```

}

9.80 PersistentHashSet.java

(APersistentSet [538]) (IObj [800]) (IEditableCollection [774])
— PersistentHashSet.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 3, 2008 */

package clojure.lang;

import java.util.List;

public class PersistentHashSet
    extends APersistentSet implements IObj, IEditableCollection {

    static public final PersistentHashSet EMPTY =
        new PersistentHashSet(null, PersistentHashMap.EMPTY);

    final IPersistentMap _meta;

    public static PersistentHashSet create(Object... init){
        PersistentHashSet ret = EMPTY;
        for(int i = 0; i < init.length; i++)
        {
            ret = (PersistentHashSet) ret.cons(init[i]);
        }
        return ret;
    }

    public static PersistentHashSet create(List init){
        PersistentHashSet ret = EMPTY;
        for(Object key : init)
        {
            ret = (PersistentHashSet) ret.cons(key);
        }
        return ret;
    }

    static public PersistentHashSet create(ISeq items){
        PersistentHashSet ret = EMPTY;
        for(; items != null; items = items.next())
```

```

    {
        ret = (PersistentHashSet) ret.cons(items.first());
    }
    return ret;
}

public static PersistentHashSet createWithCheck(Object... init){
    PersistentHashSet ret = EMPTY;
    for(int i = 0; i < init.length; i++)
    {
        ret = (PersistentHashSet) ret.cons(init[i]);
        if(ret.count() != i + 1)
            throw new IllegalArgumentException(
                "Duplicate key: " + init[i]);
    }
    return ret;
}

public static PersistentHashSet createWithCheck(List init){
    PersistentHashSet ret = EMPTY;
    int i=0;
    for(Object key : init)
    {
        ret = (PersistentHashSet) ret.cons(key);
        if(ret.count() != i + 1)
            throw new IllegalArgumentException("Duplicate key: " + key);
        ++i;
    }
    return ret;
}

static public PersistentHashSet createWithCheck(ISeq items){
    PersistentHashSet ret = EMPTY;
    for(int i=0; items != null; items = items.next(), ++i)
    {
        ret = (PersistentHashSet) ret.cons(items.first());
        if(ret.count() != i + 1)
            throw new IllegalArgumentException(
                "Duplicate key: " + items.first());
    }
    return ret;
}

PersistentHashSet(IPersistentMap meta, IPersistentMap impl){
    super(impl);
    this._meta = meta;
}

public IPersistentSet disjoin(Object key) throws Exception{
    if(contains(key))

```

```

        return new PersistentHashSet(meta(),impl.without(key));
    return this;
}

public IPersistentSet cons(Object o){
    if(contains(o))
        return this;
    return new PersistentHashSet(meta(),impl.assoc(o,o));
}

public IPersistentCollection empty(){
    return EMPTY.withMeta(meta());
}

public PersistentHashSet withMeta(IPersistentMap meta){
    return new PersistentHashSet(meta, impl);
}

public ITransientCollection asTransient() {
    return
        new TransientHashSet(((PersistentHashMap) impl).asTransient());
}

public IPersistentMap meta(){
    return _meta;
}

static final class TransientHashSet extends ATransientSet {
    TransientHashSet(ITransientMap impl) {
        super(impl);
    }

    public IPersistentCollection persistent() {
        return new PersistentHashSet(null, impl.persistent());
    }
}
}

```

9.81 PersistentList.java

(ASeq [571]) (IPersistentList [801]) (IReduce [804]) (List [1723]) (Counted [768])
— **PersistentList.java** —

```
/*
\getchunk{Clojure Copyright}
```

```
/*
package clojure.lang;

import java.io.Serializable;
import java.util.*;

public class PersistentList
    extends ASq implements IPersistentList, IReduce, List, Counted {

private final Object _first;
private final IPersistentList _rest;
private final int _count;

public static IFn creator = new RestFn(){
    final public int getRequiredArity(){
        return 0;
    }

    final protected Object doInvoke(Object args) throws Exception{
        if(args instanceof ArraySeq)
        {
            Object[] argsarray = (Object[]) ((ArraySeq) args).array;
            IPersistentList ret = EMPTY;
            for(int i = argsarray.length - 1; i >= 0; --i)
                ret = (IPersistentList) ret.cons(argsarray[i]);
            return ret;
        }
        LinkedList list = new LinkedList();
        for(ISeq s = RT.seq(args); s != null; s = s.next())
            list.add(s.first());
        return create(list);
    }

    public IObj withMeta(IPersistentMap meta){
        throw new UnsupportedOperationException();
    }

    public IPersistentMap meta(){
        return null;
    }
};

final public static EmptyList EMPTY = new EmptyList(null);

public PersistentList(Object first){
    this._first = first;
    this._rest = null;

    this._count = 1;
}
```

```
PersistentList(IPersistentMap meta,
               Object _first,
               IPersistentList _rest,
               int _count){
    super(meta);
    this._first = _first;
    this._rest = _rest;
    this._count = _count;
}

public static IPersistentList create(List init){
    IPersistentList ret = EMPTY;
    for(ListIterator i = init.listIterator(init.size())
        ; i.hasPrevious();
        {
            ret = (IPersistentList) ret.cons(i.previous());
        }
    return ret;
}

public Object first(){
    return _first;
}

public ISeq next(){
    if(_count == 1)
        return null;
    return (ISeq) _rest;
}

public Object peek(){
    return first();
}

public IPersistentList pop(){
    if(_rest == null)
        return EMPTY.withMeta(_meta);
    return _rest;
}

public int count(){
    return _count;
}

public PersistentList cons(Object o){
    return new PersistentList(meta(), o, this, _count + 1);
}

public IPersistentCollection empty(){
```

```

        return EMPTY.withMeta(meta());
    }

    public PersistentList withMeta(IPersistentMap meta){
        if(meta != _meta)
            return new PersistentList(meta, _first, _rest, _count);
        return this;
    }

    public Object reduce(IFn f) throws Exception{
        Object ret = first();
        for(ISeq s = next(); s != null; s = s.next())
            ret = f.invoke(ret, s.first());
        return ret;
    }

    public Object reduce(IFn f, Object start) throws Exception{
        Object ret = f.invoke(start, first());
        for(ISeq s = next(); s != null; s = s.next())
            ret = f.invoke(ret, s.first());
        return ret;
    }

    static class EmptyList
        extends Obj implements IPersistentList, List, ISeq, Counted{

        public int hashCode(){
            return 1;
        }

        public boolean equals(Object o) {
            return (o instanceof Sequential ||
                o instanceof List) &&
                RT.seq(o) == null;
        }

        public boolean equiv(Object o){
            return equals(o);
        }

        EmptyList(IPersistentMap meta){
            super(meta);
        }

        public Object first() {
            return null;
        }

        public ISeq next() {

```

```
        return null;
    }

    public ISeq more() {
        return this;
    }

    public PersistentList cons(Object o){
        return new PersistentList(meta(), o, null, 1);
    }

    public IPersistentCollection empty(){
        return this;
    }

    public EmptyList withMeta(IPersistentMap meta){
        if(meta != meta())
            return new EmptyList(meta);
        return this;
    }

    public Object peek(){
        return null;
    }

    public IPersistentList pop(){
        throw new IllegalStateException("Can't pop empty list");
    }

    public int count(){
        return 0;
    }

    public ISeq seq(){
        return null;
    }

    public int size(){
        return 0;
    }

    public boolean isEmpty(){
        return true;
    }

    public boolean contains(Object o){
        return false;
    }
```

```
public Iterator iterator(){
    return new Iterator(){

        public boolean hasNext(){
            return false;
        }

        public Object next(){
            throw new NoSuchElementException();
        }

        public void remove(){
            throw new UnsupportedOperationException();
        }
    };
}

public Object[] toArray(){
    return RT.EMPTY_ARRAY;
}

public boolean add(Object o){
    throw new UnsupportedOperationException();
}

public boolean remove(Object o){
    throw new UnsupportedOperationException();
}

public boolean addAll(Collection collection){
    throw new UnsupportedOperationException();
}

public void clear(){
    throw new UnsupportedOperationException();
}

public boolean retainAll(Collection collection){
    throw new UnsupportedOperationException();
}

public boolean removeAll(Collection collection){
    throw new UnsupportedOperationException();
}

public boolean containsAll(Collection collection){
    return collection.isEmpty();
}

public Object[] toArray(Object[] objects){
```

```
        if(objects.length > 0)
            objects[0] = null;
        return objects;
    }

//////////////// List stuff /////////////////////
private List reify(){
    return Collections.unmodifiableList(new ArrayList(this));
}

public List subList(int fromIndex, int toIndex){
    return reify().subList(fromIndex, toIndex);
}

public Object set(int index, Object element){
    throw new UnsupportedOperationException();
}

public Object remove(int index){
    throw new UnsupportedOperationException();
}

public int indexOf(Object o){
    ISeq s = seq();
    for(int i = 0; s != null; s = s.next(), i++)
    {
        if(Util.equiv(s.first(), o))
            return i;
    }
    return -1;
}

public int lastIndexOf(Object o){
    return reify().lastIndexOf(o);
}

public ListIterator listIterator(){
    return reify().listIterator();
}

public ListIterator listIterator(int index){
    return reify().listIterator(index);
}

public Object get(int index){
    return RT.nth(this, index);
}

public void add(int index, Object element){
    throw new UnsupportedOperationException();
```

```

    }

    public boolean addAll(int index, Collection c){
        throw new UnsupportedOperationException();
    }

}

```

9.82 PersistentQueue.java

(Obj [947]) (IPersistentList [801]) (Collection [1723]) (Counted [768])
 — PersistentQueue.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

import java.util.Collection;
import java.util.Iterator;
//import java.util.concurrent.ConcurrentLinkedQueue;

/**
 * conses onto rear, peeks/pops from front
 * See Okasaki's Batched Queues
 * This differs in that it uses a PersistentVector as the rear,
 * which is in-order,
 * so no reversing or suspensions required for persistent use
 */

public class PersistentQueue
  extends Obj implements IPersistentList, Collection, Counted{

  final public static PersistentQueue EMPTY =
    new PersistentQueue(null, 0, null, null);

  /**
  final int cnt;
  final ISeq f;
  final PersistentVector r;
  //static final int INITIAL_REAR_SIZE = 4;
  int _hash = -1;

```

```

PersistentQueue(IPersistentMap meta,
               int cnt,
               ISeq f,
               PersistentVector r){
  super(meta);
  this.cnt = cnt;
  this.f = f;
  this.r = r;
}

public boolean equiv(Object obj){

  if(!(obj instanceof Sequential))
    return false;
  ISeq ms = RT.seq(obj);
  for(ISeq s = seq(); s != null; s = s.next(), ms = ms.next())
  {
    if(ms == null || !Util.equiv(s.first(), ms.first()))
      return false;
  }
  return ms == null;
}

public boolean equals(Object obj){

  if(!(obj instanceof Sequential))
    return false;
  ISeq ms = RT.seq(obj);
  for(ISeq s = seq(); s != null; s = s.next(), ms = ms.next())
  {
    if(ms == null || !Util.equals(s.first(), ms.first()))
      return false;
  }
  return ms == null;
}

public int hashCode(){
  if(_hash == -1)
  {
    int hash = 0;
    for(ISeq s = seq(); s != null; s = s.next())
    {
      hash = Util.hashCombine(hash, Util.hash(s.first()));
    }
    this._hash = hash;
  }
  return _hash;
}

```

```

public Object peek(){
    return RT.first(f);
}

public PersistentQueue pop(){
    if(f == null) //hmmm... pop of empty queue -> empty queue?
        return this;
    //throw new IllegalStateException("popping empty queue");
    ISeq f1 = f.next();
    PersistentVector r1 = r;
    if(f1 == null)
    {
        f1 = RT.seq(r);
        r1 = null;
    }
    return new PersistentQueue(meta(), cnt - 1, f1, r1);
}

public int count(){
    return cnt;
}

public ISeq seq(){
    if(f == null)
        return null;
    return new Seq(f, RT.seq(r));
}

public PersistentQueue cons(Object o){
    if(f == null) //empty
        return new PersistentQueue(meta(), cnt + 1, RT.list(o), null);
    else
        return new PersistentQueue(meta(), cnt + 1, f,
            (r != null
                ? r
                : PersistentVector.EMPTY).cons(o));
}

public IPersistentCollection empty(){
    return EMPTY.withMeta(meta());
}

public PersistentQueue withMeta(IPersistentMap meta){
    return new PersistentQueue(meta, cnt, f, r);
}

static class Seq extends ASeq{
    final ISeq f;
    final ISeq rseq;
}

```

```
Seq(ISeq f, ISeq rseq){
    this.f = f;
    this.rseq = rseq;
}

Seq(IPersistentMap meta, ISeq f, ISeq rseq){
    super(meta);
    this.f = f;
    this.rseq = rseq;
}

public Object first(){
    return f.first();
}

public ISeq next(){
    ISeq f1 = f.next();
    ISeq r1 = rseq;
    if(f1 == null)
    {
        if(rseq == null)
            return null;
        f1 = rseq;
        r1 = null;
    }
    return new Seq(f1, r1);
}

public int count(){
    return RT.count(f) + RT.count(rseq);
}

public Seq withMeta(IPersistentMap meta){
    return new Seq(meta, f, rseq);
}
}

// java.util.Collection implementation

public Object[] toArray(){
    return RT.seqToArray(seq());
}

public boolean add(Object o){
    throw new UnsupportedOperationException();
}

public boolean remove(Object o){
    throw new UnsupportedOperationException();
```

```
}

public boolean addAll(Collection c){
    throw new UnsupportedOperationException();
}

public void clear(){
    throw new UnsupportedOperationException();
}

public boolean retainAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean removeAll(Collection c){
    throw new UnsupportedOperationException();
}

public boolean containsAll(Collection c){
    for(Object o : c)
    {
        if(contains(o))
            return true;
    }
    return false;
}

public Object[] toArray(Object[] a){
    if(a.length >= count())
    {
        ISeq s = seq();
        for(int i = 0; s != null; ++i, s = s.next())
        {
            a[i] = s.first();
        }
        if(a.length >= count())
            a[count()] = null;
        return a;
    }
    else
        return toArray();
}

public int size(){
    return count();
}

public boolean isEmpty(){
    return count() == 0;
}
```

```
public boolean contains(Object o){
    for(ISeq s = seq(); s != null; s = s.next())
    {
        if(Util.equiv(s.first(), o))
            return true;
    }
    return false;
}

public Iterator iterator(){
    return new SeqIterator(seq());
}

/*
public static void main(String[] args){
    if(args.length != 1)
    {
        System.err.println("Usage: PersistentQueue n");
        return;
    }
    int n = Integer.parseInt(args[0]);

    long startTime, estimatedTime;

    Queue list = new LinkedList();
    //Queue list = new ConcurrentLinkedQueue();
    System.out.println("Queue");
    startTime = System.nanoTime();
    for(int i = 0; i < n; i++)
    {
        list.add(i);
        list.add(i);
        list.remove();
    }
    for(int i = 0; i < n - 10; i++)
    {
        list.remove();
    }
    estimatedTime = System.nanoTime() - startTime;
    System.out.println("time: " + estimatedTime / 1000000);
    System.out.println("peek: " + list.peek());

    PersistentQueue q = PersistentQueue.EMPTY;
    System.out.println("PersistentQueue");
    startTime = System.nanoTime();
    for(int i = 0; i < n; i++)
    {
```

```

        q = q.cons(i);
        q = q.cons(i);
        q = q.pop();
    }
//    IPersistentList lastq = null;
//    IPersistentList lastq2;
    for(int i = 0; i < n - 10; i++)
    {
        //lastq2 = lastq;
        //lastq = q;
        q = q.pop();
    }
estimatedTime = System.nanoTime() - startTime;
System.out.println("time: " + estimatedTime / 1000000);
System.out.println("peek: " + q.peek());

IPersistentList q2 = q;
for(int i = 0; i < 10; i++)
{
    q2 = (IPersistentList) q2.cons(i);
}
//    for(ISeq s = q.seq();s != null;s = s.rest())
//        System.out.println("q: " + s.first().toString());
//    for(ISeq s = q2.seq();s != null;s = s.rest())
//        System.out.println("q2: " + s.first().toString());
}
*/
}

```

—————

9.83 PersistentStructMap.java

(APersistentMap [530]) (IObj [800])
 — PersistentStructMap.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Dec 16, 2007 */

package clojure.lang;

import java.util.Iterator;
import java.util.Map;
import java.io.Serializable;

public class PersistentStructMap extends APersistentMap implements IObj{

```

```

public static class Def implements Serializable{
    final ISeq keys;
    final IPersistentMap keyslots;

    Def(ISeq keys, IPersistentMap keyslots){
        this.keys = keys;
        this.keyslots = keyslots;
    }
}

final Def def;
final Object[] vals;
final IPersistentMap ext;
final IPersistentMap _meta;

static public Def createSlotMap(ISeq keys){
    if(keys == null)
        throw new IllegalArgumentException("Must supply keys");
    int c = RT.count(keys);
    Object[] v = new Object[2*c];
    int i = 0;
    for(ISeq s = keys; s != null; s = s.next(), i++)
    {
        v[2*i] = s.first();
        v[2*i+1] = i;
    }
    return new Def(keys, RT.map(v));
}

static public PersistentStructMap create(Def def, ISeq keyvals){
    Object[] vals = new Object[def.keyslots.count()];
    IPersistentMap ext = PersistentHashMap.EMPTY;
    for(; keyvals != null; keyvals = keyvals.next().next())
    {
        if(keyvals.next() == null)
            throw new IllegalArgumentException(String.format(
                "No value supplied for key: %s", keyvals.first()));
        Object k = keyvals.first();
        Object v = RT.second(keyvals);
        Map.Entry e = def.keyslots.entryAt(k);
        if(e != null)
            vals[(Integer) e.getValue()] = v;
        else
            ext = ext.assoc(k, v);
    }
    return new PersistentStructMap(null, def, vals, ext);
}

```

```

static public PersistentStructMap construct(Def def, ISeq valseq){
    Object[] vals = new Object[def.keysslots.count()];
    IPersistentMap ext = PersistentHashMap.EMPTY;
    for(int i = 0;
        i < vals.length && valseq != null; valseq = valseq.next(),
        i++)
    {
        vals[i] = valseq.first();
    }
    if(valseq != null)
        throw new IllegalArgumentException(
            "Too many arguments to struct constructor");
    return new PersistentStructMap(null, def, vals, ext);
}

static public IFn getAccessor(final Def def, Object key){
    Map.Entry e = def.keysslots.entryAt(key);
    if(e != null)
    {
        final int i = (Integer) e.getValue();
        return new AFn(){
            public Object invoke(Object arg1) throws Exception{
                PersistentStructMap m = (PersistentStructMap) arg1;
                if(m.def != def)
                    throw new Exception("Accessor/struct mismatch");
                return m.vals[i];
            }
        };
    }
    throw new IllegalArgumentException("Not a key of struct");
}

protected PersistentStructMap(IPersistentMap meta, Def def,
                             Object[] vals, IPersistentMap ext){
    this._meta = meta;
    this.ext = ext;
    this.def = def;
    this.vals = vals;
}

/**
 * Returns a new instance of PersistentStructMap using the given
 * parameters. This function is used instead of the
 * PersistentStructMap constructor by all methods that return a
 * new PersistentStructMap. This is done so as to allow subclasses
 * to return instances of their class from all PersistentStructMap
 * methods.
 */
protected PersistentStructMap makeNew(IPersistentMap meta,
                                      Def def,

```

```

        Object[] vals,
        IPersistentMap ext){
    return new PersistentStructMap(meta, def, vals, ext);
}

public IObj withMeta(IPersistentMap meta){
    if(meta == _meta)
        return this;
    return makeNew(meta, def, vals, ext);
}

public IPersistentMap meta(){
    return _meta;
}

public boolean containsKey(Object key){
    return def.keysslots.containsKey(key) || ext.containsKey(key);
}

public IMapEntry entryAt(Object key){
    Map.Entry e = def.keysslots.entryAt(key);
    if(e != null)
    {
        return new MapEntry(e.getKey(), vals[(Integer) e.getValue()]);
    }
    return ext.entryAt(key);
}

public IPersistentMap assoc(Object key, Object val){
    Map.Entry e = def.keysslots.entryAt(key);
    if(e != null)
    {
        int i = (Integer) e.getValue();
        Object[] newVals = vals.clone();
        newVals[i] = val;
        return makeNew(_meta, def, newVals, ext);
    }
    return makeNew(_meta, def, vals, ext.assoc(key, val));
}

public Object valAt(Object key){
    Integer i = (Integer) def.keysslots.valAt(key);
    if(i != null)
    {
        return vals[i];
    }
    return ext.valAt(key);
}

public Object valAt(Object key, Object notFound){
}

```

```

        Integer i = (Integer) def.keysslots.valAt(key);
        if(i != null)
        {
            return vals[i];
        }
        return ext.valAt(key, notFound);
    }

    public IPersistentMap assocEx(Object key, Object val) throws Exception{
        if(containsKey(key))
            throw new Exception("Key already present");
        return assoc(key, val);
    }

    public IPersistentMap without(Object key) throws Exception{
        Map.Entry e = def.keysslots.entryAt(key);
        if(e != null)
            throw new Exception("Can't remove struct key");
        IPersistentMap newExt = ext.without(key);
        if(newExt == ext)
            return this;
        return makeNew(_meta, def, vals, newExt);
    }

    public Iterator iterator(){
        return new SeqIterator(seq());
    }

    public int count(){
        return vals.length + RT.count(ext);
    }

    public ISeq seq(){
        return new Seq(null, def.keys, vals, 0, ext);
    }

    public IPersistentCollection empty(){
        return construct(def, null);
    }

    static class Seq extends ASeq{
        final int i;
        final ISeq keys;
        final Object[] vals;
        final IPersistentMap ext;

        public Seq(IPersistentMap meta,
                   ISeq keys,

```

```

        Object[] vals,
        int i,
        IPersistentMap ext){
super(meta);
this.i = i;
this.keys = keys;
this.vals = vals;
this.ext = ext;
}

public Obj withMeta(IPersistentMap meta){
    if(meta != _meta)
        return new Seq(meta, keys, vals, i, ext);
    return this;
}

public Object first(){
    return new MapEntry(keys.first(), vals[i]);
}

public ISeq next(){
    if(i + 1 < vals.length)
        return new Seq(_meta, keys.next(), vals, i + 1, ext);
    return ext.seq();
}
}
}

```

—————

9.84 PersistentTreeMap.java

(APersistentMap [530]) (IObj [800]) (Reversible [1094]) (Sorted [1141])
 — PersistentTreeMap.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich May 20, 2006 */

package clojure.lang;

import java.util.*;

/**
 * Persistent Red Black Tree
 * Note that instances of this class are constant values
 * i.e. add/remove etc return new values

```

```
* <p/>
* See Okasaki, Kahrs, Larsen et al
*/
public class PersistentTreeMap
    extends APersistentMap implements IObj, Reversible, Sorted{

    public final Comparator comp;
    public final Node tree;
    public final int _count;
    final IPersistentMap _meta;

    final static public PersistentTreeMap EMPTY = new PersistentTreeMap();

    static public IPersistentMap create(Map other){
        IPersistentMap ret = EMPTY;
        for(Object o : other.entrySet())
        {
            Map.Entry e = (Entry) o;
            ret = ret.assoc(e.getKey(), e.getValue());
        }
        return ret;
    }

    public PersistentTreeMap(){
        this(RT.DEFAULT_COMPARATOR);
    }

    public PersistentTreeMap withMeta(IPersistentMap meta){
        return new PersistentTreeMap(meta, comp, tree, _count);
    }

    private PersistentTreeMap(Comparator comp){
        this(null, comp);
    }

    public PersistentTreeMap(IPersistentMap meta, Comparator comp){
        this.comp = comp;
        this._meta = meta;
        tree = null;
        _count = 0;
    }

    PersistentTreeMap(IPersistentMap meta,
                      Comparator comp,
                      Node tree,
                      int _count){
        this._meta = meta;
        this.comp = comp;
```

```

        this.tree = tree;
        this._count = _count;
    }

    static public PersistentTreeMap create(ISeq items){
        IPersistentMap ret = EMPTY;
        for(; items != null; items = items.next().next())
        {
            if(items.next() == null)
                throw new IllegalArgumentException(String.format(
                    "No value supplied for key: %s", items.first()));
            ret = ret.assoc(items.first(), RT.second(items));
        }
        return (PersistentTreeMap) ret;
    }

    static public PersistentTreeMap create(Comparator comp, ISeq items){
        IPersistentMap ret = new PersistentTreeMap(comp);
        for(; items != null; items = items.next().next())
        {
            if(items.next() == null)
                throw new IllegalArgumentException(String.format(
                    "No value supplied for key: %s", items.first()));
            ret = ret.assoc(items.first(), RT.second(items));
        }
        return (PersistentTreeMap) ret;
    }

    public boolean containsKey(Object key){
        return entryAt(key) != null;
    }

    public PersistentTreeMap assocEx(Object key, Object val)
    throws Exception{
        Box found = new Box(null);
        Node t = add(tree, key, val, found);
        if(t == null)    //null == already contains key
        {
            throw new Exception("Key already present");
        }
        return new PersistentTreeMap(comp, t.blacken(), _count + 1, meta());
    }

    public PersistentTreeMap assoc(Object key, Object val){
        Box found = new Box(null);
        Node t = add(tree, key, val, found);
        if(t == null)    //null == already contains key
        {
            Node foundNode = (Node) found.val;
            if(foundNode.val() == val)  //note only get same collection

```

```

        //on identity of val, not equals()
        return this;
    return
        new PersistentTreeMap(comp,
            replace(tree, key, val),
            _count,
            meta());
    }
    return new PersistentTreeMap(comp, t.blacken(), _count + 1, meta());
}
}

public PersistentTreeMap without(Object key){
    Box found = new Box(null);
    Node t = remove(tree, key, found);
    if(t == null)
    {
        if(found.val == null)//null == doesn't contain key
            return this;
        //empty
        return new PersistentTreeMap(meta(), comp);
    }
    return new PersistentTreeMap(comp, t.blacken(), _count - 1, meta());
}

public ISeq seq(){
    if(_count > 0)
        return Seq.create(tree, true, _count);
    return null;
}

public IPersistentCollection empty(){
    return new PersistentTreeMap(meta(), comp);
}

public ISeq rseq() throws Exception{
    if(_count > 0)
        return Seq.create(tree, false, _count);
    return null;
}

public Comparator comparator(){
    return comp;
}

public Object entryKey(Object entry){
    return ((IMapEntry) entry).key();
}

public ISeq seq(boolean ascending){

```

```

        if(_count > 0)
            return Seq.create(tree, ascending, _count);
        return null;
    }

    public ISeq seqFrom(Object key, boolean ascending){
        if(_count > 0)
        {
            ISeq stack = null;
            Node t = tree;
            while(t != null)
            {
                int c = doCompare(key, t.key);
                if(c == 0)
                {
                    stack = RT.cons(t, stack);
                    return new Seq(stack, ascending);
                }
                else if(ascending)
                {
                    if(c < 0)
                    {
                        stack = RT.cons(t, stack);
                        t = t.left();
                    }
                    else
                        t = t.right();
                }
                else
                {
                    if(c > 0)
                    {
                        stack = RT.cons(t, stack);
                        t = t.right();
                    }
                    else
                        t = t.left();
                }
            }
            if(stack != null)
                return new Seq(stack, ascending);
        }
        return null;
    }

    public NodeIterator iterator(){
        return new NodeIterator(tree, true);
    }

    public NodeIterator reverseIterator(){

```

```
        return new NodeIterator(tree, false);
    }

    public Iterator keys(){
        return keys(iterator());
    }

    public Iterator vals(){
        return vals(iterator());
    }

    public Iterator keys(NodeIterator it){
        return new KeyIterator(it);
    }

    public Iterator vals(NodeIterator it){
        return new ValIterator(it);
    }

    public Object minKey(){
        Node t = min();
        return t != null ? t.key : null;
    }

    public Node min(){
        Node t = tree;
        if(t != null)
        {
            while(t.left() != null)
                t = t.left();
        }
        return t;
    }

    public Object maxKey(){
        Node t = max();
        return t != null ? t.key : null;
    }

    public Node max(){
        Node t = tree;
        if(t != null)
        {
            while(t.right() != null)
                t = t.right();
        }
        return t;
    }

    public int depth(){
```

```

        return depth(tree);
    }

    int depth(Node t){
        if(t == null)
            return 0;
        return 1 + Math.max(depth(t.left()), depth(t.right()));
    }

    public Object valAt(Object key, Object notFound){
        Node n = entryAt(key);
        return (n != null) ? n.val() : notFound;
    }

    public Object valAt(Object key){
        return valAt(key, null);
    }

    public int capacity(){
        return _count;
    }

    public int count(){
        return _count;
    }

    public Node entryAt(Object key){
        Node t = tree;
        while(t != null)
        {
            int c = doCompare(key, t.key);
            if(c == 0)
                return t;
            else if(c < 0)
                t = t.left();
            else
                t = t.right();
        }
        return t;
    }

    public int doCompare(Object k1, Object k2){
//        if(comp != null)
//            return comp.compare(k1, k2);
//        return ((Comparable) k1).compareTo(k2);
    }

    Node add(Node t, Object key, Object val, Box found){
        if(t == null)
        {

```

```

        if(val == null)
            return new Red(key);
        return new RedVal(key, val);
    }
    int c = doCompare(key, t.key);
    if(c == 0)
    {
        found.val = t;
        return null;
    }
    Node ins = c < 0
        ? add(t.left(), key, val, found)
        : add(t.right(), key, val, found);
    if(ins == null) //found below
        return null;
    if(c < 0)
        return t.addLeft(ins);
    return t.addRight(ins);
}

Node remove(Node t, Object key, Box found){
    if(t == null)
        return null; //not found indicator
    int c = doCompare(key, t.key);
    if(c == 0)
    {
        found.val = t;
        return append(t.left(), t.right());
    }
    Node del = c < 0
        ? remove(t.left(), key, found)
        : remove(t.right(), key, found);
    if(del == null && found.val == null) //not found below
        return null;
    if(c < 0)
    {
        if(t.left() instanceof Black)
            return balanceLeftDel(t.key, t.val(), del, t.right());
        else
            return red(t.key, t.val(), del, t.right());
    }
    if(t.right() instanceof Black)
        return balanceRightDel(t.key, t.val(), t.left(), del);
    return red(t.key, t.val(), t.left(), del);
//    return t.removeLeft(del);
//    return t.removeRight(del);
}

static Node append(Node left, Node right){
    if(left == null)

```

```

        return right;
    else if(right == null)
        return left;
    else if(left instanceof Red)
    {
        if(right instanceof Red)
        {
            Node app = append(left.right(), right.left());
            if(app instanceof Red)
                return
                    red(app.key, app.val(),
                        red(left.key, left.val(),
                            left.left(), app.left()),
                        red(right.key, right.val(),
                            app.right(), right.right()));
            else
                return
                    red(left.key, left.val(), left.left(),
                        red(right.key, right.val(), app, right.right()));
        }
        else
            return red(left.key, left.val(), left.left(),
                append(left.right(), right));
    }
    else if(right instanceof Red)
        return red(right.key, right.val(),
            append(left, right.left()), right.right());
    else //black/black
    {
        Node app = append(left.right(), right.left());
        if(app instanceof Red)
            return
                red(app.key, app.val(),
                    black(left.key, left.val(),
                        left.left(), app.left()),
                    black(right.key, right.val(),
                        app.right(), right.right()));
        else
            return balanceLeftDel(left.key, left.val(), left.left(),
                black(right.key, right.val(),
                    app, right.right()));
    }
}

\getchunk{PersistentTreeMap balanceLeftDel method}

\getchunk{PersistentTreeMap balanceRightDel method}

\getchunk{PersistentTreeMap leftBalance method}

```

```
\getchunk{PersistentTreeMap rightBalance method}

\getchunk{PersistentTreeMap replace method}

PersistentTreeMap(Comparator comp, Node tree,
                  int count, IPersistentMap meta){
    this._meta = meta;
    this.comp = comp;
    this.tree = tree;
    this._count = count;
}

\getchunk{PersistentTreeMap red method}

\getchunk{PersistentTreeMap black method}

public IPersistentMap meta(){
    return _meta;
}

\getchunk{PersistentTreeMap Node Class}

\getchunk{PersistentTreeMap Black Class}

\getchunk{PersistentTreeMap BlackVal Class}

\getchunk{PersistentTreeMap BlackBranch Class}

\getchunk{PersistentTreeMap BlackBranchVal Class}

\getchunk{PersistentTreeMap Red Class}

\getchunk{PersistentTreeMap RedVal Class}

\getchunk{PersistentTreeMap RedBranch Class}

\getchunk{PersistentTreeMap RedBranchVal Class}

\getchunk{PersistentTreeMap Seq Class}

\getchunk{PersistentTreeMap NodeIterator Class}

\getchunk{PersistentTreeMap KeyIterator Class}

\getchunk{PersistentTreeMap ValIterator Class}

/*
static public void main(String args[]){
    if(args.length != 1)
        System.err.println("Usage: RBTree n");
```

```

int n = Integer.parseInt(args[0]);
Integer[] ints = new Integer[n];
for(int i = 0; i < ints.length; i++)
{
    ints[i] = i;
}
Collections.shuffle((Arrays.asList(ints)));
//force the ListMap class loading now
//    try
//    {
//        //PersistentListMap.EMPTY.assocEx(1, null)
//        .assocEx(2,null).assocEx(3,null);
//    }
//    catch(Exception e)
//    {
//        e.printStackTrace(); //To change body of catch statement
//        //use File | Settings | File Templates.
//    }
System.out.println("Building set");
//IPersistentMap set = new PersistentArrayMap();
//IPersistentMap set = new PersistentHashtableMap(1001);
IPersistentMap set = PersistentHashMap.EMPTY;
//IPersistentMap set = new ListMap();
//IPersistentMap set = new ArrayMap();
//IPersistentMap set = new PersistentTreeMap();
//    for(int i = 0; i < ints.length; i++)
//    {
//        Integer anInt = ints[i];
//        set = set.add(anInt);
//    }
long startTime = System.nanoTime();
for(Integer anInt : ints)
{
    set = set.assoc(anInt, anInt);
}
//System.out.println("_count = " + set.count());

//    System.out.println("_count = " + set._count +
//                        ", min: " + set.minKey() + ", max: " +
//                        set.maxKey()
//                        + ", depth: " + set.depth());
for(Object aSet : set)
{
    IMapEntry o = (IMapEntry) aSet;
    if(!set.contains(o.key()))
        System.err.println("Can't find: " + o.key());
    //else if(n < 2000)
    //    System.out.print(o.key().toString() + ",");
}

```

```
Random rand = new Random(42);
for(int i = 0; i < ints.length / 2; i++)
{
    Integer anInt = ints[rand.nextInt(n)];
    set = set.without(anInt);
}

long estimatedTime = System.nanoTime() - startTime;
System.out.println();

System.out.println("_count = " + set.count() +
                   ", time: " + estimatedTime / 1000000);

System.out.println("Building ht");
Hashtable ht = new Hashtable(1001);
startTime = System.nanoTime();
//    for(int i = 0; i < ints.length; i++)
//    {
//        Integer anInt = ints[i];
//        ht.put(anInt,null);
//    }
    for(Integer anInt : ints)
    {
        ht.put(anInt, anInt);
    }
//System.out.println("size = " + ht.size());
//Iterator it = ht.entrySet().iterator();
for(Object o1 : ht.entrySet())
{
    Map.Entry o = (Map.Entry) o1;
    if(!ht.containsKey(o.getKey()))
        System.err.println("Can't find: " + o);
    //else if(n < 2000)
    //    System.out.print(o.toString() + ",");
}

rand = new Random(42);
for(int i = 0; i < ints.length / 2; i++)
{
    Integer anInt = ints[rand.nextInt(n)];
    ht.remove(anInt);
}
estimatedTime = System.nanoTime() - startTime;
System.out.println();
System.out.println("size = " + ht.size() + ", time: " +
                   estimatedTime / 1000000);

System.out.println("set lookup");
startTime = System.nanoTime();
```

```

int c = 0;
for(Integer anInt : ints)
{
    if(!set.contains(anInt))
        ++c;
}
estimatedTime = System.nanoTime() - startTime;
System.out.println("notfound = " + c + ", time: " +
                    estimatedTime / 1000000);

System.out.println("ht lookup");
startTime = System.nanoTime();
c = 0;
for(Integer anInt : ints)
{
    if(!ht.containsKey(anInt))
        ++c;
}
estimatedTime = System.nanoTime() - startTime;
System.out.println("notfound = " + c + ", time: " +
                    estimatedTime / 1000000);

//    System.out.println("_count = " + set._count +
//                        ", min: " + set.minKey() +
//                        ", max: " + set.maxKey() +
//                        ", depth: " + set.depth());
}
*/
}

```

9.85 PersistentTreeSet.java

(APersistentSet [538]) (IObj [800]) (Reversible [1094]) (Sorted [1141])
— PersistentTreeSet.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 3, 2008 */

package clojure.lang;

import java.util.Comparator;

public class PersistentTreeSet
    extends APersistentSet implements IObj, Reversible, Sorted{

```

```

static public final PersistentTreeSet EMPTY =
    new PersistentTreeSet(null, PersistentTreeMap.EMPTY);
final IPersistentMap _meta;

static public PersistentTreeSet create(ISeq items){
    PersistentTreeSet ret = EMPTY;
    for(; items != null; items = items.next())
    {
        ret = (PersistentTreeSet) ret.cons(items.first());
    }
    return ret;
}

static public PersistentTreeSet create(Comparator comp, ISeq items){
    PersistentTreeSet ret =
        new PersistentTreeSet(null, new PersistentTreeMap(null, comp));
    for(; items != null; items = items.next())
    {
        ret = (PersistentTreeSet) ret.cons(items.first());
    }
    return ret;
}

PersistentTreeSet(IPersistentMap meta, IPersistentMap impl){
    super(impl);
    this._meta = meta;
}

public IPersistentSet disjoin(Object key) throws Exception{
    if(contains(key))
        return new PersistentTreeSet(meta(),impl.without(key));
    return this;
}

public IPersistentSet cons(Object o){
    if(contains(o))
        return this;
    return new PersistentTreeSet(meta(),impl.assoc(o,o));
}

public IPersistentCollection empty(){
    return
        new PersistentTreeSet(meta(),(PersistentTreeMap)impl.empty());
}

public ISeq rseq() throws Exception{
    return APersistentMap.KeySeq.create(((Reversible) impl).rseq());
}

```

```

public PersistentTreeSet withMeta(IPersistentMap meta){
    return new PersistentTreeSet(meta, impl);
}

public Comparator comparator(){
    return ((Sorted)impl).comparator();
}

public Object entryKey(Object entry){
    return entry;
}

public ISeq seq(boolean ascending){
    PersistentTreeMap m = (PersistentTreeMap) impl;
    return RT.keys(m.seq(ascending));
}

public ISeq seqFrom(Object key, boolean ascending){
    PersistentTreeMap m = (PersistentTreeMap) impl;
    return RT.keys(m.seqFrom(key, ascending));
}

public IPersistentMap meta(){
    return _meta;
}
}

```

—————

9.86 PersistentVector.java

(APersistentVector [541]) (IObj [800]) (IEditableCollection [774])
 — PersistentVector.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 5, 2007 */

package clojure.lang;

import java.io.Serializable;
import java.util.List;
import java.util.concurrent.atomic.AtomicReference;

public class PersistentVector
    extends APersistentVector implements IObj, IEditableCollection{

```

```
static class Node implements Serializable {
    transient final AtomicReference<Thread> edit;
    final Object[] array;

    Node(AtomicReference<Thread> edit, Object[] array){
        this.edit = edit;
        this.array = array;
    }

    Node(AtomicReference<Thread> edit){
        this.edit = edit;
        this.array = new Object[32];
    }
}

final static AtomicReference<Thread> NOEDIT =
    new AtomicReference<Thread>(null);
final static Node EMPTY_NODE = new Node(NOEDIT, new Object[32]);

final int cnt;
final int shift;
final Node root;
final Object[] tail;
final IPersistentMap _meta;

public final static PersistentVector EMPTY =
    new PersistentVector(0, 5, EMPTY_NODE, new Object[]{});

static public PersistentVector create(ISeq items){
    TransientVector ret = EMPTY.asTransient();
    for(; items != null; items = items.next())
        ret = ret.conj(items.first());
    return ret.persistent();
}

static public PersistentVector create(List items){
    TransientVector ret = EMPTY.asTransient();
    for(Object item : items)
        ret = ret.conj(item);
    return ret.persistent();
}

static public PersistentVector create(Object... items){
    TransientVector ret = EMPTY.asTransient();
    for(Object item : items)
        ret = ret.conj(item);
    return ret.persistent();
}
```

```
PersistentVector(int cnt, int shift, Node root, Object[] tail){
    this._meta = null;
    this.cnt = cnt;
    this.shift = shift;
    this.root = root;
    this.tail = tail;
}

PersistentVector(IPersistentMap meta, int cnt, int shift,
                 Node root, Object[] tail){
    this._meta = meta;
    this.cnt = cnt;
    this.shift = shift;
    this.root = root;
    this.tail = tail;
}

public TransientVector asTransient(){
    return new TransientVector(this);
}

final int tailoff(){
    if(cnt < 32)
        return 0;
    return ((cnt - 1) >>> 5) << 5;
}

public Object[] arrayFor(int i){
    if(i >= 0 && i < cnt)
    {
        if(i >= tailoff())
            return tail;
        Node node = root;
        for(int level = shift; level > 0; level -= 5)
            node = (Node) node.array[(i >>> level) & 0x01f];
        return node.array;
    }
    throw new IndexOutOfBoundsException();
}

public Object nth(int i){
    Object[] node = arrayFor(i);
    return node[i & 0x01f];
}

public Object nth(int i, Object notFound){
    if(i >= 0 && i < cnt)
        return nth(i);
```

```

        return notFound;
    }

    public PersistentVector assocN(int i, Object val){
        if(i >= 0 && i < cnt)
        {
            if(i >= tailoff())
            {
                Object[] newTail = new Object[tail.length];
                System.arraycopy(tail, 0, newTail, 0, tail.length);
                newTail[i & 0x01f] = val;
                return
                    new PersistentVector(meta(), cnt, shift, root, newTail);
            }
        }

        return
            new PersistentVector(meta(), cnt, shift,
                doAssoc(shift, root, i, val), tail);
    }

    if(i == cnt)
        return cons(val);
    throw new IndexOutOfBoundsException();
}

private static Node doAssoc(int level, Node node, int i, Object val){
    Node ret = new Node(node.edit,node.array.clone());
    if(level == 0)
    {
        ret.array[i & 0x01f] = val;
    }
    else
    {
        int subidx = (i >>> level) & 0x01f;
        ret.array[subidx] =
            doAssoc(level - 5, (Node) node.array[subidx], i, val);
    }
    return ret;
}

public int count(){
    return cnt;
}

public PersistentVector withMeta(IPersistentMap meta){
    return new PersistentVector(meta, cnt, shift, root, tail);
}

public IPersistentMap meta(){
    return _meta;
}

```

```

public PersistentVector cons(Object val){
    int i = cnt;
    //room in tail?
    //  if(tail.length < 32)
    if(cnt - tailoff() < 32)
    {
        Object[] newTail = new Object[tail.length + 1];
        System.arraycopy(tail, 0, newTail, 0, tail.length);
        newTail[tail.length] = val;
        return
            new PersistentVector(meta(), cnt + 1, shift, root, newTail);
    }
    //full tail, push into tree
    Node newroot;
    Node tailnode = new Node(root.edit,tail);
    int newshift = shift;
    //overflow root?
    if((cnt >>> 5) > (1 << shift))
    {
        newroot = new Node(root.edit);
        newroot.array[0] = root;
        newroot.array[1] = newPath(root.edit,shift, tailnode);
        newshift += 5;
    }
    else
        newroot = pushTail(shift, root, tailnode);
    return
        new PersistentVector(meta(), cnt + 1, newshift,
                           newroot, new Object[]{val});
}

private Node pushTail(int level, Node parent, Node tailnode){
    //if parent is leaf, insert node,
    // else does it map to an existing
    // child? -> nodeToInsert = pushNode one more level
    // else alloc new path
    //return nodeToInsert placed in copy of parent
    int subidx = ((cnt - 1) >>> level) & 0x01f;
    Node ret = new Node(parent.edit, parent.array.clone());
    Node nodeToInsert;
    if(level == 5)
    {
        nodeToInsert = tailnode;
    }
    else
    {
        Node child = (Node) parent.array[subidx];
        nodeToInsert = (child != null)?

```

```
        pushTail(level-5,child, tailnode)
        :newPath(root.edit,level-5, tailnode);
    }
    ret.array[subidx] = nodeToInsert;
    return ret;
}

private static Node newPath(AtomicReference<Thread> edit,
                           int level, Node node){
    if(level == 0)
        return node;
    Node ret = new Node(edit);
    ret.array[0] = newPath(edit, level - 5, node);
    return ret;
}

public IChunkedSeq chunkedSeq(){
    if(count() == 0)
        return null;
    return new ChunkedSeq(this,0,0);
}

public ISeq seq(){
    return chunkedSeq();
}

static public final class ChunkedSeq
extends ASeq implements IChunkedSeq{

    public final PersistentVector vec;
    final Object[] node;
    final int i;
    public final int offset;

    public ChunkedSeq(PersistentVector vec, int i, int offset){
        this.vec = vec;
        this.i = i;
        this.offset = offset;
        this.node = vec.arrayFor(i);
    }

    ChunkedSeq(IPersistentMap meta, PersistentVector vec,
               Object[] node, int i, int offset){
        super(meta);
        this.vec = vec;
        this.node = node;
        this.i = i;
        this.offset = offset;
    }
}
```

```

    ChunkedSeq(PersistentVector vec, Object[] node, int i, int offset){
        this.vec = vec;
        this.node = node;
        this.i = i;
        this.offset = offset;
    }

    public IChunk chunkedFirst() throws Exception{
        return new ArrayChunk(node, offset);
    }

    public ISeq chunkedNext(){
        if(i + node.length < vec.cnt)
            return new ChunkedSeq(vec,i+ node.length,0);
        return null;
    }

    public ISeq chunkedMore(){
        ISeq s = chunkedNext();
        if(s == null)
            return PersistentList.EMPTY;
        return s;
    }

    public Obj withMeta(IPersistentMap meta){
        if(meta == this._meta)
            return this;
        return new ChunkedSeq(meta, vec, node, i, offset);
    }

    public Object first(){
        return node[offset];
    }

    public ISeq next(){
        if(offset + 1 < node.length)
            return new ChunkedSeq(vec, node, i, offset + 1);
        return chunkedNext();
    }
}

public IPersistentCollection empty(){
    return EMPTY.withMeta(meta());
}

//private Node pushTail(int level, Node node,
//                      Object[] tailNode, Box expansion){
//    Object newchild;
//    if(level == 0)
//        {

```

```
//      newchild = tailNode;
//    }
//  else
//  {
//    newchild =
//      pushTail(level - 5,
//                (Object[]) arr[arr.length - 1], tailNode, expansion);
//  if(expansion.val == null)
//  {
//    Object[] ret = arr.clone();
//    ret[arr.length - 1] = newchild;
//    return ret;
//  }
//  else
//    newchild = expansion.val;
//  }
// //expansion
// if(arr.length == 32)
// {
//   expansion.val = new Object[]{newchild};
//   return arr;
// }
// Object[] ret = new Object[arr.length + 1];
// System.arraycopy(arr, 0, ret, 0, arr.length);
// ret[arr.length] = newchild;
// expansion.val = null;
// return ret;
//}

public PersistentVector pop(){
  if(cnt == 0)
    throw new IllegalStateException("Can't pop empty vector");
  if(cnt == 1)
    return EMPTY.withMeta(meta());
  //if(tail.length > 1)
  if(cnt-tailoff() > 1)
  {
    Object[] newTail = new Object[tail.length - 1];
    System.arraycopy(tail, 0, newTail, 0, newTail.length);
    return
      new PersistentVector(meta(), cnt - 1, shift, root, newTail);
  }
  Object[] newtail = arrayFor(cnt - 2);

  Node newroot = popTail(shift, root);
  int newshift = shift;
  if(newroot == null)
  {
    newroot = EMPTY_NODE;
  }
}
```

```

if(shift > 5 && newroot.array[1] == null)
{
    newroot = (Node) newroot.array[0];
    newshift -= 5;
}
return
new PersistentVector(meta(), cnt - 1, newshift, newroot, newtail);
}

private Node popTail(int level, Node node){
    int subidx = ((cnt-2) >>> level) & 0x01f;
    if(level > 5)
    {
        Node newchild = popTail(level - 5, (Node) node.array[subidx]);
        if(newchild == null && subidx == 0)
            return null;
        else
        {
            Node ret = new Node(root.edit, node.array.clone());
            ret.array[subidx] = newchild;
            return ret;
        }
    }
    else if(subidx == 0)
        return null;
    else
    {
        Node ret = new Node(root.edit, node.array.clone());
        ret.array[subidx] = null;
        return ret;
    }
}

static final class TransientVector
extends AFn implements ITransientVector, Counted{
    int cnt;
    int shift;
    Node root;
    Object[] tail;

    TransientVector(int cnt, int shift, Node root, Object[] tail){
        this.cnt = cnt;
        this.shift = shift;
        this.root = root;
        this.tail = tail;
    }

    TransientVector(PersistentVector v){
        this(v.cnt, v.shift, editableRoot(v.root), editableTail(v.tail));
    }
}

```

```
public int count(){
    ensureEditable();
    return cnt;
}

Node ensureEditable(Node node){
    if(node.edit == root.edit)
        return node;
    return new Node(root.edit, node.array.clone());
}

void ensureEditable(){
    Thread owner = root.edit.get();
    if(owner == Thread.currentThread())
        return;
    if(owner != null)
        throw new IllegalAccessError(
            "Transient used by non-owner thread");
    throw new IllegalAccessError(
        "Transient used after persistent! call");

//    root = editableRoot(root);
//    tail = editableTail(tail);
}

static Node editableRoot(Node node){
    return
        new Node(new AtomicReference<Thread>(Thread.currentThread()),
            node.array.clone());
}

public PersistentVector persistent(){
    ensureEditable();
//    Thread owner = root.edit.get();
//    if(owner != null && owner != Thread.currentThread())
//    {
//        throw new IllegalAccessError(
//            "Mutation release by non-owner thread");
//    }
    root.edit.set(null);
    Object[] trimmedTail = new Object[cnt-tailoff()];
    System.arraycopy(tail,0,trimmedTail,0,trimmedTail.length);
    return new PersistentVector(cnt, shift, root, trimmedTail);
}

static Object[] editableTail(Object[] tl){
    Object[] ret = new Object[32];
    System.arraycopy(tl,0,ret,0,tl.length);
    return ret;
}
```

```

    }

    public TransientVector conj(Object val){
        ensureEditable();
        int i = cnt;
        //room in tail?
        if(i - tailoff() < 32)
        {
            tail[i & 0x01f] = val;
            ++cnt;
            return this;
        }
        //full tail, push into tree
        Node newroot;
        Node tailnode = new Node(root.edit, tail);
        tail = new Object[32];
        tail[0] = val;
        int newshift = shift;
        //overflow root?
        if((cnt >>> 5) > (1 << shift))
        {
            newroot = new Node(root.edit);
            newroot.array[0] = root;
            newroot.array[1] = newPath(root.edit,shift, tailnode);
            newshift += 5;
        }
        else
            newroot = pushTail(shift, root, tailnode);
        root = newroot;
        shift = newshift;
        ++cnt;
        return this;
    }

    private Node pushTail(int level, Node parent, Node tailnode){
        //if parent is leaf, insert node,
        // else does it map to an existing
        // child? -> nodeToInsert = pushNode one more level
        // else alloc new path
        //return nodeToInsert placed in parent
        parent = ensureEditable(parent);
        int subidx = ((cnt - 1) >>> level) & 0x01f;
        Node ret = parent;
        Node nodeToInsert;
        if(level == 5)
        {
            nodeToInsert = tailnode;
        }
        else
        {

```

```

        Node child = (Node) parent.array[subidx];
        nodeToInsert =
            (child != null)
            ? pushTail(level - 5, child, tailnode)
            : newPath(root.edit, level - 5, tailnode);
    }
    ret.array[subidx] = nodeToInsert;
    return ret;
}

final private int tailoff(){
    if(cnt < 32)
        return 0;
    return ((cnt-1) >>> 5) << 5;
}

private Object[] arrayFor(int i){
    if(i >= 0 && i < cnt)
    {
        if(i >= tailoff())
            return tail;
        Node node = root;
        for(int level = shift; level > 0; level -= 5)
            node = (Node) node.array[(i >>> level) & 0x01f];
        return node.array;
    }
    throw new IndexOutOfBoundsException();
}

public Object valAt(Object key){
    //note - relies on ensureEditable in 2-arg valAt
    return valAt(key, null);
}

public Object valAt(Object key, Object notFound){
    ensureEditable();
    if(Util.isInteger(key))
    {
        int i = ((Number) key).intValue();
        if(i >= 0 && i < cnt)
            return nth(i);
    }
    return notFound;
}

public Object invoke(Object arg1) throws Exception{
    //note - relies on ensureEditable in nth
    if(Util.isInteger(arg1))
        return nth(((Number) arg1).intValue());
    throw new IllegalArgumentException("Key must be integer");
}

```

```

    }

    public Object nth(int i){
        ensureEditable();
        Object[] node = arrayFor(i);
        return node[i & 0x01f];
    }

    public Object nth(int i, Object notFound){
        if(i >= 0 && i < count())
            return nth(i);
        return notFound;
    }

    public TransientVector assocN(int i, Object val){
        ensureEditable();
        if(i >= 0 && i < cnt)
        {
            if(i >= tailoff())
            {
                tail[i & 0x01f] = val;
                return this;
            }

            root = doAssoc(shift, root, i, val);
            return this;
        }
        if(i == cnt)
            return conj(val);
        throw new IndexOutOfBoundsException();
    }

    public TransientVector assoc(Object key, Object val){
        //note - relies on ensureEditable in assocN
        if(Util.isInteger(key))
        {
            int i = ((Number) key).intValue();
            return assocN(i, val);
        }
        throw new IllegalArgumentException("Key must be integer");
    }

    private Node doAssoc(int level, Node node, int i, Object val){
        node = ensureEditable(node);
        Node ret = node;
        if(level == 0)
        {
            ret.array[i & 0x01f] = val;
        }
        else

```

```

    {
        int subidx = (i >>> level) & 0x01f;
        ret.array[subidx] =
            doAssoc(level - 5, (Node) node.array[subidx], i, val);
    }
    return ret;
}

public TransientVector pop(){
    ensureEditable();
    if(cnt == 0)
        throw new IllegalStateException("Can't pop empty vector");
    if(cnt == 1)
    {
        cnt = 0;
        return this;
    }
    int i = cnt - 1;
    //pop in tail?
    if((i & 0x01f) > 0)
    {
        --cnt;
        return this;
    }

    Object[] newtail = arrayFor(cnt - 2);

    Node newroot = popTail(shift, root);
    int newshift = shift;
    if(newroot == null)
    {
        newroot = new Node(root.edit);
    }
    if(shift > 5 && newroot.array[1] == null)
    {
        newroot = ensureEditable((Node) newroot.array[0]);
        newshift -= 5;
    }
    root = newroot;
    shift = newshift;
    --cnt;
    tail = newtail;
    return this;
}

private Node popTail(int level, Node node){
    node = ensureEditable(node);
    int subidx = ((cnt - 2) >>> level) & 0x01f;
    if(level > 5)
    {

```

```

        Node newchild =
            popTail(level - 5, (Node) node.array[subidx]);
        if(newchild == null && subidx == 0)
            return null;
        else
        {
            Node ret = node;
            ret.array[subidx] = newchild;
            return ret;
        }
    }
    else if(subidx == 0)
        return null;
    else
    {
        Node ret = node;
        ret.array[subidx] = null;
        return ret;
    }
}
*/
static public void main(String[] args){
    if(args.length != 3)
    {
        System.err.println("Usage: PersistentVector size writes reads");
        return;
    }
    int size = Integer.parseInt(args[0]);
    int writes = Integer.parseInt(args[1]);
    int reads = Integer.parseInt(args[2]);
//    Vector v = new Vector(size);
//    ArrayList v = new ArrayList(size);
//    v.setSize(size);
//    PersistentArray p = new PersistentArray(size);
    PersistentVector p = PersistentVector.EMPTY;
//    MutableVector mp = p.mutable();

    for(int i = 0; i < size; i++)
    {
        v.add(i);
//        v.set(i, i);
//        p = p.set(i, 0);
        p = p.cons(i);
//        mp = mp.conj(i);
    }

    Random rand;

    rand = new Random(42);
}

```

```

long tv = 0;
System.out.println("ArrayList");
long startTime = System.nanoTime();
for(int i = 0; i < writes; i++)
{
{
v.set(rand.nextInt(size), i);
}
for(int i = 0; i < reads; i++)
{
tv += (Integer) v.get(rand.nextInt(size));
}
long estimatedTime = System.nanoTime() - startTime;
System.out.println("time: " + estimatedTime / 1000000);
System.out.println("PersistentVector");
rand = new Random(42);
startTime = System.nanoTime();
long tp = 0;

// PersistentVector oldp = p;
//Random rand2 = new Random(42);

MutableVector mp = p.mutable();
for(int i = 0; i < writes; i++)
{
//
p = p.assocN(rand.nextInt(size), i);
mp = mp.assocN(rand.nextInt(size), i);
// mp = mp.assoc(rand.nextInt(size), i);
//dummy set to force perverse branching
//oldp = oldp.assocN(rand2.nextInt(size), i);
}
for(int i = 0; i < reads; i++)
{
//
tp += (Integer) p.nth(rand.nextInt(size));
tp += (Integer) mp.nth(rand.nextInt(size));
}
// p = mp.immutable();
//mp.cons(42);
estimatedTime = System.nanoTime() - startTime;
System.out.println("time: " + estimatedTime / 1000000);
for(int i = 0; i < size / 2; i++)
{
//
mp = mp.pop();
p = p.pop();
v.remove(v.size() - 1);
}
p = (PersistentVector) mp.immutable();
//mp.pop(); //should fail
for(int i = 0; i < size / 2; i++)
{
tp += (Integer) p.nth(i);
}

```

```

        tv += (Integer) v.get(i);
    }
    System.out.println("Done: " + tv + ", " + tp);

}
// */
}

```

9.87 ProxyHandler.java

(InvocationHandler [1723])
— ProxyHandler.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Oct 4, 2007 */

package clojure.lang;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

public class ProxyHandler implements InvocationHandler{
//method-name->fn
final IPersistentMap fns;

public ProxyHandler(IPersistentMap fns){
    this.fns = fns;
}

public Object invoke(Object proxy, Method method, Object[] args)
throws Throwable{
    Class rt = method.getReturnType();
    IFn fn = (IFn) fns.valAt(method.getName());
    if(fn == null)
    {
        if(rt == Void.TYPE)
            return null;
        else if(method.getName().equals("equals"))
        {
            return proxy == args[0];
        }
        else if(method.getName().equals("hashCode"))
        {

```

```

        return System.identityHashCode(proxy);
    }
    else if(method.getName().equals("toString"))
    {
        return "Proxy: " + System.identityHashCode(proxy);
    }
    throw new UnsupportedOperationException();
}
Object ret = fn.applyTo(ArraySeq.create(args));
if(rt == Void.TYPE)
    return null;
else if(rt.isPrimitive())
{
    if(rt == Character.TYPE)
        return ret;
    else if(rt == Integer.TYPE)
        return ((Number) ret).intValue();
    else if(rt == Long.TYPE)
        return ((Number) ret).longValue();
    else if(rt == Float.TYPE)
        return ((Number) ret).floatValue();
    else if(rt == Double.TYPE)
        return ((Number) ret).doubleValue();
    else if(rt == Boolean.TYPE && !(ret instanceof Boolean))
        return ret == null ? Boolean.FALSE : Boolean.TRUE;
    else if(rt == Byte.TYPE)
        return (byte) ((Number) ret).intValue();
    else if(rt == Short.TYPE)
        return (short) ((Number) ret).intValue();
}
return ret;
}
}

```

9.88 Range.java

(ASeq [571]) (IReduce [804]) (Counted [768])
 — Range.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Apr 1, 2008 */

package clojure.lang;

```

```
public class Range extends ASeq implements IReduce, Counted{
    final int end;
    final int n;

    public Range(int start, int end){
        this.end = end;
        this.n = start;
    }

    public Range(IPersistentMap meta, int start, int end){
        super(meta);
        this.end = end;
        this.n = start;
    }

    public Obj withMeta(IPersistentMap meta){
        if(meta == meta())
            return this;
        return new Range(meta(), end, n);
    }

    public Object first(){
        return n;
    }

    public ISeq next(){
        if(n < end-1)
            return new Range(_meta, n + 1, end);
        return null;
    }

    public Object reduce(IFn f) throws Exception{
        Object ret = n;
        for(int x = n+1;x < end;x++)
            ret = f.invoke(ret, x);
        return ret;
    }

    public Object reduce(IFn f, Object start) throws Exception{
        Object ret = f.invoke(start,n);
        for(int x = n+1;x < end;x++)
            ret = f.invoke(ret, x);
        return ret;
    }

    public int count() {
        return end - n;
    }
}
```

9.89 Ratio.java

(Number [1723]) (Comparable [1723])
— Ratio.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 31, 2008 */

package clojure.lang;

import java.math.BigInteger;
import java.math.BigDecimal;
import java.math.MathContext;

public class Ratio extends Number implements Comparable{
    final public BigInteger numerator;
    final public BigInteger denominator;

    public Ratio(BigInteger numerator, BigInteger denominator){
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public boolean equals(Object arg0){
        return arg0 != null
            && arg0 instanceof Ratio
            && ((Ratio) arg0).numerator.equals(numerator)
            && ((Ratio) arg0).denominator.equals(denominator);
    }

    public int hashCode(){
        return numerator.hashCode() ^ denominator.hashCode();
    }

    public String toString(){
        return numerator.toString() + "/" + denominator.toString();
    }

    public int intValue(){
        return (int) doubleValue();
    }

    public long longValue(){
        return BigIntegerValue().longValue();
    }
```

```

    }

    public float floatValue(){
        return (float)doubleValue();
    }

    public double doubleValue(){
        return decimalValue(MathContext.DECIMAL64).doubleValue();
    }

    public BigDecimal decimalValue(){
        return decimalValue(MathContext.UNLIMITED);
    }

    public BigDecimal decimalValue(MathContext mc){
        BigDecimal numerator = new BigDecimal(this.numerator);
        BigDecimal denominator = new BigDecimal(this.denominator);

        return numerator.divide(denominator, mc);
    }

    public BigInteger bigIntegerValue(){
        return numerator.divide(denominator);
    }

    public int compareTo(Object o){
        Number other = (Number)o;
        return Numbers.compare(this, other);
    }
}

```

9.90 Ref.java

(ARef [553]) (IFn [774]) (Comparable [1723]) (IRef [805])
 — Ref.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 25, 2007 */

package clojure.lang;

import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicLong;
import java.util.concurrent.locks.ReentrantReadWriteLock;

```

```
public class Ref extends ARef implements IFn, Comparable<Ref>, IRef{
    public int compareTo(Ref ref) {
        if(this.id == ref.id)
            return 0;
        else if(this.id < ref.id)
            return -1;
        else
            return 1;
    }

    public int getMinHistory(){
        return minHistory;
    }

    public Ref setMinHistory(int minHistory){
        this.minHistory = minHistory;
        return this;
    }

    public int getMaxHistory(){
        return maxHistory;
    }

    public Ref setMaxHistory(int maxHistory){
        this.maxHistory = maxHistory;
        return this;
    }

    public static class TVal{
        Object val;
        long point;
        long msecs;
        TVal prior;
        TVal next;

        TVal(Object val, long point, long msecs, TVal prior){
            this.val = val;
            this.point = point;
            this.msecs = msecs;
            this.prior = prior;
            this.next = prior.next;
            this.prior.next = this;
            this.next.prior = this;
        }

        TVal(Object val, long point, long msecs){
            this.val = val;
            this.point = point;
            this.msecs = msecs;
        }
    }
}
```

```

        this.next = this;
        this.prior = this;
    }

}

TVal tvals;
final AtomicInteger faults;
final ReentrantReadWriteLock lock;
LockingTransaction.Info tinfo;
//IFn validator;
final long id;

volatile int minHistory = 0;
volatile int maxHistory = 10;

static final AtomicLong ids = new AtomicLong();

public Ref(Object initVal) throws Exception{
    this(initVal, null);
}

public Ref(Object initVal,IPersistentMap meta) throws Exception{
    super(meta);
    this.id = ids.getAndIncrement();
    this.faults = new AtomicInteger();
    this.lock = new ReentrantReadWriteLock();
    tvals = new TVal(initVal, 0, System.currentTimeMillis());
}

//the latest val

// ok out of transaction
Object currentValue(){
    try
    {
        lock.readLock().lock();
        if(tvals != null)
            return tvals.val;
        throw new IllegalStateException(
            this.toString() + " is unbound.");
    }
    finally
    {
        lock.readLock().unlock();
    }
}

/*

```

```
public Object deref(){
    LockingTransaction t = LockingTransaction.getRunning();
    if(t == null)
        return currentValue();
    return t.doGet(this);
}

//void validate(IFn vf, Object val){
//    try{
//        if(vf != null && !RT.booleanCast(vf.invoke(val)))
//            throw new IllegalStateException("Invalid ref state");
//    }
//    catch(RuntimeException re)
//    {
//        throw re;
//    }
//    catch(Exception e)
//    {
//        throw new IllegalStateException("Invalid ref state", e);
//    }
//}
//
//public void setValidator(IFn vf){
//    try
//    {
//        lock.writeLock().lock();
//        validate(vf,currentVal());
//        validator = vf;
//    }
//    finally
//    {
//        lock.writeLock().unlock();
//    }
//}
//
//public IFn getValidator(){
//    try
//    {
//        lock.readLock().lock();
//        return validator;
//    }
//    finally
//    {
//        lock.readLock().unlock();
//    }
//}

public Object set(Object val){
    return LockingTransaction.getEx().doSet(this, val);
}
```

```
public Object commute(IFn fn, ISeq args) throws Exception{
    return LockingTransaction.getEx().doCommute(this, fn, args);
}

public Object alter(IFn fn, ISeq args) throws Exception{
    LockingTransaction t = LockingTransaction.getEx();
    return t.doSet(this, fn.applyTo(RT.cons(t.doGet(this), args)));
}

public void touch(){
    LockingTransaction.getEx().doEnsure(this);
}

<賓客/>
boolean isBound(){
    try
    {
        lock.readLock().lock();
        return tvals != null;
    }
    finally
    {
        lock.readLock().unlock();
    }
}

public void trimHistory(){
    try
    {
        lock.writeLock().lock();
        if(tvals != null)
        {
            tvals.next = tvals;
            tvals.prior = tvals;
        }
    }
    finally
    {
        lock.writeLock().unlock();
    }
}

public int getHistoryCount(){
    try
    {
        lock.writeLock().lock();
        return histCount();
    }
}
```

```
        finally
        {
            lock.writeLock().unlock();
        }
    }

    int histCount(){
        if(tvals == null)
            return 0;
        else
        {
            int count = 0;
            for(TVal tv = tvals.next;tv != tvals;tv = tv.next)
                count++;
            return count;
        }
    }

    final public IFn fn(){
        return (IFn) deref();
    }

    public Object call() throws Exception{
        return invoke();
    }

    public void run(){
        try
        {
            invoke();
        }
        catch(Exception e)
        {
            throw new RuntimeException(e);
        }
    }

    public Object invoke()
    throws Exception{
        return fn().invoke();
    }

    public Object invoke(Object arg1)
    throws Exception{
        return fn().invoke(arg1);
    }

    public Object invoke(Object arg1, Object arg2)
    throws Exception{
        return fn().invoke(arg1, arg2);
    }
}
```

```
}
```

```
public Object invoke(Object arg1, Object arg2, Object arg3)
    throws Exception{
    return fn().invoke(arg1, arg2, arg3);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
    Object arg4)
    throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
    Object arg4, Object arg5)
    throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4, arg5);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
    Object arg4, Object arg5, Object arg6)
    throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
    Object arg4, Object arg5, Object arg6,
    Object arg7)
    throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
    Object arg4, Object arg5, Object arg6,
    Object arg7, Object arg8)
    throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
    Object arg4, Object arg5, Object arg6,
    Object arg7, Object arg8, Object arg9)
    throws Exception{
    return
        fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
    Object arg4, Object arg5, Object arg6,
    Object arg7, Object arg8, Object arg9,
```

```
        Object arg10)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
```

```

        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12,
        Object arg13, Object arg14, Object arg15)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15, arg16);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15, arg16,
               arg17);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15, arg16,
               arg17, arg18);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18);
}

```

```
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12,
        Object arg13, Object arg14, Object arg15,
        Object arg16, Object arg17, Object arg18,
        Object arg19)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15, arg16,
               arg17, arg18, arg19);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15, arg16,
               arg17, arg18, arg19, arg20);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20, Object... args)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15, arg16,
               arg17, arg18, arg19, arg20, args);
}

public Object applyTo(ISeq arglist) throws Exception{
    return AFn.applyToHelper(this, arglist);
}
```

9.91 Reflector.java

— Reflector.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Apr 19, 2006 */

package clojure.lang;

import java.lang.reflect.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Arrays;

public class Reflector{

    public static Object invokeInstanceMethod(Object target,
                                              String methodName,
                                              Object[] args)
        throws Exception{
        try
        {
            Class c = target.getClass();
            List methods = getMethods(c, args.length, methodName, false);
            return invokeMatchingMethod(methodName, methods, target, args);
        }
        catch(InvocationTargetException e)
        {
            if(e.getCause() instanceof Exception)
                throw (Exception) e.getCause();
            else if(e.getCause() instanceof Error)
                throw (Error) e.getCause();
            throw e;
        }
    }

    private static String noMethodReport(String methodName, Object target){
        return "No matching method found: " + methodName
               + (target==null?"":" for " + target.getClass());
    }

    static Object invokeMatchingMethod(String methodName,
                                       List methods,
                                       Object target,
                                       Object[] args)
        throws Exception{
```

```

Method m = null;
Object[] boxedArgs = null;
if(methods.isEmpty())
{
    throw new IllegalArgumentException(
        noMethodReport(methodName,target));
}
else if(methods.size() == 1)
{
    m = (Method) methods.get(0);
    boxedArgs = boxArgs(m.getParameterTypes(), args);
}
else //overloaded w/same arity
{
    Method foundm = null;
    for(Iterator i = methods.iterator(); i.hasNext();)
    {
        m = (Method) i.next();

        Class[] params = m.getParameterTypes();
        if(isCongruent(params, args))
        {
            if(foundm == null ||
                Compiler.subsumes(params,
                    foundm.getParameterTypes()))
            {
                foundm = m;
                boxedArgs = boxArgs(params, args);
            }
        }
        m = foundm;
    }
    if(m == null)
        throw new IllegalArgumentException(
            noMethodReport(methodName,target));

    if(!Modifier.isPublic(m.getDeclaringClass().getModifiers()))
    {
        //public method of non-public class, try to find it in
        //hierarchy
        Method oldm = m;
        m = getAsMethodOfPublicBase(m.getDeclaringClass(), m);
        if(m == null)
            throw new IllegalArgumentException(
                "Can't call public method of non-public class: " +
                oldm.toString());
    }
}
try
{

```

```

        return prepRet(m.getReturnType(), m.invoke(target, boxedArgs));
    }
    catch(InvocationTargetException e)
    {
        if(e.getCause() instanceof Exception)
            throw (Exception) e.getCause();
        else if(e.getCause() instanceof Error)
            throw (Error) e.getCause();
        throw e;
    }
}

public static Method getAsMethodOfPublicBase(Class c, Method m){
    for(Class iface : c.getInterfaces())
    {
        for(Method im : iface.getMethods())
        {
            if(im.getName().equals(m.getName())
                && Arrays.equals(m.getParameterTypes(),
                                  im.getParameterTypes()))
            {
                return im;
            }
        }
    }
    Class sc = c.getSuperclass();
    if(sc == null)
        return null;
    for(Method scm : sc.getMethods())
    {
        if(scm.getName().equals(m.getName())
            && Arrays.equals(m.getParameterTypes(),
                              scm.getParameterTypes())
            && Modifier.isPublic(scm.getDeclaringClass()
                                  .getModifiers()))
        {
            return scm;
        }
    }
    return getAsMethodOfPublicBase(sc, m);
}

public static Object invokeConstructor(Class c, Object[] args)
throws Exception{
    try
    {
        Constructor[] allctors = c.getConstructors();
        ArrayList ctors = new ArrayList();
        for(int i = 0; i < allctors.length; i++)

```

```

    {
        Constructor ctor = allctors[i];
        if(ctor.getParameterTypes().length == args.length)
            ctors.add(ctor);
    }
    if(ctors.isEmpty())
    {
        throw new IllegalArgumentException("No matching ctor found"
            + " for " + c);
    }
    else if(ctors.size() == 1)
    {
        Constructor ctor = (Constructor) ctors.get(0);
        return
            ctor.newInstance(
                boxArgs(ctor.getParameterTypes(), args));
    }
    else //overloaded w/same arity
    {
        for(Iterator iterator = ctors.iterator();
            iterator.hasNext());
        {
            Constructor ctor = (Constructor) iterator.next();
            Class[] params = ctor.getParameterTypes();
            if(isCongruent(params, args))
            {
                Object[] boxedArgs = boxArgs(params, args);
                return ctor.newInstance(boxedArgs);
            }
        }
        throw new IllegalArgumentException("No matching ctor found"
            + " for " + c);
    }
}
catch(InvocationTargetException e)
{
    if(e.getCause() instanceof Exception)
        throw (Exception) e.getCause();
    else if(e.getCause() instanceof Error)
        throw (Error) e.getCause();
    throw e;
}
}

public static Object invokeStaticMethodVariadic(String className,
                                                String methodName,
                                                Object... args)
throws Exception{
    return invokeStaticMethod(className, methodName, args);
}

```

```

}

public static Object invokeStaticMethod(String className,
                                       String methodName,
                                       Object[] args)
throws Exception{
    Class c = RT.classForName(className);
    try
    {
        return invokeStaticMethod(c, methodName, args);
    }
    catch(InvocationTargetException e)
    {
        if(e.getCause() instanceof Exception)
            throw (Exception) e.getCause();
        else if(e.getCause() instanceof Error)
            throw (Error) e.getCause();
        throw e;
    }
}

public static Object invokeStaticMethod(Class c,
                                       String methodName,
                                       Object[] args)
throws Exception{
    if(methodName.equals("new"))
        return invokeConstructor(c, args);
    List methods = getMethods(c, args.length, methodName, true);
    return invokeMatchingMethod(methodName, methods, null, args);
}

public static Object getStaticField(String className,
                                   String fieldName)
throws Exception{
    Class c = RT.classForName(className);
    return getStaticField(c, fieldName);
}

public static Object getStaticField(Class c,
                                   String fieldName)
throws Exception{
//    if(fieldName.equals("class"))
//        return c;
    Field f = getField(c, fieldName, true);
    if(f != null)
    {
        return prepRet(f.getType(), f.get(null));
    }
    throw new IllegalArgumentException(
        "No matching field found: " + fieldName
    );
}

```

```
        + " for " + c);
    }

    public static Object setStaticField(String className,
                                       String fieldName,
                                       Object val)
    throws Exception{
    Class c = RT.classForName(className);
    return setStaticField(c, fieldName, val);
}

public static Object setStaticField(Class c,
                                   String fieldName,
                                   Object val)
throws Exception{
    Field f = getField(c, fieldName, true);
    if(f != null)
    {
        f.set(null, boxArg(f.getType(), val));
        return val;
    }
    throw new IllegalArgumentException(
        "No matching field found: " + fieldName
        + " for " + c);
}

public static Object getInstanceField(Object target,
                                      String fieldName)
throws Exception{
    Class c = target.getClass();
    Field f = getField(c, fieldName, false);
    if(f != null)
    {
        return prepRet(f.getType(), f.get(target));
    }
    throw new IllegalArgumentException(
        "No matching field found: " + fieldName
        + " for " + target.getClass());
}

public static Object setInstanceField(Object target,
                                      String fieldName,
                                      Object val)
throws Exception{
    Class c = target.getClass();
    Field f = getField(c, fieldName, false);
    if(f != null)
    {
        f.set(target, boxArg(f.getType(), val));
        return val;
    }
}
```

```

        }
        throw new IllegalArgumentException(
            "No matching field found: " + fieldName
            + " for " + target.getClass());
    }

    public static Object invokeNoArgInstanceMember(Object target,
                                                   String name)
    throws Exception{
        //favor method over field
        List meths = getMethods(target.getClass(), 0, name, false);
        if(meths.size() > 0)
            return
                invokeMatchingMethod(name, meths, target, RT.EMPTY_ARRAY);
        else
            return getInstanceField(target, name);
    }

    public static Object invokeInstanceMember(Object target,
                                              String name)
    throws Exception{
        //check for field first
        Class c = target.getClass();
        Field f = getField(c, name, false);
        if(f != null) //field get
        {
            return prepRet(f.getType(), f.get(target));
        }
        return invokeInstanceMethod(target, name, RT.EMPTY_ARRAY);
    }

    public static Object invokeInstanceMember(String name,
                                              Object target,
                                              Object arg1)
    throws Exception{
        //check for field first
        Class c = target.getClass();
        Field f = getField(c, name, false);
        if(f != null) //field set
        {
            f.set(target, boxArg(f.getType(), arg1));
            return arg1;
        }
        return invokeInstanceMethod(target, name, new Object[]{arg1});
    }

    public static Object invokeInstanceMember(String name,
                                              Object target,
                                              Object... args)
    throws Exception{

```

```
        return invokeInstanceMethod(target, name, args);
    }

static public Field getField(Class c,
                            String name,
                            boolean getStatics){
    Field[] allfields = c.getFields();
    for(int i = 0; i < allfields.length; i++)
    {
        if(name.equals(allfields[i].getName())
           && Modifier.isStatic(allfields[i].getModifiers())
           == getStatics)
            return allfields[i];
    }
    return null;
}

static public List getMethods(Class c,
                             int arity,
                             String name,
                             boolean getStatics){
    Method[] allmethods = c.getMethods();
    ArrayList methods = new ArrayList();
    ArrayList bridgeMethods = new ArrayList();
    for(int i = 0; i < allmethods.length; i++)
    {
        Method method = allmethods[i];
        if(name.equals(method.getName())
           && Modifier.isStatic(method.getModifiers()) == getStatics
           && method.getParameterTypes().length == arity)
        {
            try
            {
                if(method.isBridge()
                   && c.getMethod(method.getName(),
                                  method.getParameterTypes())
                   .equals(method))
                    bridgeMethods.add(method);
                else
                    methods.add(method);
            }
            catch(NoSuchMethodException e)
            {
            }
        }
    }
    //        && (!method.isBridge()
    //          || (c == StringBuilder.class &&
    //              c.getMethod(method.getName(),
    //                          method.getParameterTypes())
    //
```

```

//           .equals(method)))
//           {
//             methods.add(allmethods[i]);
//           }
}

if(methods.isEmpty())
  methods.addAll(bridgeMethods);

if(!getStatics && c.isInterface())
{
  allmethods = Object.class.getMethods();
  for(int i = 0; i < allmethods.length; i++)
  {
    if(name.equals(allmethods[i].getName())
      && Modifier.isStatic(allmethods[i].getModifiers())
      == getStatics
      && allmethods[i].getParameterTypes().length == arity)
    {
      methods.add(allmethods[i]);
    }
  }
}
return methods;
}

static Object boxArg(Class paramType, Object arg){
  if(!paramType.isPrimitive())
    return paramType.cast(arg);
  else if(paramType == boolean.class)
    return Boolean.class.cast(arg);
  else if(paramType == char.class)
    return Character.class.cast(arg);
  else if(arg instanceof Number)
  {
    Number n = (Number) arg;
    if(paramType == int.class)
      return n.intValue();
    else if(paramType == float.class)
      return n.floatValue();
    else if(paramType == double.class)
      return n.doubleValue();
    else if(paramType == long.class)
      return n.longValue();
    else if(paramType == short.class)
      return n.shortValue();
    else if(paramType == byte.class)
      return n.byteValue();
  }
}

```

```
throw new IllegalArgumentException(
    "Unexpected param type, expected: " + paramType +
    ", given: " + arg.getClass().getName());
}

static Object[] boxArgs(Class[] params, Object[] args){
    if(params.length == 0)
        return null;
    Object[] ret = new Object[params.length];
    for(int i = 0; i < params.length; i++)
    {
        Object arg = args[i];
        Class paramType = params[i];
        ret[i] = boxArg(paramType, arg);
    }
    return ret;
}

static public boolean paramArgTypeMatch(Class paramType,
                                         Class argType){
    if(argType == null)
        return !paramType.isPrimitive();
    if(paramType == argType || paramType.isAssignableFrom(argType))
        return true;
    if(paramType == int.class)
        return argType == Integer.class
            || argType == long.class
            || argType == Long.class;// || argType==FixNum.class;
    else if(paramType == float.class)
        return argType == Float.class
            || argType == double.class;
    else if(paramType == double.class)
        return argType == Double.class
            || argType == float.class;
// || argType == DoubleNum.class;
    else if(paramType == long.class)
        return argType == Long.class
            || argType == int.class;
// || argType == BigNum.class;
    else if(paramType == char.class)
        return argType == Character.class;
    else if(paramType == short.class)
        return argType == Short.class;
    else if(paramType == byte.class)
        return argType == Byte.class;
    else if(paramType == boolean.class)
        return argType == Boolean.class;
    return false;
}
```

```

static boolean isCongruent(Class[] params, Object[] args){
    boolean ret = false;
    if(args == null)
        return params.length == 0;
    if(params.length == args.length)
    {
        ret = true;
        for(int i = 0; ret && i < params.length; i++)
        {
            Object arg = args[i];
            Class argType = (arg == null) ? null : arg.getClass();
            Class paramType = params[i];
            ret = paramArgTypeMatch(paramType, argType);
        }
    }
    return ret;
}

public static Object prepRet(Class c, Object x){
    if (!(c.isPrimitive() || c == Boolean.class))
        return x;
    if(x instanceof Boolean)
        return ((Boolean) x)?Boolean.TRUE:Boolean.FALSE;
    else if(x instanceof Integer)
    {
        return ((Integer)x).longValue();
    }
    else if(x instanceof Float)
        return Double.valueOf(((Float) x).doubleValue());
    return x;
}
}

```

9.92 Repl.java

— Repl.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Oct 18, 2007 */

package clojure.lang;

import clojure.main;

```

```

public class Repl {

    public static void main(String[] args) throws Exception{
        main.legacy_repl(args);
    }
}

```

—————

9.93 RestFn.java

(AFunction [519])

— RestFn.java —

```

/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

public abstract class RestFn extends AFunction{

    abstract public int getRequiredArity();

    protected Object doInvoke(Object args)
        throws Exception{
            return null;
    }

    protected Object doInvoke(Object arg1, Object args)
        throws Exception{
            return null;
    }

    protected Object doInvoke(Object arg1, Object arg2, Object args)
        throws Exception{
            return null;
    }

    protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                                Object args)
        throws Exception{
            return null;
    }

    protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                                Object arg4, Object args)
        throws Exception{

```

```
        return null;
    }

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9,
                         Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9,
                         Object arg10, Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9,
                         Object arg10, Object arg11, Object args)
```

```
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9,
                         Object arg10, Object arg11, Object arg12,
                         Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9,
                         Object arg10, Object arg11, Object arg12,
                         Object arg13, Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9,
                         Object arg10, Object arg11, Object arg12,
                         Object arg13, Object arg14, Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9,
                         Object arg10, Object arg11, Object arg12,
                         Object arg13, Object arg14, Object arg15,
                         Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                         Object arg4, Object arg5, Object arg6,
                         Object arg7, Object arg8, Object arg9,
                         Object arg10, Object arg11, Object arg12,
                         Object arg13, Object arg14, Object arg15,
                         Object arg16, Object args)
throws Exception{
    return null;
}
```

```

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                        Object arg4, Object arg5, Object arg6,
                        Object arg7, Object arg8, Object arg9,
                        Object arg10, Object arg11, Object arg12,
                        Object arg13, Object arg14, Object arg15,
                        Object arg16, Object arg17, Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                        Object arg4, Object arg5, Object arg6,
                        Object arg7, Object arg8, Object arg9,
                        Object arg10, Object arg11, Object arg12,
                        Object arg13, Object arg14, Object arg15,
                        Object arg16, Object arg17, Object arg18,
                        Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                        Object arg4, Object arg5, Object arg6,
                        Object arg7, Object arg8, Object arg9,
                        Object arg10, Object arg11, Object arg12,
                        Object arg13, Object arg14, Object arg15,
                        Object arg16, Object arg17, Object arg18,
                        Object arg19, Object args)
throws Exception{
    return null;
}

protected Object doInvoke(Object arg1, Object arg2, Object arg3,
                        Object arg4, Object arg5, Object arg6,
                        Object arg7, Object arg8, Object arg9,
                        Object arg10, Object arg11, Object arg12,
                        Object arg13, Object arg14, Object arg15,
                        Object arg16, Object arg17, Object arg18,
                        Object arg19, Object arg20, Object args)
throws Exception{
    return null;
}

public Object applyTo(ISeq args) throws Exception{
    if(RT.boundedLength(args, getRequiredArity()) <= getRequiredArity())
    {
        return AFn.applyToHelper(this, Util.ret1(args,args = null));
    }
}

```

```
switch(getRequiredArity())
{
    case 0:
        return doInvoke(Util.ret1(args,args = null));
    case 1:
        return doInvoke(args.first()
                      , Util.ret1(args.next(),args=null));
    case 2:
        return doInvoke(args.first()
                      , (args = args.next()).first()
                      , Util.ret1(args.next(),args=null));
    case 3:
        return doInvoke(args.first()
                      , (args = args.next()).first()
                      , (args = args.next()).first()
                      , Util.ret1(args.next(),args=null));
    case 4:
        return doInvoke(args.first()
                      , (args = args.next()).first()
                      , (args = args.next()).first()
                      , (args = args.next()).first()
                      , Util.ret1(args.next(),args=null));
    case 5:
        return doInvoke(args.first()
                      , (args = args.next()).first()
                      , (args = args.next()).first()
                      , (args = args.next()).first()
                      , (args = args.next()).first()
                      , Util.ret1(args.next(),args=null));
    case 6:
        return doInvoke(args.first()
                      , (args = args.next()).first()
                      , Util.ret1(args.next(),args=null));
    case 7:
        return doInvoke(args.first()
                      , (args = args.next()).first()
                      , Util.ret1(args.next(),args=null));
    case 8:
        return doInvoke(args.first()
                      , (args = args.next()).first()
                      , (args = args.next()).first()
```



```
, (args = args.next()).first()
, Util.ret1(args.next(),args=null));
case 13:
    return doInvoke(args.first())
        , (args = args.next()).first()
        , Util.ret1(args.next(),args=null));
case 14:
    return doInvoke(args.first())
        , (args = args.next()).first()
        , Util.ret1(args.next(),args=null));
case 15:
    return doInvoke(args.first())
        , (args = args.next()).first()
        , (args = args.next()).first()
```



```
, (args = args.next()).first()
, Util.ret1(args.next(),args=null));
case 19:
    return doInvoke(args.first()
, (args = args.next()).first()
, Util.ret1(args.next(),args=null));
case 20:
    return doInvoke(args.first()
, (args = args.next()).first()
```

```
        , (args = args.next()).first()
        , (args = args.next()).first()
        , (args = args.next()).first()
        , Util.ret1(args.next(), args=null));
    }
    return throwArity(-1);
}

public Object invoke() throws Exception{
    switch(getRequiredArity())
    {
    case 0:
        return doInvoke(null);
    default:
        return throwArity(0);
    }
}

public Object invoke(Object arg1) throws Exception{
    switch(getRequiredArity())
    {
    case 0:
        return doInvoke(ArraySeq.create(
            Util.ret1(arg1, arg1 = null)));
    case 1:
        return doInvoke(Util.ret1(arg1, arg1 = null), null);
    default:
        return throwArity(1);
    }
}

public Object invoke(Object arg1, Object arg2)
throws Exception{
    switch(getRequiredArity())
    {
    case 0:
        return doInvoke(ArraySeq.create(
            Util.ret1(arg1, arg1 = null),
            Util.ret1(arg2, arg2 = null)));
    case 1:
        return doInvoke(Util.ret1(arg1, arg1 = null),
            ArraySeq.create(
                Util.ret1(arg2, arg2 = null)));
    case 2:
        return doInvoke(Util.ret1(arg1, arg1 = null),
            Util.ret1(arg2, arg2 = null), null);
    default:

```

```
        return throwArity(2);
    }

}

public Object invoke(Object arg1, Object arg2, Object arg3)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(
                Util.ret1(arg1, arg1 = null),
                Util.ret1(arg2, arg2 = null),
                Util.ret1(arg3, arg3 = null)));
        case 1:
            return doInvoke(Util.ret1(arg1, arg1 = null),
                ArraySeq.create(
                    Util.ret1(arg2, arg2 = null),
                    Util.ret1(arg3, arg3 = null)));
        case 2:
            return doInvoke(Util.ret1(arg1, arg1 = null),
                Util.ret1(arg2, arg2 = null),
                ArraySeq.create(
                    Util.ret1(arg3, arg3 = null)));
        case 3:
            return doInvoke(Util.ret1(arg1, arg1 = null),
                Util.ret1(arg2, arg2 = null),
                Util.ret1(arg3, arg3 = null),
                null);
        default:
            return throwArity(3);
    }
}

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(
                Util.ret1(arg1, arg1 = null),
                Util.ret1(arg2, arg2 = null),
                Util.ret1(arg3, arg3 = null),
                Util.ret1(arg4, arg4 = null)));
        case 1:
            return doInvoke(Util.ret1(arg1, arg1 = null),
                ArraySeq.create(
                    Util.ret1(arg2, arg2 = null),
```



```
        Util.ret1(arg5, arg5 = null)));
case 3:
    return doInvoke(Util.ret1(arg1, arg1 = null),
                  Util.ret1(arg2, arg2 = null),
                  Util.ret1(arg3, arg3 = null),
                  ArraySeq.create(
                      Util.ret1(arg4, arg4 = null),
                      Util.ret1(arg5, arg5 = null)));
case 4:
    return doInvoke(Util.ret1(arg1, arg1 = null),
                  Util.ret1(arg2, arg2 = null),
                  Util.ret1(arg3, arg3 = null),
                  Util.ret1(arg4, arg4 = null),
                  ArraySeq.create(
                      Util.ret1(arg5, arg5 = null)));
case 5:
    return doInvoke(Util.ret1(arg1, arg1 = null),
                  Util.ret1(arg2, arg2 = null),
                  Util.ret1(arg3, arg3 = null),
                  Util.ret1(arg4, arg4 = null),
                  Util.ret1(arg5, arg5 = null), null);
default:
    return throwArity(5);
}

}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6)
throws Exception{
    switch(getRequiredArity())
    {
case 0:
    return doInvoke(ArraySeq.create(
                    Util.ret1(arg1, arg1 = null),
                    Util.ret1(arg2, arg2 = null),
                    Util.ret1(arg3, arg3 = null),
                    Util.ret1(arg4, arg4 = null),
                    Util.ret1(arg5, arg5 = null),
                    Util.ret1(arg6, arg6 = null)));
case 1:
    return doInvoke(Util.ret1(arg1, arg1 = null),
                  ArraySeq.create(
                      Util.ret1(arg2, arg2 = null),
                      Util.ret1(arg3, arg3 = null),
                      Util.ret1(arg4, arg4 = null),
                      Util.ret1(arg5, arg5 = null),
                      Util.ret1(arg6, arg6 = null)));
case 2:
    return doInvoke(Util.ret1(arg1, arg1 = null),
```

```
        Util.ret1(arg2, arg2 = null),
        ArraySeq.create(
            Util.ret1(arg3, arg3 = null),
            Util.ret1(arg4, arg4 = null),
            Util.ret1(arg5, arg5 = null),
            Util.ret1(arg6, arg6 = null)));
    case 3:
        return doInvoke(Util.ret1(arg1, arg1 = null),
            Util.ret1(arg2, arg2 = null),
            Util.ret1(arg3, arg3 = null),
            ArraySeq.create(
                Util.ret1(arg4, arg4 = null),
                Util.ret1(arg5, arg5 = null),
                Util.ret1(arg6, arg6 = null)));
    case 4:
        return doInvoke(Util.ret1(arg1, arg1 = null),
            Util.ret1(arg2, arg2 = null),
            Util.ret1(arg3, arg3 = null),
            Util.ret1(arg4, arg4 = null),
            ArraySeq.create(
                Util.ret1(arg5, arg5 = null),
                Util.ret1(arg6, arg6 = null)));
    case 5:
        return doInvoke(Util.ret1(arg1, arg1 = null),
            Util.ret1(arg2, arg2 = null),
            Util.ret1(arg3, arg3 = null),
            Util.ret1(arg4, arg4 = null),
            Util.ret1(arg5, arg5 = null),
            ArraySeq.create(
                Util.ret1(arg6, arg6 = null)));
    case 6:
        return doInvoke(Util.ret1(arg1, arg1 = null),
            Util.ret1(arg2, arg2 = null),
            Util.ret1(arg3, arg3 = null),
            Util.ret1(arg4, arg4 = null),
            Util.ret1(arg5, arg5 = null),
            Util.ret1(arg6, arg6 = null),
            null);
    default:
        return throwArity(6);
    }
}

public Object invoke(Object arg1, Object arg2, Object arg3,
    Object arg4, Object arg5, Object arg6,
    Object arg7)
throws Exception{
    switch(getRequiredArity())
    {
```

```
case 0:
    return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                    arg5, arg6, arg7));
case 1:
    return doInvoke(arg1,
                    ArraySeq.create(arg2, arg3, arg4, arg5,
                                   arg6, arg7));
case 2:
    return doInvoke(arg1, arg2,
                    ArraySeq.create(arg3, arg4, arg5, arg6,
                                   arg7));
case 3:
    return doInvoke(arg1, arg2, arg3,
                    ArraySeq.create(arg4, arg5, arg6, arg7));
case 4:
    return doInvoke(arg1, arg2, arg3, arg4,
                    ArraySeq.create(arg5, arg6, arg7));
case 5:
    return doInvoke(arg1, arg2, arg3, arg4, arg5,
                    ArraySeq.create(arg6, arg7));
case 6:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                    ArraySeq.create(arg7));
case 7:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                   null);
default:
    return throwArity(7);
}

}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                            arg5, arg6, arg7, arg8));
        case 1:
            return doInvoke(arg1,
                            ArraySeq.create(arg2, arg3, arg4, arg5,
                                           arg6, arg7, arg8));
        case 2:
            return doInvoke(arg1, arg2,
                            ArraySeq.create(arg3, arg4, arg5, arg6,
                                           arg7, arg8));
        case 3:
```

```

        return doInvoke(arg1, arg2, arg3,
                        ArraySeq.create(arg4, arg5, arg6, arg7,
                        arg8));
    case 4:
        return doInvoke(arg1, arg2, arg3, arg4,
                        ArraySeq.create(arg5, arg6, arg7, arg8));
    case 5:
        return doInvoke(arg1, arg2, arg3, arg4, arg5,
                        ArraySeq.create(arg6, arg7, arg8));
    case 6:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                        ArraySeq.create(arg7, arg8));
    case 7:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        ArraySeq.create(arg8));
    case 8:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, null);
    default:
        return throwArity(8);
    }
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                             arg5, arg6, arg7, arg8, arg9));
        case 1:
            return doInvoke(arg1,
                            ArraySeq.create(arg2, arg3, arg4, arg5,
                            arg6, arg7, arg8, arg9));
        case 2:
            return doInvoke(arg1, arg2,
                            ArraySeq.create(arg3, arg4, arg5, arg6,
                            arg7, arg8, arg9));
        case 3:
            return doInvoke(arg1, arg2, arg3,
                            ArraySeq.create(arg4, arg5, arg6, arg7,
                            arg8, arg9));
        case 4:
            return doInvoke(arg1, arg2, arg3, arg4,
                            ArraySeq.create(arg5, arg6, arg7, arg8,
                            arg9));
        case 5:
            return doInvoke(arg1, arg2, arg3, arg4, arg5,
                            ArraySeq.create(arg6, arg7, arg8, arg9));
    }
}

```

```
        return doInvoke(arg1, arg2, arg3, arg4, arg5,
                        ArraySeq.create(arg6, arg7, arg8, arg9));
    case 6:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                        ArraySeq.create(arg7, arg8, arg9));
    case 7:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        ArraySeq.create(arg8, arg9));
    case 8:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, ArraySeq.create(arg9));
    case 9:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, null);
    default:
        return throwArity(9);
    }

}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                             arg5, arg6, arg7, arg8, arg9, arg10));
        case 1:
            return doInvoke(arg1,
                           ArraySeq.create(arg2, arg3, arg4, arg5,
                                         arg6, arg7, arg8, arg9, arg10));
        case 2:
            return doInvoke(arg1, arg2,
                           ArraySeq.create(arg3, arg4, arg5, arg6,
                                         arg7, arg8, arg9, arg10));
        case 3:
            return doInvoke(arg1, arg2, arg3,
                           ArraySeq.create(arg4, arg5, arg6, arg7,
                                         arg8, arg9, arg10));
        case 4:
            return doInvoke(arg1, arg2, arg3, arg4,
                           ArraySeq.create(arg5, arg6, arg7, arg8,
                                         arg9, arg10));
        case 5:
            return doInvoke(arg1, arg2, arg3, arg4, arg5,
                           ArraySeq.create(arg6, arg7, arg8, arg9,
                                         arg10));
    }
}
```

```

        case 6:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                            ArraySeq.create(arg7, arg8, arg9, arg10));
        case 7:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            ArraySeq.create(arg8, arg9, arg10));
        case 8:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, ArraySeq.create(arg9, arg10));
        case 9:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, ArraySeq.create(arg10));
        case 10:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, null);
        default:
            return throwArity(10);
    }

}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                             arg5, arg6, arg7, arg8, arg9, arg10,
                                             arg11));
        case 1:
            return doInvoke(arg1,
                           ArraySeq.create(arg2, arg3, arg4, arg5,
                                         arg6, arg7, arg8, arg9, arg10, arg11));
        case 2:
            return doInvoke(arg1, arg2,
                           ArraySeq.create(arg3, arg4, arg5, arg6,
                                         arg7, arg8, arg9, arg10, arg11));
        case 3:
            return doInvoke(arg1, arg2, arg3,
                           ArraySeq.create(arg4, arg5, arg6, arg7,
                                         arg8, arg9, arg10, arg11));
        case 4:
            return doInvoke(arg1, arg2, arg3, arg4,
                           ArraySeq.create(arg5, arg6, arg7, arg8,
                                         arg9, arg10, arg11));
        case 5:
            return doInvoke(arg1, arg2, arg3, arg4, arg5,
                           ...

```

```

        ArraySeq.create(arg6, arg7, arg8, arg9,
                        arg10, arg11));
    case 6:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                        ArraySeq.create(arg7, arg8, arg9, arg10,
                                      arg11));
    case 7:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        ArraySeq.create(arg8, arg9, arg10, arg11));
    case 8:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, ArraySeq.create(arg9, arg10, arg11));
    case 9:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, ArraySeq.create(arg10, arg11));
    case 10:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, ArraySeq.create(arg11));
    case 11:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, null);
    default:
        return throwArity(11);
    }
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12)
throws Exception{
    switch(getRequiredArity())
    {
    case 0:
        return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                         arg5, arg6, arg7, arg8, arg9, arg10,
                                         arg11, arg12));
    case 1:
        return doInvoke(arg1,
                      ArraySeq.create(arg2, arg3, arg4, arg5,
                                     arg6, arg7, arg8, arg9, arg10, arg11,
                                     arg12));
    case 2:
        return doInvoke(arg1, arg2,
                      ArraySeq.create(arg3, arg4, arg5, arg6,
                                     arg7, arg8, arg9, arg10, arg11, arg12));
    case 3:
        return doInvoke(arg1, arg2, arg3,
                      ArraySeq.create(arg4, arg5, arg6, arg7,
                                     arg8, arg9, arg10, arg11, arg12));
    }
}

```

```

                        arg8, arg9, arg10, arg11, arg12));
case 4:
    return doInvoke(arg1, arg2, arg3, arg4,
                    ArraySeq.create(arg5, arg6, arg7, arg8,
                                   arg9, arg10, arg11, arg12));
case 5:
    return doInvoke(arg1, arg2, arg3, arg4, arg5,
                    ArraySeq.create(arg6, arg7, arg8, arg9,
                                   arg10, arg11, arg12));
case 6:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                    ArraySeq.create(arg7, arg8, arg9, arg10,
                                   arg11, arg12));
case 7:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    ArraySeq.create(arg8, arg9, arg10, arg11,
                                   arg12));
case 8:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8,
                    ArraySeq.create(arg9, arg10, arg11,
                                   arg12));
case 9:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9,
                    ArraySeq.create(arg10, arg11, arg12));
case 10:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10,
                    ArraySeq.create(arg11, arg12));
case 11:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11,
                    ArraySeq.create(arg12));
case 12:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, null);
default:
    return throwArity(12);
}

}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13)
throws Exception{
    switch(getRequiredArity())

```

```
{  
case 0:  
    return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,  
                                    arg5, arg6, arg7, arg8, arg9, arg10,  
                                    arg11, arg12, arg13));  
case 1:  
    return doInvoke(arg1,  
                  ArraySeq.create(arg2, arg3, arg4, arg5,  
                                 arg6, arg7, arg8, arg9, arg10, arg11,  
                                 arg12, arg13));  
case 2:  
    return doInvoke(arg1, arg2,  
                  ArraySeq.create(arg3, arg4, arg5, arg6,  
                                 arg7, arg8, arg9, arg10, arg11, arg12,  
                                 arg13));  
case 3:  
    return doInvoke(arg1, arg2, arg3,  
                  ArraySeq.create(arg4, arg5, arg6, arg7,  
                                 arg8, arg9, arg10, arg11, arg12,  
                                 arg13));  
case 4:  
    return doInvoke(arg1, arg2, arg3, arg4,  
                  ArraySeq.create(arg5, arg6, arg7, arg8,  
                                 arg9, arg10, arg11, arg12, arg13));  
case 5:  
    return doInvoke(arg1, arg2, arg3, arg4, arg5,  
                  ArraySeq.create(arg6, arg7, arg8, arg9,  
                                 arg10, arg11, arg12, arg13));  
case 6:  
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,  
                  ArraySeq.create(arg7, arg8, arg9, arg10,  
                                 arg11, arg12, arg13));  
case 7:  
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,  
                  ArraySeq.create(arg8, arg9, arg10, arg11,  
                                 arg12, arg13));  
case 8:  
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,  
                  arg8,  
                  ArraySeq.create(arg9, arg10, arg11,  
                                 arg12, arg13));  
case 9:  
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,  
                  arg8, arg9,  
                  ArraySeq.create(arg10, arg11, arg12,  
                                 arg13));  
case 10:  
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,  
                  arg8, arg9, arg10,  
                  ArraySeq.create(arg11, arg12, arg13));
```



```

        case 5:
            return doInvoke(arg1, arg2, arg3, arg4, arg5,
                            ArraySeq.create(arg6, arg7, arg8, arg9,
                                           arg10, arg11, arg12, arg13, arg14));
        case 6:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                            ArraySeq.create(arg7, arg8, arg9, arg10,
                                           arg11, arg12, arg13, arg14));
        case 7:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            ArraySeq.create(arg8, arg9, arg10, arg11,
                                           arg12, arg13, arg14));
        case 8:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8,
                            ArraySeq.create(arg9, arg10, arg11,
                                           arg12, arg13, arg14));
        case 9:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9,
                            ArraySeq.create(arg10, arg11, arg12,
                                           arg13, arg14));
        case 10:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10,
                            ArraySeq.create(arg11, arg12, arg13,
                                           arg14));
        case 11:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11,
                            ArraySeq.create(arg12, arg13, arg14));
        case 12:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11, arg12,
                            ArraySeq.create(arg13, arg14));
        case 13:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11, arg12, arg13,
                            ArraySeq.create(arg14));
        case 14:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11, arg12, arg13,
                            arg14, null);
        default:
            return throwArity(14);
    }

}

public Object invoke(Object arg1, Object arg2, Object arg3,

```

```

        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12,
        Object arg13, Object arg14, Object arg15)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                arg5, arg6, arg7, arg8, arg9, arg10,
                arg11, arg12, arg13, arg14, arg15));
        case 1:
            return doInvoke(arg1,
                ArraySeq.create(arg2, arg3, arg4, arg5,
                arg6, arg7, arg8, arg9, arg10, arg11,
                arg12, arg13, arg14, arg15));
        case 2:
            return doInvoke(arg1, arg2,
                ArraySeq.create(arg3, arg4, arg5, arg6,
                arg7, arg8, arg9, arg10, arg11, arg12,
                arg13, arg14, arg15));
        case 3:
            return doInvoke(arg1, arg2, arg3,
                ArraySeq.create(arg4, arg5, arg6, arg7,
                arg8, arg9, arg10, arg11, arg12,
                arg13, arg14, arg15));
        case 4:
            return doInvoke(arg1, arg2, arg3, arg4,
                ArraySeq.create(arg5, arg6, arg7, arg8,
                arg9, arg10, arg11, arg12, arg13,
                arg14, arg15));
        case 5:
            return doInvoke(arg1, arg2, arg3, arg4, arg5,
                ArraySeq.create(arg6, arg7, arg8, arg9,
                arg10, arg11, arg12, arg13, arg14,
                arg15));
        case 6:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                ArraySeq.create(arg7, arg8, arg9, arg10,
                arg11, arg12, arg13, arg14, arg15));
        case 7:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                ArraySeq.create(arg8, arg9, arg10, arg11,
                arg12, arg13, arg14, arg15));
        case 8:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                arg8,
                ArraySeq.create(arg9, arg10, arg11,
                arg12, arg13, arg14, arg15));
        case 9:
    }
}
```

```
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9,
                        ArraySeq.create(arg10, arg11, arg12,
                                      arg13, arg14, arg15));
    case 10:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10,
                        ArraySeq.create(arg11, arg12, arg13,
                                      arg14, arg15));
    case 11:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11,
                        ArraySeq.create(arg12, arg13, arg14,
                                      arg15));
    case 12:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12,
                        ArraySeq.create(arg13, arg14, arg15));
    case 13:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        ArraySeq.create(arg14, arg15));
    case 14:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14,
                        ArraySeq.create(arg15));
    case 15:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, null);
    default:
        return throwArity(15);
    }
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                            arg5, arg6, arg7, arg8, arg9, arg10,
                                            arg11, arg12, arg13, arg14, arg15,
```



```
case 11:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11,
                    ArraySeq.create(arg12, arg13, arg14,
                                    arg15, arg16));
case 12:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12,
                    ArraySeq.create(arg13, arg14, arg15,
                                    arg16));
case 13:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    ArraySeq.create(arg14, arg15, arg16));
case 14:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    arg14,
                    ArraySeq.create(arg15, arg16));
case 15:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    arg14, arg15,
                    ArraySeq.create(arg16));
case 16:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    arg14, arg15, arg16, null);
default:
    return throwArity(16);
}

}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                            arg5, arg6, arg7, arg8, arg9, arg10,
                                            arg11, arg12, arg13, arg14, arg15,
                                            arg16, arg17));
        case 1:
            return doInvoke(arg1,
```



```

        case 11:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11,
                            ArraySeq.create(arg12, arg13, arg14,
                                            arg15, arg16, arg17));
        case 12:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11, arg12,
                            ArraySeq.create(arg13, arg14, arg15,
                                            arg16, arg17));
        case 13:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11, arg12, arg13,
                            ArraySeq.create(arg14, arg15, arg16,
                                            arg17));
        case 14:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11, arg12, arg13,
                            arg14,
                            ArraySeq.create(arg15, arg16, arg17));
        case 15:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11, arg12, arg13,
                            arg14, arg15,
                            ArraySeq.create(arg16, arg17));
        case 16:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11, arg12, arg13,
                            arg14, arg15, arg16,
                            ArraySeq.create(arg17));
        case 17:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                            arg8, arg9, arg10, arg11, arg12, arg13,
                            arg14, arg15, arg16, arg17, null);
        default:
            return throwArity(17);
    }
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:

```



```
        arg13, arg14, arg15, arg16, arg17,
        arg18));
case 10:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10,
                    ArraySeq.create(arg11, arg12, arg13,
                                   arg14, arg15, arg16, arg17, arg18));
case 11:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11,
                    ArraySeq.create(arg12, arg13, arg14,
                                   arg15, arg16, arg17, arg18));
case 12:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12,
                    ArraySeq.create(arg13, arg14, arg15,
                                   arg16, arg17, arg18));
case 13:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    ArraySeq.create(arg14, arg15, arg16,
                                   arg17, arg18));
case 14:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    arg14,
                    ArraySeq.create(arg15, arg16, arg17,
                                   arg18));
case 15:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    arg14, arg15,
                    ArraySeq.create(arg16, arg17, arg18));
case 16:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    arg14, arg15, arg16,
                    ArraySeq.create(arg17, arg18));
case 17:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    arg14, arg15, arg16, arg17,
                    ArraySeq.create(arg18));
case 18:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                    arg8, arg9, arg10, arg11, arg12, arg13,
                    arg14, arg15, arg16, arg17, arg18, null);
default:
    return throwArity(18);
}
```

```
}
```

```
public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19)
throws Exception{
    switch(getRequiredArity())
    {
        case 0:
            return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                             arg5, arg6, arg7, arg8, arg9, arg10,
                                             arg11, arg12, arg13, arg14, arg15,
                                             arg16, arg17, arg18, arg19));
        case 1:
            return doInvoke(arg1,
                           ArraySeq.create(arg2, arg3, arg4, arg5,
                                         arg6, arg7, arg8, arg9, arg10, arg11,
                                         arg12, arg13, arg14, arg15, arg16,
                                         arg17, arg18, arg19));
        case 2:
            return doInvoke(arg1, arg2,
                           ArraySeq.create(arg3, arg4, arg5, arg6,
                                         arg7, arg8, arg9, arg10, arg11, arg12,
                                         arg13, arg14, arg15, arg16, arg17,
                                         arg18, arg19));
        case 3:
            return doInvoke(arg1, arg2, arg3,
                           ArraySeq.create(arg4, arg5, arg6,
                                         arg7, arg8, arg9, arg10, arg11,
                                         arg12, arg13, arg14, arg15, arg16,
                                         arg17, arg18, arg19));
        case 4:
            return doInvoke(arg1, arg2, arg3, arg4,
                           ArraySeq.create(arg5, arg6, arg7, arg8,
                                         arg9, arg10, arg11, arg12, arg13,
                                         arg14, arg15, arg16, arg17, arg18,
                                         arg19));
        case 5:
            return doInvoke(arg1, arg2, arg3, arg4, arg5,
                           ArraySeq.create(arg6, arg7, arg8, arg9,
                                         arg10, arg11, arg12, arg13, arg14, arg15,
                                         arg16, arg17, arg18, arg19));
        case 6:
            return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                           ArraySeq.create(arg7, arg8, arg9, arg10,
```

```
        arg11, arg12, arg13, arg14, arg15,
        arg16, arg17, arg18, arg19));
case 7:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        ArraySeq.create(arg8, arg9, arg10, arg11,
            arg12, arg13, arg14, arg15, arg16, arg17,
            arg18, arg19));
case 8:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        arg8,
        ArraySeq.create(arg9, arg10, arg11,
            arg12, arg13, arg14, arg15, arg16,
            arg17, arg18, arg19));
case 9:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        arg8, arg9,
        ArraySeq.create(arg10, arg11, arg12,
            arg13, arg14, arg15, arg16, arg17,
            arg18, arg19));
case 10:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        arg8, arg9, arg10,
        ArraySeq.create(arg11, arg12, arg13,
            arg14, arg15, arg16, arg17, arg18,
            arg19));
case 11:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        arg8, arg9, arg10, arg11,
        ArraySeq.create(arg12, arg13, arg14,
            arg15, arg16, arg17, arg18, arg19));
case 12:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        arg8, arg9, arg10, arg11, arg12,
        ArraySeq.create(arg13, arg14, arg15,
            arg16, arg17, arg18, arg19));
case 13:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        arg8, arg9, arg10, arg11, arg12, arg13,
        ArraySeq.create(arg14, arg15, arg16,
            arg17, arg18, arg19));
case 14:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        arg8, arg9, arg10, arg11, arg12, arg13,
        arg14,
        ArraySeq.create(arg15, arg16, arg17,
            arg18, arg19));
case 15:
    return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
        arg8, arg9, arg10, arg11, arg12, arg13,
        arg14, arg15,
```

```

        ArraySeq.create(arg16, arg17, arg18,
                        arg19));
    case 16:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16,
                        ArraySeq.create(arg17, arg18, arg19));
    case 17:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16, arg17,
                        ArraySeq.create(arg18, arg19));
    case 18:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16, arg17, arg18,
                        ArraySeq.create(arg19));
    case 19:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16, arg17, arg18, arg19,
                        null);
    default:
        return throwArity(19);
    }
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20)
    throws Exception{
switch(getRequiredArity())
{
    case 0:
        return doInvoke(ArraySeq.create(arg1, arg2, arg3, arg4,
                                         arg5, arg6, arg7, arg8, arg9, arg10,
                                         arg11, arg12, arg13, arg14, arg15,
                                         arg16, arg17, arg18, arg19, arg20));
    case 1:
        return doInvoke(arg1,
                        ArraySeq.create(arg2, arg3, arg4, arg5,
                                       arg6, arg7, arg8, arg9, arg10, arg11,
                                       arg12, arg13, arg14, arg15, arg16,
                                       arg17, arg18, arg19, arg20));
    case 2:
}
}

```

```
        return doInvoke(arg1, arg2,
                        ArraySeq.create(arg3, arg4, arg5, arg6,
                                       arg7, arg8, arg9, arg10, arg11, arg12,
                                       arg13, arg14, arg15, arg16, arg17,
                                       arg18, arg19, arg20));
    case 3:
        return doInvoke(arg1, arg2, arg3,
                        ArraySeq.create(arg4, arg5, arg6,
                                       arg7, arg8, arg9, arg10, arg11, arg12,
                                       arg13, arg14, arg15, arg16, arg17,
                                       arg18, arg19, arg20));
    case 4:
        return doInvoke(arg1, arg2, arg3, arg4,
                        ArraySeq.create(arg5, arg6, arg7, arg8,
                                       arg9, arg10, arg11, arg12, arg13, arg14,
                                       arg15, arg16, arg17, arg18, arg19,
                                       arg20));
    case 5:
        return doInvoke(arg1, arg2, arg3, arg4, arg5,
                        ArraySeq.create(arg6, arg7, arg8, arg9,
                                       arg10, arg11, arg12, arg13, arg14,
                                       arg15, arg16, arg17, arg18, arg19,
                                       arg20));
    case 6:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6,
                        ArraySeq.create(arg7, arg8, arg9, arg10,
                                       arg11, arg12, arg13, arg14, arg15,
                                       arg16, arg17, arg18, arg19, arg20));
    case 7:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        ArraySeq.create(arg8, arg9, arg10, arg11,
                                       arg12, arg13, arg14, arg15, arg16, arg17,
                                       arg18, arg19, arg20));
    case 8:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8,
                        ArraySeq.create(arg9, arg10, arg11, arg12,
                                       arg13, arg14, arg15, arg16, arg17, arg18,
                                       arg19, arg20));
    case 9:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9,
                        ArraySeq.create(arg10, arg11, arg12,
                                       arg13, arg14, arg15, arg16, arg17,
                                       arg18, arg19, arg20));
    case 10:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10,
                        ArraySeq.create(arg11, arg12, arg13,
                                       arg14, arg15, arg16, arg17, arg18,
```



```
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16, arg17, arg18, arg19,
                        arg20, null);
    default:
        return throwArity(20);
}
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20, Object... args)
throws Exception{
switch(getRequiredArity())
{
case 0:
    return doInvoke(
        ontoArrayPrepend(args, arg1, arg2, arg3,
                        arg4, arg5, arg6, arg7, arg8, arg9,
                        arg10, arg11, arg12, arg13, arg14,
                        arg15, arg16, arg17, arg18, arg19, arg20));
case 1:
    return doInvoke(arg1,
        ontoArrayPrepend(args, arg2, arg3, arg4,
                        arg5, arg6, arg7, arg8, arg9, arg10, arg11,
                        arg12, arg13, arg14, arg15, arg16, arg17,
                        arg18, arg19, arg20));
case 2:
    return doInvoke(arg1, arg2,
        ontoArrayPrepend(args, arg3, arg4, arg5,
                        arg6, arg7, arg8, arg9, arg10, arg11,
                        arg12, arg13, arg14, arg15, arg16,
                        arg17, arg18, arg19, arg20));
case 3:
    return doInvoke(arg1, arg2, arg3,
        ontoArrayPrepend(args, arg4, arg5, arg6,
                        arg7, arg8, arg9, arg10, arg11, arg12,
                        arg13, arg14, arg15, arg16, arg17,
                        arg18, arg19, arg20));
case 4:
    return doInvoke(arg1, arg2, arg3, arg4,
        ontoArrayPrepend(args, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16, arg17, arg18,
                        arg19, arg20));
```



```

        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14,
                        ontoArrayPrepend(args, arg15, arg16, arg17,
                                         arg18, arg19, arg20));
    case 15:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15,
                        ontoArrayPrepend(args, arg16, arg17, arg18,
                                         arg19, arg20));
    case 16:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16,
                        ontoArrayPrepend(args, arg17, arg18, arg19,
                                         arg20));
    case 17:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16, arg17,
                        ontoArrayPrepend(args, arg18, arg19,
                                         arg20));
    case 18:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16, arg17, arg18,
                        ontoArrayPrepend(args, arg19, arg20));
    case 19:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16, arg17, arg18, arg19,
                        ontoArrayPrepend(args, arg20));
    case 20:
        return doInvoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7,
                        arg8, arg9, arg10, arg11, arg12, arg13,
                        arg14, arg15, arg16, arg17, arg18, arg19,
                        arg20, ArraySeq.create(args));
    default:
        return throwArity(21);
}
}

protected static ISeq ontoArrayPrepend(Object[] array,
                                      Object... args){
    ISeq ret = ArraySeq.create(array);
    for(int i = args.length - 1; i >= 0; --i)
        ret = RT.cons(args[i], ret);
}

```

```

        return ret;
    }

protected static ISeq findKey(Object key, ISeq args){
    while(args != null)
    {
        if(key == args.first())
            return args.next();
        args = RT.next(args);
        args = RT.next(args);
    }
    return null;
}

}

```

—————

9.94 Reversible.java

— Reversible.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jan 5, 2008 */

package clojure.lang;

public interface Reversible{
ISeq rseq() throws Exception;
}

```

—————

9.95 RT.java

— RT.java —

```

/*
\getchunk{Clojure Copyright}
*/

```

```
/* rich Mar 25, 2006 4:28:27 PM */

package clojure.lang;

import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.Callable;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.io.*;
import java.lang.reflect.Array;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.security.AccessController;
import java.security.PrivilegedAction;
import java.net.URL;
import java.net.JarURLConnection;
import java.nio.charset.Charset;

public class RT{

    static final public Boolean T =
        Boolean.TRUE;//Keyword.intern(Symbol.intern(null, "t"));
    static final public Boolean F =
        Boolean.FALSE;//Keyword.intern(Symbol.intern(null, "t"));
    static final public String LOADER_SUFFIX = "__init__";

    //simple-symbol->class
    final static IPersistentMap DEFAULT_IMPORTS = map(
        // Symbol.intern("RT"), "clojure.lang.RT",
        // Symbol.intern("Num"), "clojure.lang.Num",
        // Symbol.intern("Symbol"), "clojure.lang.Symbol",
        // Symbol.intern("Keyword"), "clojure.lang.Keyword",
        // Symbol.intern("Var"), "clojure.lang.Var",
        // Symbol.intern("Ref"), "clojure.lang.Ref",
        // Symbol.intern("IFn"), "clojure.lang.IFn",
        // Symbol.intern("IObj"), "clojure.lang.IObj",
        // Symbol.intern("ISeq"), "clojure.lang.ISeq",
        // Symbol.intern("IPersistentCollection"),
        //         "clojure.lang.IIPersistentCollection",
        // Symbol.intern("IPersistentMap"), "clojure.lang.IIPersistentMap",
        // Symbol.intern("IPersistentList"), "clojure.lang.IIPersistentList",
        // Symbol.intern("IPersistentVector"), "clojure.lang.IIPersistentVector",
        Symbol.intern("Boolean"), Boolean.class,
        Symbol.intern("Byte"), Byte.class,
        Symbol.intern("Character"), Character.class,
        Symbol.intern("Class"), Class.class,
        Symbol.intern("ClassLoader"), ClassLoader.class,
        Symbol.intern("Compiler"), Compiler.class,
        Symbol.intern("Double"), Double.class,
```

```
Symbol.intern("Enum"), Enum.class,
Symbol.intern("Float"), Float.class,
Symbol.intern("InheritableThreadLocal"), InheritableThreadLocal.class,
Symbol.intern("Integer"), Integer.class,
Symbol.intern("Long"), Long.class,
Symbol.intern("Math"), Math.class,
Symbol.intern("Number"), Number.class,
Symbol.intern("Object"), Object.class,
Symbol.intern("Package"), Package.class,
Symbol.intern("Process"), Process.class,
Symbol.intern("ProcessBuilder"), ProcessBuilder.class,
Symbol.intern("Runtime"), Runtime.class,
Symbol.intern("RuntimePermission"), RuntimePermission.class,
Symbol.intern("SecurityManager"), SecurityManager.class,
Symbol.intern("Short"), Short.class,
Symbol.intern("StackTraceElement"), StackTraceElement.class,
Symbol.intern("StrictMath"), StrictMath.class,
Symbol.intern("String"), String.class,
Symbol.intern("StringBuffer"), StringBuffer.class,
Symbol.intern("StringBuilder"), StringBuilder.class,
Symbol.intern("System"), System.class,
Symbol.intern("Thread"), Thread.class,
Symbol.intern("ThreadGroup"), ThreadGroup.class,
Symbol.intern("ThreadLocal"), ThreadLocal.class,
Symbol.intern("Throwable"), Throwable.class,
Symbol.intern("Void"), Void.class,
Symbol.intern("Appendable"), Appendable.class,
Symbol.intern("CharSequence"), CharSequence.class,
Symbol.intern("Cloneable"), Cloneable.class,
Symbol.intern("Comparable"), Comparable.class,
Symbol.intern("Iterable"), Iterable.class,
Symbol.intern("Readable"), Readable.class,
Symbol.intern("Runnable"), Runnable.class,
Symbol.intern("Callable"), Callable.class,
Symbol.intern("BigInteger"), BigInteger.class,
Symbol.intern("BigDecimal"), BigDecimal.class,
Symbol.intern("ArithmaticException"), ArithmaticException.class,
Symbol.intern("ArrayIndexOutOfBoundsException"),
    ArrayIndexOutOfBoundsException.class,
Symbol.intern("ArrayStoreException"), ArrayStoreException.class,
Symbol.intern("ClassCastException"), ClassCastException.class,
Symbol.intern("ClassNotFoundException"), ClassNotFoundException.class,
Symbol.intern("CloneNotSupportedException"),
    CloneNotSupportedException.class,
Symbol.intern("EnumConstantNotPresentException"),
    EnumConstantNotPresentException.class,
Symbol.intern("Exception"), Exception.class,
Symbol.intern("IllegalAccessException"), IllegalAccessException.class,
Symbol.intern("IllegalArgumentException"),
    IllegalArgumentException.class,
```

```
Symbol.intern("IllegalMonitorStateException"),
           IllegalMonitorStateException.class,
Symbol.intern("IllegalStateException"), IllegalStateException.class,
Symbol.intern("IllegalThreadStateException"),
           IllegalThreadStateException.class,
Symbol.intern("IndexOutOfBoundsException"),
           IndexOutOfBoundsException.class,
Symbol.intern("InstantiationException"), InstantiationException.class,
Symbol.intern("InterruptedException"), InterruptedException.class,
Symbol.intern("NegativeArraySizeException"),
           NegativeArraySizeException.class,
Symbol.intern("NoSuchFieldException"), NoSuchFieldException.class,
Symbol.intern("NoSuchMethodException"), NoSuchMethodException.class,
Symbol.intern("NullPointerException"), NullPointerException.class,
Symbol.intern("NumberFormatException"), NumberFormatException.class,
Symbol.intern("RuntimeException"), RuntimeException.class,
Symbol.intern("SecurityException"), SecurityException.class,
Symbol.intern("StringIndexOutOfBoundsException"),
           StringIndexOutOfBoundsException.class,
Symbol.intern("TypeNotPresentException"),
           TypeNotPresentException.class,
Symbol.intern("UnsupportedOperationException"),
           UnsupportedOperationException.class,
Symbol.intern("AbstractMethodError"), AbstractMethodError.class,
Symbol.intern("AssertionError"), AssertionError.class,
Symbol.intern("ClassCircularityError"), ClassCircularityError.class,
Symbol.intern("ClassFormatError"), ClassFormatError.class,
Symbol.intern("Error"), Error.class,
Symbol.intern("ExceptionInInitializerError"),
           ExceptionInInitializerError.class,
Symbol.intern("IllegalAccessException"), IllegalAccessException.class,
Symbol.intern("IncompatibleClassChangeError"),
           IncompatibleClassChangeError.class,
Symbol.intern("InstantiationException"), InstantiationException.class,
Symbol.intern("InternalError"), InternalError.class,
Symbol.intern("LinkageError"), LinkageError.class,
Symbol.intern("NoClassDefFoundError"), NoClassDefFoundError.class,
Symbol.intern("NoSuchFieldError"), NoSuchFieldError.class,
Symbol.intern("NoSuchMethodError"), NoSuchMethodError.class,
Symbol.intern("OutOfMemoryError"), OutOfMemoryError.class,
Symbol.intern("StackOverflowError"), StackOverflowError.class,
Symbol.intern("ThreadDeath"), ThreadDeath.class,
Symbol.intern("UnknownError"), UnknownError.class,
Symbol.intern("UnsatisfiedLinkError"), UnsatisfiedLinkError.class,
Symbol.intern("UnsupportedClassVersionError"),
           UnsupportedClassVersionError.class,
Symbol.intern("VerifyError"), VerifyError.class,
Symbol.intern("VirtualMachineError"), VirtualMachineError.class,
Symbol.intern("Thread$UncaughtExceptionHandler"),
           Thread.UncaughtExceptionHandler.class,
```

```

Symbol.intern("Thread$State"), Thread.State.class,
Symbol.intern("Deprecated"), Deprecated.class,
Symbol.intern("Override"), Override.class,
Symbol.intern("SuppressWarnings"), SuppressWarnings.class

// Symbol.intern("Collection"), "java.util.Collection",
// Symbol.intern("Comparator"), "java.util.Comparator",
// Symbol.intern("Enumeration"), "java.util.Enumeration",
// Symbol.intern("EventListener"), "java.util.EventListener",
// Symbol.intern("Formattable"), "java.util.Formattable",
// Symbol.intern("Iterator"), "java.util.Iterator",
// Symbol.intern("List"), "java.util.List",
// Symbol.intern("ListIterator"), "java.util.ListIterator",
// Symbol.intern("Map"), "java.util.Map",
// Symbol.intern("Map$Entry"), "java.util.Map$Entry",
// Symbol.intern("Observer"), "java.util.Observer",
// Symbol.intern("Queue"), "java.util.Queue",
// Symbol.intern("RandomAccess"), "java.util.RandomAccess",
// Symbol.intern("Set"), "java.util.Set",
// Symbol.intern("SortedMap"), "java.util.SortedMap",
// Symbol.intern("SortedSet"), "java.util.SortedSet"
);

// single instance of UTF-8 Charset, so as to avoid
// catching UnsupportedCharsetExceptions everywhere
static public Charset UTF8 = Charset.forName("UTF-8");

static public final Namespace CLOJURE_NS =
    Namespace.findOrCreate(Symbol.intern("clojure.core"));
//static final Namespace USER_NS =
//    Namespace.findOrCreate(Symbol.intern("user"));
final static public Var OUT =
    Var.intern(CLOJURE_NS, Symbol.intern("*out*"),
               new OutputStreamWriter(System.out))
        .setDynamic();
final static public Var IN =
    Var.intern(CLOJURE_NS, Symbol.intern("*in*"),
               new LineNumberingPushbackReader(
                   new InputStreamReader(System.in)))
        .setDynamic();
final static public Var ERR =
    Var.intern(CLOJURE_NS, Symbol.intern("*err*"),
               new PrintWriter(
                   new OutputStreamWriter(System.err), true))
        .setDynamic();
final static Keyword TAG_KEY = Keyword.intern(null, "tag");
final static public Var AGENT =
    Var.intern(CLOJURE_NS, Symbol.intern("*agent*"), null)
        .setDynamic();
final static public Var READEVAL =

```

```
Var.intern(CLOJURE_NS, Symbol.intern("*read-eval*"), T)
.setDynamic();
final static public Var ASSERT =
Var.intern(CLOJURE_NS, Symbol.intern("*assert*"), T)
.setDynamic();
final static public Var MATH_CONTEXT =
Var.intern(CLOJURE_NS, Symbol.intern("*math-context*"), null)
.setDynamic();
static Keyword LINE_KEY = Keyword.intern(null, "line");
static Keyword FILE_KEY = Keyword.intern(null, "file");
static Keyword DECLARED_KEY = Keyword.intern(null, "declared");
static Keyword DOC_KEY = Keyword.intern(null, "doc");
final static public Var USE_CONTEXT_CLASSLOADER =
Var.intern(CLOJURE_NS,
Symbol.intern("*use-context-classloader*"), T)
.setDynamic();
//final static public Var CURRENT_MODULE =
//  Var.intern(Symbol.intern("clojure.core", "current-module"),
//  Module.findOrCreateModule("clojure/user"));

final static Symbol LOAD_FILE = Symbol.intern("load-file");
final static Symbol IN_NAMESPACE = Symbol.intern("in-ns");
final static Symbol NAMESPACE = Symbol.intern("ns");
static final Symbol IDENTICAL = Symbol.intern("identical?");
final static Var CMD_LINE_ARGS =
Var.intern(CLOJURE_NS,
Symbol.intern("*command-line-args*"), null)
.setDynamic();
//symbol
final public static Var CURRENT_NS =
Var.intern(CLOJURE_NS, Symbol.intern("*ns*"), CLOJURE_NS)
.setDynamic();

final static Var FLUSH_ON_NEWLINE =
Var.intern(CLOJURE_NS, Symbol.intern("*flush-on-newline*"), T)
.setDynamic();
final static Var PRINT_META =
Var.intern(CLOJURE_NS, Symbol.intern("*print-meta*"), F)
.setDynamic();
final static Var PRINT_READABLY =
Var.intern(CLOJURE_NS, Symbol.intern("*print-readably*"), T)
.setDynamic();
final static Var PRINT_DUP =
Var.intern(CLOJURE_NS, Symbol.intern("*print-dup*"), F)
.setDynamic();
final static Var WARN_ON_REFLECTION =
Var.intern(CLOJURE_NS, Symbol.intern("*warn-on-reflection*"), F)
.setDynamic();
final static Var ALLOW_UNRESOLVED_VARS =
Var.intern(CLOJURE_NS,
```

```

        Symbol.intern("*allow-unresolved-vars*"), F)
.setDynamic();

final static Var IN_NS_VAR =
Var.intern(CLOJURE_NS, Symbol.intern("in-ns"), F);
final static Var NS_VAR =
Var.intern(CLOJURE_NS, Symbol.intern("ns"), F);
static final Var PRINT_INITIALIZED =
Var.intern(CLOJURE_NS, Symbol.intern("print-initialized"));
static final Var PR_ON =
Var.intern(CLOJURE_NS, Symbol.intern("pr-on"));
//final static Var IMPORTS =
//    Var.intern(CLOJURE_NS, Symbol.intern("*imports*"),
//    DEFAULT_IMPORTS);
final static IFn inNamespace = new AFn(){
    public Object invoke(Object arg1) throws Exception{
        Symbol nsname = (Symbol) arg1;
        Namespace ns = Namespace.findOrCreate(nsname);
        CURRENT_NS.set(ns);
        return ns;
    }
};

final static IFn bootNamespace = new AFn(){
    public Object invoke(Object __form,
Object __env, Object arg1)
throws Exception{
    Symbol nsname = (Symbol) arg1;
    Namespace ns = Namespace.findOrCreate(nsname);
    CURRENT_NS.set(ns);
    return ns;
}
};

public static List<String> processCommandLine(String[] args){
List<String> arglist = Arrays.asList(args);
int split = arglist.indexOf("--");
if(split >= 0) {
    CMD_LINE_ARGS.bindRoot(
        RT.seq(arglist.subList(split + 1, args.length)));
    return arglist.subList(0, split);
}
return arglist;
}

// duck typing stderr plays nice with e.g. swank
public static PrintWriter errPrintWriter(){
    Writer w = (Writer) ERR.deref();
    if (w instanceof PrintWriter) {
        return (PrintWriter) w;
    }
}

```

```

        } else {
            return new PrintWriter(w);
        }
    }

static public final Object[] EMPTY_ARRAY = new Object[] {};
static public final Comparator DEFAULT_COMPARATOR =
new DefaultComparator();

private static final class DefaultComparator
implements Comparator, Serializable {
    public int compare(Object o1, Object o2){
        return Util.compare(o1, o2);
    }

    private Object readResolve() throws ObjectStreamException {
        // ensures that we aren't hanging onto a new default
        // comparator for every sorted set, etc., we deserialize
        return DEFAULT_COMPARATOR;
    }
}

static AtomicInteger id = new AtomicInteger(1);

static public void addURL(Object url) throws Exception{
    URL u =
        (url instanceof String) ? (new URL((String) url)) : (URL) url;
    ClassLoader ccl = Thread.currentThread().getContextClassLoader();
    if(ccl instanceof DynamicClassLoader)
        ((DynamicClassLoader)ccl).addURL(u);
    else
        throw new IllegalAccessException(
            "Context classloader is not a DynamicClassLoader");
}

static{
    Keyword arglists_kw = Keyword.intern(null, "arglists");
    Symbol namesym = Symbol.intern("name");
    OUT.setTag(Symbol.intern("java.io.Writer"));
    CURRENT_NS.setTag(Symbol.intern("clojure.lang.Namespace"));
    AGENT.setMeta(map(DOC_KEY,
        "The agent currently running an action on "+
        "this thread, else nil"));
    AGENT.setTag(Symbol.intern("clojure.lang.Agent"));
    MATH_CONTEXT.setTag(Symbol.intern("java.math.MathContext"));
    Var nv = Var.intern(CLOJURE_NS, NAMESPACE, bootNamespace);
    nv.setMacro();
    Var v;
    v = Var.intern(CLOJURE_NS, IN_NAMESPACE, inNamespace);
    v.setMeta(map(DOC_KEY,

```

```

    "Sets *ns* to the namespace named by the symbol, "+
    "creating it if needed.",
    arglists_kw, list(vector(namesym))));

v = Var.intern(CLOJURE_NS, LOAD_FILE,
    new AFn(){
        public Object invoke(Object arg1)
        throws Exception{
            return Compiler.loadFile((String) arg1);
        }
    });
v.setMeta(map(DOC_KEY,
    "Sequentially read and evaluate the set of "+
    "forms contained in the file.",
    arglists_kw, list(vector(namesym))));

try {
    doInit();
}
catch(Exception e) {
    throw new RuntimeException(e);
}
}

static public Var var(String ns, String name){
    return
        Var.intern	Namespace.findOrCreate(Symbol.intern(null, ns)),
        Symbol.intern(null, name));
}

static public Var var(String ns, String name, Object init){
    return
        Var.intern(
            Namespace.findOrCreate(Symbol.intern(null, ns)),
            Symbol.intern(null, name), init);
}

public static void loadResourceScript(String name)
throws Exception{
    loadResourceScript(name, true);
}

public static void maybeLoadResourceScript(String name)
throws Exception{
    loadResourceScript(name, false);
}

public static void loadResourceScript(String name,
                                      boolean failIfNotFound)
throws Exception{
    loadResourceScript(RT.class, name, failIfNotFound);
}

```

```
}

public static void loadResourceScript(Class c, String name)
throws Exception{
    loadResourceScript(c, name, true);
}

public static void loadResourceScript(Class c,
                                      String name,
                                      boolean failIfNotFound)
throws Exception{
    int slash = name.lastIndexOf('/');
    String file = slash >= 0 ? name.substring(slash + 1) : name;
    InputStream ins = baseLoader().getResourceAsStream(name);
    if(ins != null) {
        try {
            Compiler.load(new InputStreamReader(ins, UTF8), name, file);
        }
        finally {
            ins.close();
        }
    }
    else if(failIfNotFound) {
        throw new FileNotFoundException(
            "Could not locate Clojure resource on classpath: " + name);
    }
}

static public void init() throws Exception{
    RT.errPrintWriter().println(
        "No need to call RT.init() anymore");
}

static public long lastModified(URL url, String libfile)
throws Exception{
    if(url.getProtocol().equals("jar")) {
        return
            ((JarURLConnection) url.openConnection())
                .getJarFile()
                .getEntry(libfile)
                .getTime();
    }
    else {
        return url.openConnection().getLastModified();
    }
}

static void compile(String cljfile) throws Exception{
    InputStream ins = baseLoader().getResourceAsStream(cljfile);
    if(ins != null) {
```

```

        try {
            Compiler.compile(
                new InputStreamReader(ins, UTF8), cljfile,
                cljfile.substring(1 + cljfile.lastIndexOf("/")));
        }
        finally {
            ins.close();
        }

    }
    else
        throw new FileNotFoundException(
            "Could not locate Clojure resource on classpath: " + cljfile);
}

static public void load(String scriptbase) throws Exception{
    load(scriptbase, true);
}

static public void load(String scriptbase, boolean failIfNotFound)
throws Exception{
    String classfile = scriptbase + LOADER_SUFFIX + ".class";
    String cljfile = scriptbase + ".cljs";
    URL classURL = baseLoader().getResource(classfile);
    URL cljURL = baseLoader().getResource(cljfile);
    boolean loaded = false;

    if((classURL != null &&
        (cljURL == null
         || lastModified(classURL, classfile) >
            lastModified(cljURL, cljfile)))
       || classURL == null) {
        try {
            Var.pushThreadBindings(
                RT.map(CURRENT_NS, CURRENT_NS.deref(),
                    WARN_ON_REFLECTION, WARN_ON_REFLECTION.deref()));
            loaded =
                (loadClassForName(scriptbase.replace('/', '.') +
                    LOADER_SUFFIX) != null);
        }
        finally {
            Var.popThreadBindings();
        }
    }
    if(!loaded && cljURL != null) {
        if(booleanCast(Compiler.COMPILE_FILES.deref()))
            compile(cljfile);
        else
            loadResourceScript(RT.class, cljfile);
    }
}

```

```

        else if(!loaded && failIfNotFound)
            throw new FileNotFoundException(String.format(
                "Could not locate %s or %s on classpath: ",
                classfile, cljfile));
    }

static void doInit() throws Exception{
    load("clojure/core");

    Var.pushThreadBindings(
        RT.map(CURRENT_NS, CURRENT_NS.deref(),
            WARN_ON_REFLECTION, WARN_ON_REFLECTION.deref()));
    try {
        Symbol USER = Symbol.intern("user");
        Symbol CLOJURE = Symbol.intern("clojure.core");

        Var.in_ns = var("clojure.core", "in-ns");
        Var refer = var("clojure.core", "refer");
        in_ns.invoke(USER);
        refer.invoke(CLOJURE);
        maybeLoadResourceScript("user.clj");
    }
    finally {
        Var.popThreadBindings();
    }
}

static public int nextID(){
    return id.getAndIncrement();
}

//////////////// Collections support //////////////////////////////

static public ISeq seq(Object coll){
    if(coll instanceof ASeq)
        return (ASeq) coll;
    else if(coll instanceof LazySeq)
        return ((LazySeq) coll).seq();
    else
        return seqFrom(coll);
}

static ISeq seqFrom(Object coll){
    if(coll instanceof Seqable)
        return ((Seqable) coll).seq();
    else if(coll == null)
        return null;
    else if(coll instanceof Iterable)
        return IteratorSeq.create(((Iterable) coll).iterator());
}

```

```

        else if(coll.getClass().isArray())
            return ArraySeq.createFromObject(coll);
        else if(coll instanceof CharSequence)
            return StringSeq.create((CharSequence) coll);
        else if(coll instanceof Map)
            return seq(((Map) coll).entrySet());
        else {
            Class c = coll.getClass();
            Class sc = c.getSuperclass();
            throw new IllegalArgumentException(
                "Don't know how to create ISeq from: " + c.getName());
        }
    }

    static public ISeq keys(Object coll){
        return APersistentMap.KeySeq.create(seq(coll));
    }

    static public ISeq vals(Object coll){
        return APersistentMap.ValSeq.create(seq(coll));
    }

    static public IPersistentMap meta(Object x){
        if(x instanceof IMeta)
            return ((IMeta) x).meta();
        return null;
    }

    public static int count(Object o){
        if(o instanceof Counted)
            return ((Counted) o).count();
        return countFrom(Util.ret1(o, o == null));
    }

    static int countFrom(Object o){
        if(o == null)
            return 0;
        else if(o instanceof IPersistentCollection) {
            ISeq s = seq(o);
            o = null;
            int i = 0;
            for(; s != null; s = s.next()) {
                if(s instanceof Counted)
                    return i + s.count();
                i++;
            }
            return i;
        }
        else if(o instanceof CharSequence)
            return ((CharSequence) o).length();
    }
}

```

```
        else if(o instanceof Collection)
            return ((Collection) o).size();
        else if(o instanceof Map)
            return ((Map) o).size();
        else if(o.getClass().isArray())
            return Array.getLength(o);

        throw new UnsupportedOperationException(
            "count not supported on this type: " +
            o.getClass().getSimpleName());
    }

    static public IPersistentCollection conj(IPersistentCollection coll,
                                              Object x){
        if(coll == null)
            return new PersistentList(x);
        return coll.cons(x);
    }

    static public ISeq cons(Object x, Object coll){
        //ISeq y = seq(coll);
        if(coll == null)
            return new PersistentList(x);
        else if(coll instanceof ISeq)
            return new Cons(x, (ISeq) coll);
        else
            return new Cons(x, seq(coll));
    }

    static public Object first(Object x){
        if(x instanceof ISeq)
            return ((ISeq) x).first();
        ISeq seq = seq(x);
        if(seq == null)
            return null;
        return seq.first();
    }

    static public Object second(Object x){
        return first(next(x));
    }

    static public Object third(Object x){
        return first(next(next(x)));
    }

    static public Object fourth(Object x){
        return first(next(next(next(x))));
    }
```

```

static public ISeq next(Object x){
    if(x instanceof ISeq)
        return ((ISeq) x).next();
    ISeq seq = seq(x);
    if(seq == null)
        return null;
    return seq.next();
}

static public ISeq more(Object x){
    if(x instanceof ISeq)
        return ((ISeq) x).more();
    ISeq seq = seq(x);
    if(seq == null)
        return PersistentList.EMPTY;
    return seq.more();
}

//static public Seqable more(Object x){
//    Seqable ret = null;
//    if(x instanceof ISeq)
//        ret = ((ISeq) x).more();
//    else
//        {
//            ISeq seq = seq(x);
//            if(seq == null)
//                ret = PersistentList.EMPTY;
//            else
//                ret = seq.more();
//        }
//    if(ret == null)
//        ret = PersistentList.EMPTY;
//    return ret;
//}

static public Object peek(Object x){
    if(x == null)
        return null;
    return ((IPersistentStack) x).peek();
}

static public Object pop(Object x){
    if(x == null)
        return null;
    return ((IPersistentStack) x).pop();
}

static public Object get(Object coll, Object key){
    if(coll instanceof ILookup)
        return ((ILookup) coll).valAt(key);
}

```

```
        return getFrom(coll, key);
    }

static Object getFrom(Object coll, Object key){
    if(coll == null)
        return null;
    else if(coll instanceof Map) {
        Map m = (Map) coll;
        return m.get(key);
    }
    else if(coll instanceof IPersistentSet) {
        IPersistentSet set = (IPersistentSet) coll;
        return set.get(key);
    }
    else if(key instanceof Number &&
            (coll instanceof String ||
             coll.getClass().isArray())) {
        int n = ((Number) key).intValue();
        if(n >= 0 && n < count(coll))
            return nth(coll, n);
        return null;
    }
    return null;
}

static public Object get(Object coll, Object key, Object notFound){
    if(coll instanceof ILookup)
        return ((ILookup) coll).valAt(key, notFound);
    return getFrom(coll, key, notFound);
}

static Object getFrom(Object coll, Object key, Object notFound){
    if(coll == null)
        return notFound;
    else if(coll instanceof Map) {
        Map m = (Map) coll;
        if(m.containsKey(key))
            return m.get(key);
        return notFound;
    }
    else if(coll instanceof IPersistentSet) {
        IPersistentSet set = (IPersistentSet) coll;
        if(set.contains(key))
            return set.get(key);
        return notFound;
    }
    else if(key instanceof Number &&
            (coll instanceof String ||
             coll.getClass().isArray())) {
```

```

        int n = ((Number) key).intValue();
        return n >= 0 && n < count(coll) ? nth(coll, n) : notFound;
    }
    return notFound;
}

static public Associative assoc(Object coll, Object key, Object val){
    if(coll == null)
        return new PersistentArrayMap(new Object[]{key, val});
    return ((Associative) coll).assoc(key, val);
}

static public Object contains(Object coll, Object key){
    if(coll == null)
        return F;
    else if(coll instanceof Associative)
        return ((Associative) coll).containsKey(key) ? T : F;
    else if(coll instanceof IPersistentSet)
        return ((IPersistentSet) coll).contains(key) ? T : F;
    else if(coll instanceof Map) {
        Map m = (Map) coll;
        return m.containsKey(key) ? T : F;
    }
    else if(key instanceof Number &&
            (coll instanceof String ||
             coll.getClass().isArray())) {
        int n = ((Number) key).intValue();
        return n >= 0 && n < count(coll);
    }
    return F;
}

static public Object find(Object coll, Object key){
    if(coll == null)
        return null;
    else if(coll instanceof Associative)
        return ((Associative) coll).entryAt(key);
    else {
        Map m = (Map) coll;
        if(m.containsKey(key))
            return new MapEntry(key, m.get(key));
        return null;
    }
}

//takes a seq of key,val,key,val

//returns tail starting at val of matching key if found, else null
static public ISeq findKey(Keyword key, ISeq keyvals)

```

```

throws Exception{
    while(keyvals != null) {
        ISeq r = keyvals.next();
        if(r == null)
            throw new Exception("Malformed keyword argslist");
        if(keyvals.first() == key)
            return r;
        keyvals = r.next();
    }
    return null;
}

static public Object dissoc(Object coll, Object key)
throws Exception{
    if(coll == null)
        return null;
    return ((IPersistentMap) coll).without(key);
}

static public Object nth(Object coll, int n){
    if(coll instanceof Indexed)
        return ((Indexed) coll).nth(n);
    return nthFrom(Util.ret1(coll, coll = null), n);
}

static Object nthFrom(Object coll, int n){
    if(coll == null)
        return null;
    else if(coll instanceof CharSequence)
        return Character.valueOf(((CharSequence) coll).charAt(n));
    else if(coll.getClass().isArray())
        return Reflector.prepRet(coll.getClass().getComponentType(),
                               Array.get(coll, n));
    else if(coll instanceof RandomAccess)
        return ((List) coll).get(n);
    else if(coll instanceof Matcher)
        return ((Matcher) coll).group(n);

    else if(coll instanceof Map.Entry) {
        Map.Entry e = (Map.Entry) coll;
        if(n == 0)
            return e.getKey();
        else if(n == 1)
            return e.getValue();
        throw new IndexOutOfBoundsException();
    }

    else if(coll instanceof Sequential) {
        ISeq seq = RT.seq(coll);
        coll = null;
    }
}

```

```
        for(int i = 0; i <= n && seq != null; ++i, seq = seq.next()) {
            if(i == n)
                return seq.first();
        }
        throw new IndexOutOfBoundsException();
    }
    else
        throw new UnsupportedOperationException(
            "nth not supported on this type: " +
            coll.getClass().getSimpleName());
}

static public Object nth(Object coll, int n, Object notFound){
    if(coll instanceof Indexed) {
        Indexed v = (Indexed) coll;
        return v.nth(n, notFound);
    }
    return nthFrom(coll, n, notFound);
}

static Object nthFrom(Object coll, int n, Object notFound){
    if(coll == null)
        return notFound;
    else if(n < 0)
        return notFound;

    else if(coll instanceof CharSequence) {
        CharSequence s = (CharSequence) coll;
        if(n < s.length())
            return Character.valueOf(s.charAt(n));
        return notFound;
    }
    else if(coll.getClass().isArray()) {
        if(n < Array.getLength(coll))
            return
                Reflector.prepRet(coll.getClass().getComponentType(),
                    Array.get(coll, n));
        return notFound;
    }
    else if(coll instanceof RandomAccess) {
        List list = (List) coll;
        if(n < list.size())
            return list.get(n);
        return notFound;
    }
    else if(coll instanceof Matcher) {
        Matcher m = (Matcher) coll;
        if(n < m.groupCount())
            return m.group(n);
        return notFound;
    }
}
```

```

        }

        else if(coll instanceof Map.Entry) {
            Map.Entry e = (Map.Entry) coll;
            if(n == 0)
                return e.getKey();
            else if(n == 1)
                return e.getValue();
            return notFound;
        }
        else if(coll instanceof Sequential) {
            ISeq seq = RT.seq(coll);
            coll = null;
            for(int i = 0;
                i <= n && seq != null;
                ++i, seq = seq.next()) {
                if(i == n)
                    return seq.first();
            }
            return notFound;
        }
        else
            throw new UnsupportedOperationException(
                "nth not supported on this type: " +
                coll.getClass().getSimpleClassName());
    }

    static public Object assocN(int n, Object val, Object coll){
        if(coll == null)
            return null;
        else if(coll instanceof IPersistentVector)
            return ((IPersistentVector) coll).assocN(n, val);
        else if(coll instanceof Object[]) {
            //hmm... this is not persistent
            Object[] array = ((Object[]) coll);
            array[n] = val;
            return array;
        }
        else
            return null;
    }

    static boolean hasTag(Object o, Object tag){
        return Util.equals(tag, RT.get(RT.meta(o), TAG_KEY));
    }

    /**
     * ***** Boxing/casts *****
     */
    static public Object box(Object x){
        return x;
    }
}

```

```
}

static public Character box(char x){
    return Character.valueOf(x);
}

static public Object box(boolean x){
    return x ? T : F;
}

static public Object box(Boolean x){
    return x;// ? T : null;
}

static public Number box(byte x){
    return x;//Num.from(x);
}

static public Number box(short x){
    return x;//Num.from(x);
}

static public Number box(int x){
    return x;//Num.from(x);
}

static public Number box(long x){
    return x;//Num.from(x);
}

static public Number box(float x){
    return x;//Num.from(x);
}

static public Number box(double x){
    return x;//Num.from(x);
}

static public char charCast(Object x){
    if(x instanceof Character)
        return ((Character) x).charValue();

    long n = ((Number) x).longValue();
    if(n < Character.MIN_VALUE || n > Character.MAX_VALUE)
        throw new IllegalArgumentException(
            "Value out of range for char: " + x);

    return (char) n;
}
```

```
static public boolean booleanCast(Object x){
    if(x instanceof Boolean)
        return ((Boolean) x).booleanValue();
    return x != null;
}

static public boolean booleanCast(boolean x){
    return x;
}

static public byte byteCast(Object x){
    if(x instanceof Byte)
        return ((Byte) x).byteValue();
    long n = longCast(x);
    if(n < Byte.MIN_VALUE || n > Byte.MAX_VALUE)
        throw new IllegalArgumentException(
            "Value out of range for byte: " + x);

    return (byte) n;
}

static public short shortCast(Object x){
    if(x instanceof Short)
        return ((Short) x).shortValue();
    long n = longCast(x);
    if(n < Short.MIN_VALUE || n > Short.MAX_VALUE)
        throw new IllegalArgumentException(
            "Value out of range for short: " + x);

    return (short) n;
}

static public int intCast(Object x){
    if(x instanceof Integer)
        return ((Integer)x).intValue();
    if(x instanceof Number)
    {
        long n = longCast(x);
        return intCast(n);
    }
    return ((Character) x).charValue();
}

static public int intCast(char x){
    return x;
}

static public int intCast(byte x){
    return x;
}
```

```
static public int intCast(short x){
    return x;
}

static public int intCast(int x){
    return x;
}

static public int intCast(float x){
    if(x < Integer.MIN_VALUE || x > Integer.MAX_VALUE)
        throw new IllegalArgumentException(
            "Value out of range for int: " + x);
    return (int) x;
}

static public int intCast(long x){
    int i = (int) x;
    if(i != x)
        throw new IllegalArgumentException(
            "Value out of range for int: " + x);
    return i;
}

static public int intCast(double x){
    if(x < Integer.MIN_VALUE || x > Integer.MAX_VALUE)
        throw new IllegalArgumentException(
            "Value out of range for int: " + x);
    return (int) x;
}

static public long longCast(Object x){
    if(x instanceof Integer || x instanceof Long)
        return ((Number) x).longValue();
    else if (x instanceof BigInt)
    {
        BigInt bi = (BigInt) x;
        if(bi.bipart == null)
            return bi.lpart;
        else
            throw new IllegalArgumentException(
                "Value out of range for long: " + x);
    }
    else if (x instanceof BigInteger)
    {
        BigInteger bi = (BigInteger) x;
        if(bi.bitLength() < 64)
            return bi.longValue();
        else
            throw new IllegalArgumentException(
                "Value out of range for BigInteger: " + x);
    }
}
```

```
        "Value out of range for long: " + x);
    }
    return ((Number) x).longValue();
}

static public long longCast(int x){
    return x;
}

static public long longCast(float x){
    if(x < Long.MIN_VALUE || x > Long.MAX_VALUE)
        throw new IllegalArgumentException(
            "Value out of range for long: " + x);
    return (long) x;
}

static public long longCast(long x){
    return x;
}

static public long longCast(double x){
    if(x < Long.MIN_VALUE || x > Long.MAX_VALUE)
        throw new IllegalArgumentException(
            "Value out of range for long: " + x);
    return (long) x;
}

static public float floatCast(Object x){
    if(x instanceof Float)
        return ((Float) x).floatValue();

    double n = ((Number) x).doubleValue();
    if(n < -Float.MAX_VALUE || n > Float.MAX_VALUE)
        throw new IllegalArgumentException(
            "Value out of range for float: " + x);

    return (float) n;
}

static public float floatCast(int x){
    return x;
}

static public float floatCast(float x){
    return x;
}

static public float floatCast(long x){
    return x;
```

```
}

static public float floatCast(double x){
    if(x < -Float.MAX_VALUE || x > Float.MAX_VALUE)
        throw new IllegalArgumentException(
            "Value out of range for float: " + x);

    return (float) x;
}

static public double doubleCast(Object x){
    return ((Number) x).doubleValue();
}

static public double doubleCast(int x){
    return x;
}

static public double doubleCast(float x){
    return x;
}

static public double doubleCast(long x){
    return x;
}

static public double doubleCast(double x){
    return x;
}

static public byte uncheckedByteCast(Object x){
    return ((Number) x).byteValue();
}

static public byte uncheckedByteCast(byte x){
    return x;
}

static public byte uncheckedByteCast(short x){
    return (byte) x;
}

static public byte uncheckedByteCast(int x){
    return (byte) x;
}

static public byte uncheckedByteCast(long x){
    return (byte) x;
}
```

```
static public byte uncheckedByteCast(float x){
    return (byte) x;
}

static public byte uncheckedByteCast(double x){
    return (byte) x;
}

static public short uncheckedShortCast(Object x){
    return ((Number) x).shortValue();
}

static public short uncheckedShortCast(byte x){
    return x;
}

static public short uncheckedShortCast(short x){
    return x;
}

static public short uncheckedShortCast(int x){
    return (short) x;
}

static public short uncheckedShortCast(long x){
    return (short) x;
}

static public short uncheckedShortCast(float x){
    return (short) x;
}

static public short uncheckedShortCast(double x){
    return (short) x;
}

static public char uncheckedCharCast(Object x){
    if(x instanceof Character)
        return ((Character) x).charValue();
    return (char) ((Number) x).longValue();
}

static public char uncheckedCharCast(byte x){
    return (char) x;
}

static public char uncheckedCharCast(short x){
    return (char) x;
}
```

```
static public char uncheckedCharCast(char x){
    return x;
}

static public char uncheckedCharCast(int x){
    return (char) x;
}

static public char uncheckedCharCast(long x){
    return (char) x;
}

static public char uncheckedCharCast(float x){
    return (char) x;
}

static public char uncheckedCharCast(double x){
    return (char) x;
}

static public int uncheckedIntCast(Object x){
    if(x instanceof Number)
        return ((Number)x).intValue();
    return ((Character) x).charValue();
}

static public int uncheckedIntCast(byte x){
    return x;
}

static public int uncheckedIntCast(short x){
    return x;
}

static public int uncheckedIntCast(char x){
    return x;
}

static public int uncheckedIntCast(int x){
    return x;
}

static public int uncheckedIntCast(long x){
    return (int) x;
}

static public int uncheckedIntCast(float x){
    return (int) x;
}
```

```
static public int uncheckedIntCast(double x){
    return (int) x;
}

static public long uncheckedLongCast(Object x){
    return ((Number) x).longValue();
}

static public long uncheckedLongCast(byte x){
    return x;
}

static public long uncheckedLongCast(short x){
    return x;
}

static public long uncheckedLongCast(int x){
    return x;
}

static public long uncheckedLongCast(long x){
    return x;
}

static public long uncheckedLongCast(float x){
    return (long) x;
}

static public long uncheckedLongCast(double x){
    return (long) x;
}

static public float uncheckedFloatCast(Object x){
    return ((Number) x).floatValue();
}

static public float uncheckedFloatCast(byte x){
    return x;
}

static public float uncheckedFloatCast(short x){
    return x;
}

static public float uncheckedFloatCast(int x){
    return x;
}

static public float uncheckedFloatCast(long x){
    return x;
}
```

```
}

static public float uncheckedFloatCast(float x){
    return x;
}

static public float uncheckedFloatCast(double x){
    return (float) x;
}

static public double uncheckedDoubleCast(Object x){
    return ((Number) x).doubleValue();
}

static public double uncheckedDoubleCast(byte x){
    return x;
}

static public double uncheckedDoubleCast(short x){
    return x;
}

static public double uncheckedDoubleCast(int x){
    return x;
}

static public double uncheckedDoubleCast(long x){
    return x;
}

static public double uncheckedDoubleCast(float x){
    return x;
}

static public double uncheckedDoubleCast(double x){
    return x;
}

static public IPersistentMap map(Object... init){
    if(init == null)
        return PersistentArrayMap.EMPTY;
    else if(init.length <= PersistentArrayMap.HASHTABLE_THRESHOLD)
        return PersistentArrayMap.createWithCheck(init);
    return PersistentHashMap.createWithCheck(init);
}

static public IPersistentSet set(Object... init){
    return PersistentHashSet.createWithCheck(init);
}
```

```
static public IPersistentVector vector(Object... init){
    return LazilyPersistentVector.createOwning(init);
}

static public IPersistentVector
subvec(IPersistentVector v, int start, int end){
    if(end < start || start < 0 || end > v.count())
        throw new IndexOutOfBoundsException();
    if(start == end)
        return PersistentVector.EMPTY;
    return new APersistentVector.SubVector(null, v, start, end);
}

/**
 * ***** list support *****
 */

static public ISeq list(){
    return null;
}

static public ISeq list(Object arg1){
    return new PersistentList(arg1);
}

static public ISeq list(Object arg1, Object arg2){
    return listStar(arg1, arg2, null);
}

static public ISeq list(Object arg1, Object arg2, Object arg3){
    return listStar(arg1, arg2, arg3, null);
}

static public ISeq list(Object arg1, Object arg2, Object arg3,
                      Object arg4){
    return listStar(arg1, arg2, arg3, arg4, null);
}

static public ISeq list(Object arg1, Object arg2, Object arg3,
                      Object arg4, Object arg5){
    return listStar(arg1, arg2, arg3, arg4, arg5, null);
}

static public ISeq listStar(Object arg1, ISeq rest){
    return (ISeq) cons(arg1, rest);
}

static public ISeq listStar(Object arg1, Object arg2, ISeq rest){
    return (ISeq) cons(arg1, cons(arg2, rest));
}
```

```

}

static public ISeq listStar(Object arg1, Object arg2, Object arg3,
                           ISeq rest){
    return (ISeq) cons(arg1, cons(arg2, cons(arg3, rest)));
}

static public ISeq listStar(Object arg1, Object arg2, Object arg3,
                           Object arg4, ISeq rest){
    return
        (ISeq) cons(arg1, cons(arg2, cons(arg3, cons(arg4, rest)))));
}

static public ISeq listStar(Object arg1, Object arg2, Object arg3,
                           Object arg4, Object arg5, ISeq rest){
    return
        (ISeq) cons(arg1,
                    cons(arg2,
                        cons(arg3,
                            cons(arg4,
                                cons(arg5, rest))))));
}

static public ISeq arrayToList(Object[] a) throws Exception{
    ISeq ret = null;
    for(int i = a.length - 1; i >= 0; --i)
        ret = (ISeq) cons(a[i], ret);
    return ret;
}

static public Object[] object_array(Object sizeOrSeq){
    if(sizeOrSeq instanceof Number)
        return new Object[((Number) sizeOrSeq).intValue()];
    else
    {
        ISeq s = RT.seq(sizeOrSeq);
        int size = RT.count(s);
        Object[] ret = new Object[size];
        for(int i = 0; i < size && s != null; i++, s = s.next())
            ret[i] = s.first();
        return ret;
    }
}

static public Object[] toArray(Object coll) throws Exception{
    if(coll == null)
        return EMPTY_ARRAY;
    else if(coll instanceof Object[])
        return (Object[]) coll;
    else if(coll instanceof Collection)

```

```

        return ((Collection) coll).toArray();
    else if(coll instanceof Map)
        return ((Map) coll).entrySet().toArray();
    else if(coll instanceof String) {
        char[] chars = ((String) coll).toCharArray();
        Object[] ret = new Object[chars.length];
        for(int i = 0; i < chars.length; i++)
            ret[i] = chars[i];
        return ret;
    }
    else if(coll.getClass().isArray()) {
        ISeq s = (seq(coll));
        Object[] ret = new Object[count(s)];
        for(int i = 0; i < ret.length; i++, s = s.next())
            ret[i] = s.first();
        return ret;
    }
    else
        throw new Exception("Unable to convert: " +
                            coll.getClass() + " to Object[]");
}

static public Object[] seqToArray(ISeq seq){
    int len = length(seq);
    Object[] ret = new Object[len];
    for(int i = 0; seq != null; ++i, seq = seq.next())
        ret[i] = seq.first();
    return ret;
}

static public Object seqToTypedArray(ISeq seq) throws Exception{
    Class type = (seq != null) ? seq.first().getClass() : Object.class;
    return seqToTypedArray(type, seq);
}

static public Object seqToTypedArray(Class type, ISeq seq)
throws Exception{
    Object ret = Array.newInstance(type, length(seq));
    if(type == Integer.TYPE){
        for(int i = 0; seq != null; ++i, seq=seq.next()){
            Array.set(ret, i, intCast(seq.first()));
        }
    } else if(type == Byte.TYPE) {
        for(int i = 0; seq != null; ++i, seq=seq.next()){
            Array.set(ret, i, byteCast(seq.first()));
        }
    } else if(type == Float.TYPE) {
        for(int i = 0; seq != null; ++i, seq=seq.next()){
            Array.set(ret, i, floatCast(seq.first()));
        }
    }
}

```

```

    } else if(type == Short.TYPE) {
        for(int i = 0; seq != null; ++i, seq=seq.next()){
            Array.set(ret, i, shortCast(seq.first()));
        }
    } else if(type == Character.TYPE) {
        for(int i = 0; seq != null; ++i, seq=seq.next()){
            Array.set(ret, i, charCast(seq.first()));
        }
    } else {
        for(int i = 0; seq != null; ++i, seq=seq.next()){
            Array.set(ret, i, seq.first());
        }
    }
    return ret;
}

static public int length(ISeq list){
    int i = 0;
    for(ISeq c = list; c != null; c = c.next()) {
        i++;
    }
    return i;
}

static public int boundedLength(ISeq list, int limit)
throws Exception{
    int i = 0;
    for(ISeq c = list; c != null && i <= limit; c = c.next()) {
        i++;
    }
    return i;
}

//// reader support //// 

static Character readRet(int ret){
    if(ret == -1)
        return null;
    return box((char) ret);
}

static public Character readChar(Reader r) throws Exception{
    int ret = r.read();
    return readRet(ret);
}

static public Character peekChar(Reader r) throws Exception{
    int ret;
    if(r instanceof PushbackReader) {
        ret = r.read();
    }
}

```

```
        ((PushbackReader) r).unread(ret);
    }
    else {
        r.mark(1);
        ret = r.read();
        r.reset();
    }

    return readRet(ret);
}

static public int getLineNumber(Reader r){
    if(r instanceof LineNumberingPushbackReader)
        return ((LineNumberingPushbackReader) r).getLineNumber();
    return 0;
}

static public LineNumberingPushbackReader
getLineNumberingReader(Reader r){
    if(isLineNumberingReader(r))
        return (LineNumberingPushbackReader) r;
    return new LineNumberingPushbackReader(r);
}

static public boolean isLineNumberingReader(Reader r){
    return r instanceof LineNumberingPushbackReader;
}

static public String resolveClassNameInContext(String className){
    //todo - look up in context var
    return className;
}

static public boolean suppressRead(){
    //todo - look up in suppress-read var
    return false;
}

static public String printString(Object x){
    try {
        StringWriter sw = new StringWriter();
        print(x, sw);
        return sw.toString();
    }
    catch(Exception e) {
        throw new RuntimeException(e);
    }
}

static public Object readString(String s){
```

```

PushbackReader r = new PushbackReader(new StringReader(s));
try {
    return LispReader.read(r, true, null, false);
}
catch(Exception e) {
    throw new RuntimeException(e);
}
}

static public void print(Object x, Writer w) throws Exception{
    //call multimethod
    if(PRINT_INITIALIZED.isBound() &&
       RT.booleanCast(PRINT_INITIALIZED.deref()))
        PR_ON.invoke(x, w);
    /*
    else {
        boolean readably = booleanCast(PRINT_READABLY.deref());
        if(x instanceof Obj) {
            Obj o = (Obj) x;
            if(RT.count(o.meta()) > 0 &&
               ((readably && booleanCast(PRINT_META.deref())))
               || booleanCast(PRINT_DUP.deref())))
                IPersistentMap meta = o.meta();
                w.write("#^");
                if(meta.count() == 1 && meta.containsKey(TAG_KEY))
                    print(meta.valAt(TAG_KEY), w);
                else
                    print(meta, w);
                w.write(' ');
            }
        }
        if(x == null)
            w.write("nil");
        else if(x instanceof ISeq || x instanceof IPersistentList) {
            w.write('(');
            printInnerSeq(seq(x), w);
            w.write(')');
        }
        else if(x instanceof String) {
            String s = (String) x;
            if(!readably)
                w.write(s);
            else {
                w.write('\'');
                //w.write(x.toString());
                for(int i = 0; i < s.length(); i++) {
                    char c = s.charAt(i);
                    switch(c) {
                        case '\n':
                            w.write("\\\\n");
                    }
                }
            }
        }
    }
}

```

```
        break;
    case '\t':
        w.write("\t");
        break;
    case '\r':
        w.write("\r");
        break;
    case '':
        w.write("\\\"");
        break;
    case '\\':
        w.write("\\\\");
        break;
    case '\f':
        w.write("\f");
        break;
    case '\b':
        w.write("\b");
        break;
    default:
        w.write(c);
    }
}
w.write('\'');
}
}
else if(x instanceof IPersistentMap) {
    w.write('{');
    for(ISeq s = seq(x); s != null; s = s.next()) {
        IMapEntry e = (IMapEntry) s.first();
        print(e.key(), w);
        w.write(' ');
        print(e.val(), w);
        if(s.next() != null)
            w.write(", ");
    }
    w.write('}');
}
else if(x instanceof IPersistentVector) {
    IPersistentVector a = (IPersistentVector) x;
    w.write('[');
    for(int i = 0; i < a.count(); i++) {
        print(a.nth(i), w);
        if(i < a.count() - 1)
            w.write(' ');
    }
    w.write(']');
}
else if(x instanceof IPersistentSet) {
    w.write("#{");
}
```

```

        for(ISeq s = seq(x); s != null; s = s.next()) {
            print(s.first(), w);
            if(s.next() != null)
                w.write(" ");
        }
        w.write('}');
    }
    else if(x instanceof Character) {
        char c = ((Character) x).charValue();
        if(!readably)
            w.write(c);
        else {
            w.write('\\');
            switch(c) {
                case '\n':
                    w.write("newline");
                    break;
                case '\t':
                    w.write("tab");
                    break;
                case ' ':
                    w.write("space");
                    break;
                case '\b':
                    w.write("backspace");
                    break;
                case '\f':
                    w.write("formfeed");
                    break;
                case '\r':
                    w.write("return");
                    break;
                default:
                    w.write(c);
            }
        }
    }
    else if(x instanceof Class) {
        w.write("#=");
        w.write(((Class) x).getName());
    }
    else if(x instanceof BigDecimal && readably) {
        w.write(x.toString());
        w.write('M');
    }
    else if(x instanceof BigInt && readably) {
        w.write(x.toString());
        w.write('N');
    }
    else if(x instanceof BigInteger && readably) {

```

```

        w.write(x.toString());
        w.write("BIGINT");
    }
    else if(x instanceof Var) {
        Var v = (Var) x;
        w.write("#=(var " + v.ns.name + "/" + v.sym + ")");
    }
    else if(x instanceof Pattern) {
        Pattern p = (Pattern) x;
        w.write("#\" " + p.pattern() + "\"");
    }
    else w.write(x.toString());
}
/**/
}

private static void printInnerSeq(ISeq x, Writer w)
throws Exception{
    for(ISeq s = x; s != null; s = s.next()) {
        print(s.first(), w);
        if(s.next() != null)
            w.write(' ');
    }
}

static public void formatAesthetic(Writer w, Object obj)
throws IOException{
    if(obj == null)
        w.write("null");
    else
        w.write(obj.toString());
}

static public void formatStandard(Writer w, Object obj)
throws IOException{
    if(obj == null)
        w.write("null");
    else if(obj instanceof String) {
        w.write('\"');
        w.write((String) obj);
        w.write('\"');
    }
    else if(obj instanceof Character) {
        w.write('\\');
        char c = ((Character) obj).charValue();
        switch(c) {
            case '\n':
                w.write("newline");
                break;
            case '\t':

```

```

        w.write("tab");
        break;
    case ' ':
        w.write("space");
        break;
    case '\b':
        w.write("backspace");
        break;
    case '\f':
        w.write("formfeed");
        break;
    default:
        w.write(c);
    }
}
else
    w.write(obj.toString());
}

static public Object format(Object o, String s, Object... args)
throws Exception{
    Writer w;
    if(o == null)
        w = new StringWriter();
    else if(Util.equals(o, T))
        w = (Writer) OUT.deref();
    else
        w = (Writer) o;
    doFormat(w, s, ArraySeq.create(args));
    if(o == null)
        return w.toString();
    return null;
}

static public ISeq doFormat(Writer w, String s, ISeq args)
throws Exception{
    for(int i = 0; i < s.length(); ) {
        char c = s.charAt(i++);
        switch(Character.toLowerCase(c)) {
            case '~':
                char d = s.charAt(i++);
                switch(Character.toLowerCase(d)) {
                    case '%':
                        w.write('\n');
                        break;
                    case 't':
                        w.write('\t');
                        break;
                    case 'a':
                        if(args == null)

```

```

        throw new IllegalArgumentException(
                "Missing argument");
    RT.formatAesthetic(w, RT.first(args));
    args = RT.next(args);
    break;
  case 's':
    if(args == null)
      throw new IllegalArgumentException(
              "Missing argument");
    RT.formatStandard(w, RT.first(args));
    args = RT.next(args);
    break;
  case '{':
    int j = s.indexOf("}", i);      //note - does not nest
    if(j == -1)
      throw new IllegalArgumentException("Missing }");
    String subs = s.substring(i, j);
    for(ISeq sargs = RT.seq(RT.first(args)); sargs != null;)
      sargs = doFormat(w, subs, sargs);
    args = RT.next(args);
    i = j + 2; //skip }
    break;
  case '^':
    if(args == null)
      return null;
    break;
  case '~':
    w.write('~');
    break;
  default:
    throw new IllegalArgumentException(
            "Unsupported ~ directive: " + d);
  }
  break;
default:
  w.write(c);
}
}
return args;
}
/////////////////////////////// values /////////////////////
static public Object[] setValues(Object... vals){
  //ThreadLocalData.setValues(vals);
  if(vals.length > 0)
    return vals;//[0];
  return null;
}

```

```

static public ClassLoader makeClassLoader(){
    return (ClassLoader) AccessController
        .doPrivileged(new PrivilegedAction(){
            public Object run(){
                try{
                    Var.pushThreadBindings(
                        RT.map(USE_CONTEXT_CLASSLOADER, RT.T));
                    // getRootClassLoader();
                    return new DynamicClassLoader(baseLoader());
                }
                finally{
                    Var.popThreadBindings();
                }
            }
        });
}

static public ClassLoader baseLoader(){
    if(Compiler.LOADER.isBound())
        return (ClassLoader) Compiler.LOADER.deref();
    else if(booleanCast(USE_CONTEXT_CLASSLOADER.deref()))
        return Thread.currentThread().getContextClassLoader();
    return Compiler.class.getClassLoader();
}

static public Class classForName(String name)
throws ClassNotFoundException{

    return Class.forName(name, true, baseLoader());
}

static public Class loadClassForName(String name)
throws ClassNotFoundException{
    try
    {
        Class.forName(name, false, baseLoader());
    }
    catch(ClassNotFoundException e)
    {
        return null;
    }
    return Class.forName(name, true, baseLoader());
}

static public float aget(float[] xs, int i){
    return xs[i];
}

static public float aset(float[] xs, int i, float v){
    xs[i] = v;
}

```

```
        return v;
    }

    static public int alength(float[] xs){
        return xs.length;
    }

    static public float[] aclone(float[] xs){
        return xs.clone();
    }

    static public double aget(double[] xs, int i){
        return xs[i];
    }

    static public double aset(double[] xs, int i, double v){
        xs[i] = v;
        return v;
    }

    static public int alength(double[] xs){
        return xs.length;
    }

    static public double[] aclone(double[] xs){
        return xs.clone();
    }

    static public int aget(int[] xs, int i){
        return xs[i];
    }

    static public int aset(int[] xs, int i, int v){
        xs[i] = v;
        return v;
    }

    static public int alength(int[] xs){
        return xs.length;
    }

    static public int[] aclone(int[] xs){
        return xs.clone();
    }

    static public long aget(long[] xs, int i){
        return xs[i];
    }

    static public long aset(long[] xs, int i, long v){
```

```
    xs[i] = v;
    return v;
}

static public int alength(long[] xs){
    return xs.length;
}

static public long[] aclone(long[] xs){
    return xs.clone();
}

static public char aget(char[] xs, int i){
    return xs[i];
}

static public char aset(char[] xs, int i, char v){
    xs[i] = v;
    return v;
}

static public int alength(char[] xs){
    return xs.length;
}

static public char[] aclone(char[] xs){
    return xs.clone();
}

static public byte aget(byte[] xs, int i){
    return xs[i];
}

static public byte aset(byte[] xs, int i, byte v){
    xs[i] = v;
    return v;
}

static public int alength(byte[] xs){
    return xs.length;
}

static public byte[] aclone(byte[] xs){
    return xs.clone();
}

static public short aget(short[] xs, int i){
    return xs[i];
}
```

```
static public short aset(short[] xs, int i, short v){
    xs[i] = v;
    return v;
}

static public int alength(short[] xs){
    return xs.length;
}

static public short[] aclone(short[] xs){
    return xs.clone();
}

static public boolean aget(boolean[] xs, int i){
    return xs[i];
}

static public boolean aset(boolean[] xs, int i, boolean v){
    xs[i] = v;
    return v;
}

static public int alength(boolean[] xs){
    return xs.length;
}

static public boolean[] aclone(boolean[] xs){
    return xs.clone();
}

static public Object aget(Object[] xs, int i){
    return xs[i];
}

static public Object aset(Object[] xs, int i, Object v){
    xs[i] = v;
    return v;
}

static public int alength(Object[] xs){
    return xs.length;
}

static public Object[] aclone(Object[] xs){
    return xs.clone();
}

}
```

9.96 Script.java

— Script.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Oct 18, 2007 */

package clojure.lang;

import clojure.main;

public class Script {

    public static void main(String[] args) throws Exception{
        main.legacy_script(args);
    }
}
```

9.97 SeqEnumeration.java

(Enumeration [1723])

— SeqEnumeration.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 3, 2008 */

package clojure.lang;

import java.util.Enumeration;

public class SeqEnumeration implements Enumeration{
    ISeq seq;

    public SeqEnumeration(ISeq seq){
        this.seq = seq;
    }
}
```

```

public boolean hasMoreElements(){
    return seq != null;
}

public Object nextElement(){
    Object ret = RT.first(seq);
    seq = RT.next(seq);
    return ret;
}
}

```

9.98 SeqIterator.java

(Iterator [1723])
— SeqIterator.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jun 19, 2007 */

package clojure.lang;

import java.util.Iterator;
import java.util.NoSuchElementException;

public class SeqIterator implements Iterator{

ISeq seq;

public SeqIterator(ISeq seq){
    this.seq = seq;
}

public boolean hasNext(){
    return seq != null;
}

public Object next() throws NoSuchElementException {
    if(seq == null)
        throw new NoSuchElementException();
    Object ret = RT.first(seq);
    seq = RT.next(seq);
    return ret;
}

```

```
public void remove(){
    throw new UnsupportedOperationException();
}
```

9.99 Seqable.java

— Seqable.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Jan 28, 2009 */

package clojure.lang;

public interface Seqable {
    ISeq seq();
}
```

9.100 Sequential.java

— Sequential.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure.lang;

public interface Sequential {
}
```

9.101 Settable.java

— Settable.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Dec 31, 2008 */

package clojure.lang;

public interface Settable {
    Object doSet(Object val) throws Exception;
    Object doReset(Object val) throws Exception;
}
```

—————

9.102 Sorted.java

— Sorted.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Apr 15, 2008 */

package clojure.lang;

import java.util.Comparator;

public interface Sorted{
    Comparator comparator();

    Object entryKey(Object entry);

    ISeq seq(boolean ascending);

    ISeq seqFrom(Object key, boolean ascending);
}
```

—————

9.103 StringSeq.java

(ASeq [571]) (IndexedSeq [799])
— StringSeq.java —

/*

```
\getchunk{Clojure Copyright}
*/
/* rich Dec 6, 2007 */

package clojure.lang;

public class StringSeq extends ASeq implements IndexedSeq{
    public final CharSequence s;
    public final int i;

    static public StringSeq create(CharSequence s){
        if(s.length() == 0)
            return null;
        return new StringSeq(null, s, 0);
    }

    StringSeq(IPersistentMap meta, CharSequence s, int i){
        super(meta);
        this.s = s;
        this.i = i;
    }

    public Obj withMeta(IPersistentMap meta){
        if(meta == meta())
            return this;
        return new StringSeq(meta, s, i);
    }

    public Object first(){
        return Character.valueOf(s.charAt(i));
    }

    public ISeq next(){
        if(i + 1 < s.length())
            return new StringSeq(_meta, s, i + 1);
        return null;
    }

    public int index(){
        return i;
    }

    public int count(){
        return s.length() - i;
    }
}
```



9.104 Symbol.java

(AFn [509]) (IObj [800]) (Comparable [1723]) (Named [861]) (Serializable [1723])
— Symbol.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Mar 25, 2006 11:42:47 AM */

package clojure.lang;

import java.io.Serializable;
import java.io.ObjectStreamException;

public class Symbol
    implements AFn, IObj, Comparable, Named, Serializable{

    \getchunk{Symbol private data}

    \getchunk{Symbol method toString}

    public String getNamespace(){
        return ns;
    }

    public String getName(){
        return name;
    }

    // the create thunks preserve binary compatibility with code compiled
    // against earlier version of Clojure and can be removed (at some point).
    static public Symbol create(String ns, String name) {
        return Symbol.intern(ns, name);
    }

    static public Symbol create(String nsname) {
        return Symbol.intern(nsname);
    }

    \getchunk{Symbol intern method 2}

    \getchunk{Symbol intern method}

    private Symbol(String ns_interned, String name_interned){
        this.name = name_interned;
        this.ns = ns_interned;
        this.hash = Util.hashCombine(name.hashCode(), Util.hash(ns));
        this._meta = null;
    }
}
```

```
}

\getchunk{Symbol method equals}

public int hashCode(){
    return hash;
}

public IObj withMeta(IPersistentMap meta){
    return new Symbol(meta, ns, name);
}

private Symbol(IPersistentMap meta, String ns, String name){
    this.name = name;
    this.ns = ns;
    this._meta = meta;
    this.hash = Util.hashCombine(name.hashCode(), Util.hash(ns));
}

public int compareTo(Object o){
    Symbol s = (Symbol) o;
    if(this.equals(o))
        return 0;
    if(this.ns == null && s.ns != null)
        return -1;
    if(this.ns != null)
    {
        if(s.ns == null)
            return 1;
        int nsc = this.ns.compareTo(s.ns);
        if(nsc != 0)
            return nsc;
    }
    return this.name.compareTo(s.name);
}

private Object readResolve() throws ObjectStreamException{
    return intern(ns, name);
}

public Object invoke(Object obj) throws Exception{
    return RT.get(obj, this);
}

public Object invoke(Object obj, Object notFound)
throws Exception{
    return RT.get(obj, this, notFound);
}

public IPersistentMap meta(){
```

```

        return _meta;
    }
}

```

—————

9.105 TransactionalHashMap.java

(AbstractMap [1723]) (ConcurrentMap [1723])
 — TransactionalHashMap.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 31, 2008 */

package clojure.lang;

import java.util.concurrent.ConcurrentMap;
import java.util.*;

public class TransactionalHashMap<K, V>
  extends AbstractMap<K, V> implements ConcurrentMap<K, V>{
final Ref[] bins;

IPersistentMap mapAt(int bin){
    return (IPersistentMap) bins[bin].deref();
}

final int binFor(Object k){
    //spread hashes, a la Cliff Click
    int h = k.hashCode();
    h ^= (h >>> 20) ^ (h >>> 12);
    h ^= (h >>> 7) ^ (h >>> 4);
    return h % bins.length;
//    return k.hashCode() % bins.length;
}

Entry entryAt(Object k){
    return mapAt(binFor(k)).entryAt(k);
}

public TransactionalHashMap() throws Exception{
    this(421);
}

public TransactionalHashMap(int nBins) throws Exception{

```

```

bins = new Ref[nBins];
for(int i = 0; i < nBins; i++)
    bins[i] = new Ref(PersistentHashMap.EMPTY);
}

public TransactionalHashMap(Map<? extends K, ? extends V> m)
throws Exception{
    this(m.size());
    putAll(m);
}

public int size(){
    int n = 0;
    for(int i = 0; i < bins.length; i++)
    {
        n += mapAt(i).count();
    }
    return n;
}

public boolean isEmpty(){
    return size() == 0;
}

public boolean containsKey(Object k){
    return entryAt(k) != null;
}

public V get(Object k){
    Entry e = entryAt(k);
    if(e != null)
        return (V) e.getValue();
    return null;
}

public V put(K k, V v){
    Ref r = bins[binFor(k)];
    IPersistentMap map = (IPersistentMap) r.deref();
    Object ret = map.valAt(k);
    r.set(map.assoc(k, v));
    return (V) ret;
}

public V remove(Object k){
    Ref r = bins[binFor(k)];
    IPersistentMap map = (IPersistentMap) r.deref();
    Object ret = map.valAt(k);
    //checked exceptions are a bad idea, especially in an interface
    try
    {

```

```

        r.set(map.without(k));
    }
    catch(Exception e)
    {
        throw new RuntimeException(e);
    }
    return (V) ret;
}

public void putAll(Map<? extends K, ? extends V> map){
    for(Iterator i = map.entrySet().iterator(); i.hasNext();)
    {
        Entry<K, V> e = (Entry) i.next();
        put(e.getKey(), e.getValue());
    }
}

public void clear(){
    for(int i = 0; i < bins.length; i++)
    {
        Ref r = bins[i];
        IPersistentMap map = (IPersistentMap) r.deref();
        if(map.count() > 0)
        {
            r.set(PersistentHashMap.EMPTY);
        }
    }
}

public Set<Entry<K, V>> entrySet(){
    final ArrayList<Map.Entry<K, V>> entries =
        new ArrayList(bins.length);
    for(int i = 0; i < bins.length; i++)
    {
        IPersistentMap map = mapAt(i);
        if(map.count() > 0)
            entries.addAll((Collection) RT.seq(map));
    }
    return new AbstractSet<Entry<K, V>>(){
        public Iterator iterator(){
            return Collections.unmodifiableList(entries).iterator();
        }

        public int size(){
            return entries.size();
        }
    };
}

public V putIfAbsent(K k, V v){

```

```

Ref r = bins[binFor(k)];
IPersistentMap map = (IPersistentMap) r.deref();
Entry e = map.entryAt(k);
if(e == null)
{
    r.set(map.assoc(k, v));
    return null;
}
else
    return (V) e.getValue();
}

public boolean remove(Object k, Object v){
Ref r = bins[binFor(k)];
IPersistentMap map = (IPersistentMap) r.deref();
Entry e = map.entryAt(k);
if(e != null && e.getValue().equals(v))
{
    //checked exceptions are a bad idea, especially
    //in an interface
    try
    {
        r.set(map.without(k));
    }
    catch(Exception ex)
    {
        throw new RuntimeException(ex);
    }
    return true;
}
return false;
}

public boolean replace(K k, V oldv, V newv){
Ref r = bins[binFor(k)];
IPersistentMap map = (IPersistentMap) r.deref();
Entry e = map.entryAt(k);
if(e != null && e.getValue().equals(oldv))
{
    r.set(map.assoc(k, newv));
    return true;
}
return false;
}

public V replace(K k, V v){
Ref r = bins[binFor(k)];
IPersistentMap map = (IPersistentMap) r.deref();
Entry e = map.entryAt(k);
if(e != null)

```

```

    {
      r.set(map.assoc(k, v));
      return (V) e.getValue();
    }
    return null;
}
}

```

9.106 Util.java

— Util.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Apr 19, 2008 */

package clojure.lang;

import java.math.BigInteger;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.lang.ref.SoftReference;
import java.lang.ref.ReferenceQueue;

public class Util{
static public boolean equiv(Object k1, Object k2){
  if(k1 == k2)
    return true;
  if(k1 != null)
  {
    if(k1 instanceof Number && k2 instanceof Number)
      return Numbers.equal((Number)k1, (Number)k2);
    else if(k1 instanceof IPersistentCollection ||
            k2 instanceof IPersistentCollection)
      return pcequiv(k1,k2);
    return k1.equals(k2);
  }
  return false;
}

static public boolean equiv(long k1, long k2){
  return k1 == k2;
}

```

```
static public boolean equiv(Object k1, long k2){
    return equiv(k1, (Object)k2);
}

static public boolean equiv(long k1, Object k2){
    return equiv((Object)k1, k2);
}

static public boolean equiv(double k1, double k2){
    return k1 == k2;
}

static public boolean equiv(Object k1, double k2){
    return equiv(k1, (Object)k2);
}

static public boolean equiv(double k1, Object k2){
    return equiv((Object)k1, k2);
}

static public boolean pcequiv(Object k1, Object k2){
    if(k1 instanceof IPersistentCollection)
        return ((IPersistentCollection)k1).equiv(k2);
    return ((IPersistentCollection)k2).equiv(k1);
}

static public boolean equals(Object k1, Object k2){
    if(k1 == k2)
        return true;
    return k1 != null && k1.equals(k2);
}

static public boolean identical(Object k1, Object k2){
    return k1 == k2;
}

static public Class classOf(Object x){
    if(x != null)
        return x.getClass();
    return null;
}

static public int compare(Object k1, Object k2){
    if(k1 == k2)
        return 0;
    if(k1 != null)
    {
        if(k2 == null)
            return 1;
```

```
    if(k1 instanceof Number)
        return Numbers.compare((Number) k1, (Number) k2);
    return ((Comparable) k1).compareTo(k2);
}
return -1;
}

static public int hash(Object o){
    if(o == null)
        return 0;
    return o.hashCode();
}

static public int hashCombine(int seed, int hash){
    //a la boost
    seed ^= hash + 0x9e3779b9 + (seed << 6) + (seed >> 2);
    return seed;
}

static public boolean isPrimitive(Class c){
    return c != null && c.isPrimitive() && !(c == Void.TYPE);
}

static public boolean isInteger(Object x){
    return x instanceof Integer
        || x instanceof Long
        || x instanceof BigInt
        || x instanceof BigInteger;
}

static public Object ret1(Object ret, Object nil){
    return ret;
}

static public ISeq ret1(ISeq ret, Object nil){
    return ret;
}

static public <K,V> void
clearCache(ReferenceQueue rq,
    ConcurrentHashMap<K, SoftReference<V>> cache){
    //cleanup any dead entries
    if(rq.poll() != null)
    {
        while(rq.poll() != null)
            ;
        for(Map.Entry<K, SoftReference<V>> e : cache.entrySet())
        {
            if(e.getValue().get() == null)
                cache.remove(e.getKey(), e.getValue());
        }
    }
}
```

```

        }
    }
}

-----

```

9.107 Var.java

(ARef [553]) (IFn [774]) (IRef [805]) (Settable [1140])
— Var.java —

```

/*
\getchunk{Clojure Copyright}
*/
/* rich Jul 31, 2007 */

package clojure.lang;

import java.util.concurrent.atomic.AtomicBoolean;

public final class Var extends ARef implements IFn, IRef, Settable{

    static class TBox{

        volatile Object val;
        final Thread thread;

        public TBox(Thread t, Object val){
            this.thread = t;
            this.val = val;
        }
    }

    static public class Unbound extends AFn{
        final public Var v;

        public Unbound(Var v){
            this.v = v;
        }

        public String toString(){
            return "Unbound: " + v;
        }

        public Object throwArity(int n){
            throw new IllegalStateException(

```

```
        "Attempting to call unbound fn: " + v);
    }

static class Frame{
    //Var->TBox
    Associative bindings;
    //Var->val
//    Associative frameBindings;
    Frame prev;

    public Frame(){
        this(PersistentHashMap.EMPTY, null);
    }

    public Frame(Associative bindings, Frame prev){
//        this.frameBindings = frameBindings;
        this.bindings = bindings;
        this.prev = prev;
    }
}

static final ThreadLocal<Frame> dvals = new ThreadLocal<Frame>(){

    protected Frame initialValue(){
        return new Frame();
    }
};

static public volatile int rev = 0;

static Keyword privateKey = Keyword.intern(null, "private");
static IPersistentMap privateMeta =
    new PersistentArrayMap(new Object[]{privateKey, Boolean.TRUE});
static Keyword macroKey = Keyword.intern(null, "macro");
static Keyword nameKey = Keyword.intern(null, "name");
static Keyword nsKey = Keyword.intern(null, "ns");
//static Keyword tagKey = Keyword.intern(null, "tag");

private volatile Object root;

volatile boolean dynamic = false;
transient final AtomicBoolean threadBound;
public final Symbol sym;
public final Namespace ns;

//IPersistentMap _meta;

public static Object getThreadBindingFrame(){
```

```

        Frame f = dvals.get();
        if(f != null)
            return f;
        return new Frame();
    }

    public static void resetThreadBindingFrame(Object frame){
        dvals.set((Frame) frame);
    }

    public Var setDynamic(){
        this.dynamic = true;
        return this;
    }

    public Var setDynamic(boolean b){
        this.dynamic = b;
        return this;
    }

    public final boolean isDynamic(){
        return dynamic;
    }

    public static Var intern	Namespace ns, Symbol sym, Object root){
        return intern(ns, sym, root, true);
    }

    public static Var intern	Namespace ns,
                                Symbol sym,
                                Object root,
                                boolean replaceRoot){
        Var dvout = ns.intern(sym);
        if(!dvout.hasRoot() || replaceRoot)
            dvout.bindRoot(root);
        return dvout;
    }

    public String toString(){
        if(ns != null)
            return "#" + ns.name + "/" + sym;
        return "#<Var: " + (sym != null
                            ? sym.toString()
                            : "--unnamed--") + ">";
    }

    public static Var find(Symbol nsQualifiedSym){
        if(nsQualifiedSym.ns == null)
            throw new IllegalArgumentException(

```

```
    "Symbol must be namespace-qualified");
Namespace ns = Namespace.find(Symbol.intern(nsQualifiedSym.ns));
if(ns == null)
    throw new IllegalArgumentException(
        "No such namespace: " + nsQualifiedSym.ns);
return ns.findInternedVar(Symbol.intern(nsQualifiedSym.name));
}

public static Var intern(Symbol nsName, Symbol sym){
    Namespace ns = Namespace.findOrCreate(nsName);
    return intern(ns, sym);
}

public static Var internPrivate(String nsName, String sym){
    Namespace ns = Namespace.findOrCreate(Symbol.intern(nsName));
    Var ret = intern(ns, Symbol.intern(sym));
    ret.setMeta(privateMeta);
    return ret;
}

public static Var intern(Namespace ns, Symbol sym){
    return ns.intern(sym);
}

public static Var create(){
    return new Var(null, null);
}

public static Var create(Object root){
    return new Var(null, null, root);
}

Var(Namespace ns, Symbol sym){
    this.ns = ns;
    this.sym = sym;
    this.threadBound = new AtomicBoolean(false);
    this.root = new Unbound(this);
    setMeta(PersistentHashMap.EMPTY);
}

Var(Namespace ns, Symbol sym, Object root){
    this(ns, sym);
    this.root = root;
    ++rev;
}

public boolean isBound(){
    return
        hasRoot() ||
```

```

        (threadBound.get() && dvals.get().bindings.containsKey(this));
    }

    final public Object get(){
        if(!threadBound.get())
            return root;
        return deref();
    }

    final public Object deref(){
        TBox b = getThreadBinding();
        if(b != null)
            return b.val;
        return root;
    }

    public void setValidator(IFn vf){
        if(hasRoot())
            validate(vf, root);
        validator = vf;
    }

    public Object alter(IFn fn, ISeq args) throws Exception{
        set(fn.applyTo(RT.cons(deref(), args)));
        return this;
    }

    public Object set(Object val){
        validate(getValidator(), val);
        TBox b = getThreadBinding();
        if(b != null)
        {
            if(Thread.currentThread() != b.thread)
                throw new IllegalStateException(String.format(
                    "Can't set!: %s from non-binding thread", sym));
            return (b.val = val);
        }
        throw new IllegalStateException(String.format(
            "Can't change/establish root binding of: %s with set", sym));
    }

    public Object doSet(Object val) throws Exception {
        return set(val);
    }

    public Object doReset(Object val) throws Exception {
        bindRoot(val);
        return val;
    }
}

```

```
public void setMeta(IPersistentMap m) {
    //ensure these basis keys
    resetMeta(m.assoc(nameKey, sym).assoc(nsKey, ns));
}

public void setMacro() {
    try
    {
        alterMeta(assoc, RT.list(macroKey, RT.T));
    }
    catch (Exception e)
    {
        throw new RuntimeException(e);
    }
}

public boolean isMacro(){
    return RT.booleanCast(meta().valAt(macroKey));
}

//public void setExported(boolean state){
//    _meta = _meta.assoc(privateKey, state);
//}

public boolean isPublic(){
    return !RT.booleanCast(meta().valAt(privateKey));
}

final public Object getRawRoot(){
    return root;
}

public Object getTag(){
    return meta().valAt(RT.TAG_KEY);
}

public void setTag(Symbol tag) {
    try
    {
        alterMeta(assoc, RT.list(RT.TAG_KEY, tag));
    }
    catch (Exception e)
    {
        throw new RuntimeException(e);
    }
}

final public boolean hasRoot(){
    return !(root instanceof Unbound);
}
```

```

//binding root always clears macro flag
synchronized public void bindRoot(Object root){
    validate(getValidator(), root);
    Object oldroot = this.root;
    this.root = root;
    ++rev;
    try
    {
        alterMeta(dissoc, RT.list(macroKey));
    }
    catch (Exception e)
    {
        throw new RuntimeException(e);
    }
    notifyWatches(oldroot,this.root);
}

synchronized void swapRoot(Object root){
    validate(getValidator(), root);
    Object oldroot = this.root;
    this.root = root;
    ++rev;
    notifyWatches(oldroot,root);
}

synchronized public void unbindRoot(){
    this.root = new Unbound(this);
    ++rev;
}

synchronized public void commuteRoot(IFn fn) throws Exception{
    Object newRoot = fn.invoke(root);
    validate(getValidator(), newRoot);
    Object oldroot = root;
    this.root = newRoot;
    ++rev;
    notifyWatches(oldroot,newRoot);
}

synchronized public Object alterRoot(IFn fn, ISeq args)
throws Exception{
    Object newRoot = fn.applyTo(RT.cons(root, args));
    validate(getValidator(), newRoot);
    Object oldroot = root;
    this.root = newRoot;
    ++rev;
    notifyWatches(oldroot,newRoot);
    return newRoot;
}

```

```

public static void pushThreadBindings(Associative bindings){
    Frame f = dvals.get();
    Associative bmap = f.bindings;
    for(ISeq bs = bindings.seq(); bs != null; bs = bs.next())
    {
        IMapEntry e = (IMapEntry) bs.first();
        Var v = (Var) e.key();
        if(!v.dynamic)
            throw new IllegalStateException(String.format(
                "Can't dynamically bind non-dynamic var: %s/%s",
                v.ns, v.sym));
        v.validate(v.getValidator(), e.val());
        v.threadBound.set(true);
        bmap =
            bmap.assoc(v, new TBox(Thread.currentThread(), e.val()));
    }
    dvals.set(new Frame(bmap, f));
}

public static void popThreadBindings(){
    Frame f = dvals.get();
    if(f.prev == null)
        throw new IllegalStateException("Pop without matching push");
    dvals.set(f.prev);
}

public static Associative getThreadBindings(){
    Frame f = dvals.get();
    IPersistentMap ret = PersistentHashMap.EMPTY;
    for(ISeq bs = f.bindings.seq(); bs != null; bs = bs.next())
    {
        IMapEntry e = (IMapEntry) bs.first();
        Var v = (Var) e.key();
        TBox b = (TBox) e.val();
        ret = ret.assoc(v, b.val());
    }
    return ret;
}

public final TBox getThreadBinding(){
    if(threadBound.get())
    {
        IMapEntry e = dvals.get().bindings.entryAt(this);
        if(e != null)
            return (TBox) e.val();
    }
    return null;
}

```

```
final public IFn fn(){
    return (IFn) deref();
}

public Object call() throws Exception{
    return invoke();
}

public void run(){
    try
    {
        invoke();
    }
    catch(Exception e)
    {
        throw new RuntimeException(e);
    }
}

public Object invoke()
throws Exception{
    return fn().invoke();
}

public Object invoke(Object arg1)
throws Exception{
    return fn().invoke(arg1);
}

public Object invoke(Object arg1, Object arg2)
throws Exception{
    return fn().invoke(arg1, arg2);
}

public Object invoke(Object arg1, Object arg2, Object arg3)
throws Exception{
    return fn().invoke(arg1, arg2, arg3);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4)
throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
Object arg4, Object arg5)
throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4, arg5);
}
```

```
public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6)
throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7)
throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8)
throws Exception{
    return fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
                arg10);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
                arg10, arg11);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
```

```
        Object arg4, Object arg5, Object arg6,
        Object arg7, Object arg8, Object arg9,
        Object arg10, Object arg11, Object arg12)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16)
throws Exception{
    return
```

```
fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
           arg10, arg11, arg12, arg13, arg14, arg15,
           arg16);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17)
throws Exception{
    return
        fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
                   arg10, arg11, arg12, arg13, arg14, arg15,
                   arg16, arg17);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18)
throws Exception{
    return
        fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
                   arg10, arg11, arg12, arg13, arg14, arg15,
                   arg16, arg17, arg18);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19)
throws Exception{
    return
        fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
                   arg10, arg11, arg12, arg13, arg14, arg15,
                   arg16, arg17, arg18, arg19);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
```

```

        Object arg16, Object arg17, Object arg18,
        Object arg19, Object arg20)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15,
               arg16, arg17, arg18, arg19, arg20);
}

public Object invoke(Object arg1, Object arg2, Object arg3,
                     Object arg4, Object arg5, Object arg6,
                     Object arg7, Object arg8, Object arg9,
                     Object arg10, Object arg11, Object arg12,
                     Object arg13, Object arg14, Object arg15,
                     Object arg16, Object arg17, Object arg18,
                     Object arg19, Object arg20, Object... args)
throws Exception{
    return
    fn().invoke(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9,
               arg10, arg11, arg12, arg13, arg14, arg15,
               arg16, arg17, arg18, arg19, arg20, args);
}

public Object applyTo(ISeq arglist) throws Exception{
    return AFn.applyToHelper(this, arglist);
}

static IFn assoc = new AFn(){
    @Override
    public Object invoke(Object m, Object k, Object v) throws Exception {
        return RT.assoc(m, k, v);
    }
};
static IFn dissoc = new AFn() {
    @Override
    public Object invoke(Object c, Object k) throws Exception {
        return RT.dissoc(c, k);
    }
};
}

```

9.108 XMLHandler.java

(DefaultHandler [1723])
— XMLHandler.java —

```
/*
\getchunk{Clojure Copyright}
*/
/* rich Dec 17, 2007 */

package clojure.lang;

import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.Locator;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class XMLHandler extends DefaultHandler{
ContentHandler h;

public XMLHandler(ContentHandler h){
    this.h = h;
}

public void setDocumentLocator(Locator locator){
    h.setDocumentLocator(locator);
}

public void startDocument() throws SAXException{
    h.startDocument();
}

public void endDocument() throws SAXException{
    h.endDocument();
}

public void startPrefixMapping(String prefix, String uri)
throws SAXException{
    h.startPrefixMapping(prefix, uri);
}

public void endPrefixMapping(String prefix)
throws SAXException{
    h.endPrefixMapping(prefix);
}

public void startElement(String uri,
                        String localName,
                        String qName,
                        Attributes atts)
throws SAXException{
    h.startElement(uri, localName, qName, atts);
}
```

```
public void endElement(String uri, String localName, String qName)
    throws SAXException{
    h.endElement(uri, localName, qName);
}

public void characters(char ch[], int start, int length)
    throws SAXException{
    h.characters(ch, start, length);
}

public void ignorableWhitespace(char ch[], int start, int length)
    throws SAXException{
    h.ignorableWhitespace(ch, start, length);
}

public void processingInstruction(String target, String data)
    throws SAXException{
    h.processingInstruction(target, data);
}

public void skippedEntity(String name) throws SAXException{
    h.skippedEntity(name);
}

/*
public static void main(String[] args){
    try
    {
        ContentHandler dummy = new DefaultHandler();
        SAXParserFactory f = SAXParserFactory.newInstance();
        //f.setNamespaceAware(true);
        SAXParser p = f.newSAXParser();
        p.parse("http://arstechnica.com/journals.rssx",
            new XMLHandler(dummy));
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
/**/
}
```

Chapter 10

jvm/clojure

10.1 main.java

— main.java —

```
/*
\getchunk{Clojure Copyright}
*/
package clojure;

import clojure.lang.Symbol;
import clojure.lang.Var;
import clojure.lang.RT;

public class main{

final static private Symbol CLOJURE_MAIN =
    Symbol.intern("clojure.main");
final static private Var REQUIRE =
    RT.var("clojure.core", "require");
final static private Var LEGACY_REPL =
    RT.var("clojure.main", "legacy-repl");
final static private Var LEGACY_SCRIPT =
    RT.var("clojure.main", "legacy-script");
final static private Var MAIN =
    RT.var("clojure.main", "main");

public static void legacy_repl(String[] args) throws Exception{
    REQUIRE.invoke(CLOJURE_MAIN);
    LEGACY_REPL.invoke(RT.seq(args));
}
```

```
public static void legacy_script(String[] args) throws Exception{
    REQUIRE.invoke(CLOJURE_MAIN);
    LEGACY_SCRIPT.invoke(RT.seq(args));
}

public static void main(String[] args) throws Exception{
    REQUIRE.invoke(CLOJURE_MAIN);
    MAIN.applyTo(RT.seq(args));
}
}
```

Chapter 11

clj/clojure/

11.1 core.clj

```
— core.clj —  
\getchunk{Clojure Copyright}  
  
(ns ^{:doc "The core Clojure language."  
      :author "Rich Hickey"}  
  clojure.core)  
  
(def unquote)  
(def unquote-splicing)  
  
(def  
  ^{:arglists '([& items])  
   :doc "Creates a new list containing the items."  
   :added "1.0"}  
  list (. clojure.lang.PersistentList creator))  
  
(def  
  ^{:arglists '([x seq])  
   :doc "Returns a new seq where x is the first element and seq is  
   the rest."  
   :added "1.0"  
   :static true}  
  cons (fn* ^:static cons [x seq] (. clojure.lang.RT (cons x seq))))  
  
;during bootstrap we don't have destructuring let, loop or fn,  
;will redefine later  
(def
```

```

^{:macro true
  :added "1.0"}
let (fn* let [&form &env & decl] (cons 'let* decl)))

(def
^{:macro true
  :added "1.0"}
loop (fn* loop [&form &env & decl] (cons 'loop* decl)))

(def
^{:macro true
  :added "1.0"}
fn (fn* fn [&form &env & decl]
      (.withMeta ^clojure.lang.IObj (cons 'fn* decl)
                 (.meta ^clojure.lang.IMeta &form)))))

(def
^{:arglists '([coll])
  :doc "Returns the first item in the collection. Calls seq on its
        argument. If coll is nil, returns nil."
  :added "1.0"
  :static true}
first (fn ^:static first [coll] (. clojure.lang.RT (first coll)))))

(def
^{:arglists '([coll])
  :tag clojure.lang.ISeq
  :doc "Returns a seq of the items after the first. Calls seq on its
        argument. If there are no more items, returns nil."
  :added "1.0"
  :static true}
next (fn ^:static next [x] (. clojure.lang.RT (next x)))))

(def
^{:arglists '([coll])
  :tag clojure.lang.ISeq
  :doc "Returns a possibly empty seq of the items after the first.
        Calls seq on its argument."
  :added "1.0"
  :static true}
rest (fn ^:static rest [x] (. clojure.lang.RT (more x)))))

(def
^{:arglists '([coll x] [coll x & xs])
  :doc "conj[oin]. Returns a new collection with the xs
        'added'. (conj nil item) returns (item). The 'addition' may
        happen at different 'places' depending on the concrete type."
  :added "1.0"
  :static true}
conj (fn ^:static conj

```

```

([coll x] (. clojure.lang.RT (conj coll x)))
([coll x & xs]
 (if xs
     (recur (conj coll x) (first xs) (next xs))
     (conj coll x)))))

(def
^{:doc "Same as (first (next x))"
:arglists '([x])
:added "1.0"
:static true}
second (fn ^:static second [x] (first (next x)))))

(def
^{:doc "Same as (first (first x))"
:arglists '([x])
:added "1.0"
:static true}
ffirst (fn ^:static ffirst [x] (first (first x)))))

(def
^{:doc "Same as (next (first x))"
:arglists '([x])
:added "1.0"
:static true}
nfirst (fn ^:static nfirst [x] (next (first x)))))

(def
^{:doc "Same as (first (next x))"
:arglists '([x])
:added "1.0"
:static true}
fnext (fn ^:static fnext [x] (first (next x)))))

(def
^{:doc "Same as (next (next x))"
:arglists '([x])
:added "1.0"
:static true}
nnext (fn ^:static nnext [x] (next (next x)))))

(def
^{:arglists '(^clojure.lang.ISeq [coll])
:doc "Returns a seq on the collection. If the collection is
empty, returns nil. (seq nil) returns nil. seq also works on
Strings, native Java arrays (of reference types) and any objects
that implement Iterable."
:tag clojure.lang.ISeq
:added "1.0"
:static true}

```

```

seq (fn ^:static seq ^clojure.lang.ISeq [coll]
      (. clojure.lang.RT (seq coll)))

(def
  {:arglists '([`Class c x])
   :doc "Evaluates x and tests if it is an instance of the class
         c. Returns true or false"
   :added "1.0"}
  instance? (fn instance? [^Class c x] (. c (instance? x)))))

(def
  {:arglists '([x])
   :doc "Return true if x implements ISeq"
   :added "1.0"
   :static true}
  seq? (fn ^:static seq? [x] (instance? clojure.lang.ISeq x)))

(def
  {:arglists '([x])
   :doc "Return true if x is a Character"
   :added "1.0"
   :static true}
  char? (fn ^:static char? [x] (instance? Character x)))

(def
  {:arglists '([x])
   :doc "Return true if x is a String"
   :added "1.0"
   :static true}
  string? (fn ^:static string? [x] (instance? String x)))

(def
  {:arglists '([x])
   :doc "Return true if x implements IPersistentMap"
   :added "1.0"
   :static true}
  map? (fn ^:static map? [x] (instance? clojure.lang.IPersistentMap x)))

(def
  {:arglists '([x])
   :doc "Return true if x implements IPersistentVector"
   :added "1.0"
   :static true}
  vector? (fn ^:static vector? [x]
             (instance? clojure.lang.IPersistentVector x)))

(def
  {:arglists '([map key val] [map key val & kvs])
   :doc "assoc[iate]. When applied to a map, returns a new map of the
         same (hashed/sorted) type, that contains the mapping of key(s) to
         value(s). If multiple keys are present, the last one wins."})
  
```

```

val(s). When applied to a vector, returns a new vector that
contains val at index. Note - index must be <= (count vector)."
:added "1.0"
:static true}

assoc
(fn ^:static assoc
 ([map key val] (. clojure.lang.RT (assoc map key val)))
 ([map key val & kvs]
 (let [ret (assoc map key val)]
 (if kvs
 (recur ret (first kvs) (second kvs) (nnnext kvs))
 ret)))))

;;;;;;;;
(def
^{:arglists '([obj])
 :doc "Returns the metadata of obj, returns nil if there is no
       metadata."
:added "1.0"
:static true}
meta (fn ^:static meta [x]
 (if (instance? clojure.lang.IMeta x)
 (. ^clojure.lang.IMeta x (meta)))))

(def
^{:arglists '([^clojure.lang.IObj obj m])
 :doc "Returns an object of the same type and value as obj, with
       map m as its metadata."
:added "1.0"
:static true}
with-meta (fn ^:static with-meta [^clojure.lang.IObj x m]
 (. x (withMeta m)))))

(def ^{:private true :dynamic true}
 assert-valid-fdecl (fn [fdecl]))

(def
^{:private true}
sigs
(fn [fdecl]
 (assert-valid-fdecl fdecl)
 (let [asig
 (fn [fdecl]
 (let [arglist (first fdecl)
 ;elide implicit macro args
 arglist
 (if
 (clojure.lang.Util>equals '&form (first arglist))
 (clojure.lang.RT/subvec arglist 2
 (clojure.lang.RT/count arglist))

```

```

                        arglist)
body (next fdecl)]
(if (map? (first body))
(if (next body)
(with-meta arglist
(conj
(if (meta arglist) (meta arglist) {})
(first body)))
arglist)
arglist)))]
(if (seq? (first fdecl))
(loop [ret [] fdecls fdecl]
(if fdecls
(recur (conj ret (asig (first fdecls))) (next fdecls))
(seq ret)))
(list (asig fdecl)))))

(def
{:arglists '([coll])
:doc "Return the last item in coll, in linear time"
:added "1.0"
:static true}
last (fn ^:static last [s]
(if (next s)
(recur (next s))
(first s)))))

(def
{:arglists '([coll])
:doc "Return a seq of all but the last item in coll, in linear time"
:added "1.0"
:static true}
butlast (fn ^:static butlast [s]
(loop [ret [] s s]
(if (next s)
(recur (conj ret (first s)) (next s))
(seq ret)))))

(def
{:doc "Same as (def name (fn [params*] exprs*)) or (def
name (fn ([params*] exprs*)+)) with any doc-string or attrs added
to the var metadata"
:arglists '([name doc-string? attr-map? [params*] body]
[name doc-string? attr-map?
([params*] body)+ attr-map?])
:added "1.0"}
defn (fn defn [&form &env name & fdecl]
(let [m (if (string? (first fdecl))

```

```

{:doc (first fdecl)}
{})
fdecl (if (string? (first fdecl))
         (next fdecl)
         fdecl)
m (if (map? (first fdecl))
      (conj m (first fdecl))
      m)
fdecl (if (map? (first fdecl))
         (next fdecl)
         fdecl)
fdecl (if (vector? (first fdecl))
         (list fdecl)
         fdecl)
m (if (map? (last fdecl))
      (conj m (last fdecl))
      m)
fdecl (if (map? (last fdecl))
         (butlast fdecl)
         fdecl)
m (conj {:arglists (list 'quote (sig fdecl))} m)
m (let [inline (:inline m)
        ifn (first inline)
        iname (second inline)]
    ;; same as:
    ;; (if (and (= 'fn ifn) (not (symbol? iname))) ...)
    (if
        (if (clojure.lang.Util/equiv 'fn ifn)
            (if (instance? clojure.lang.Symbol iname)
                false true))
        ;; inserts the same fn name to the inline fn
        ;; if it does not have one
        (assoc m :inline
               (cons ifn
                     (cons (clojure.lang.Symbol/intern
                               (.concat
                                 (.getName ^clojure.lang.Symbol name)
                                 "__inliner"))
                           (next inline)))))

    m)
m (conj (if (meta name) (meta name) {}) m)
(list 'def (with-meta name m)
      ;;todo - restore propagation of fn name
      ;;must figure out how to convey primitive hints
      ;;to self calls first
      (cons 'fn fdecl) ))))

(. (var defn) (setMacro))

(defn cast

```

```

"Throws a ClassCastException if x is not a c, else returns x."
{:added "1.0"
 :static true}
[~Class c x]
(. c (cast x))

(defn to-array
  "Returns an array of Objects containing the contents of coll, which
  can be any Collection. Maps to java.util.Collection.toArray()."
{:tag "[Ljava.lang.Object;"
 :added "1.0"
 :static true}
[coll] (. clojure.lang.RT (toArray coll)))

(defn vector
  "Creates a new vector containing the args."
{:added "1.0"
 :static true}
([] [])
([a] [a])
([a b] [a b])
([a b c] [a b c])
([a b c d] [a b c d])
([a b c d & args]
  (. clojure.lang.LazilyPersistentVector
    (create (cons a (cons b (cons c (cons d args)))))))

(defn vec
  "Creates a new vector containing the contents of coll."
{:added "1.0"
 :static true}
([coll]
 (if (instance? java.util.Collection coll)
   (clojure.lang.LazilyPersistentVector/create coll)
   (. clojure.lang.LazilyPersistentVector
     (createOwning (to-array coll)))))

(defn hash-map
  "keyval => key val
  Returns a new hash map with supplied mappings."
{:added "1.0"
 :static true}
([] {})
([& keyvals]
  (. clojure.lang.PersistentHashMap (createWithCheck keyvals)))

(defn hash-set
  "Returns a new hash set with supplied keys."
{:added "1.0"
 :static true}

```

```
([] #{})
([& keys]
 (clojure.lang.PersistentHashSet/createWithCheck keys)))

(defn sorted-map
  "keyval => key val
  Returns a new sorted map with supplied mappings."
  {:added "1.0"
   :static true}
  [& keyvals]
  (clojure.lang.PersistentTreeMap/create keyvals))

(defn sorted-map-by
  "keyval => key val
  Returns a new sorted map with supplied mappings,
  using the supplied comparator."
  {:added "1.0"
   :static true}
  ([comparator & keyvals]
   (clojure.lang.PersistentTreeMap/create comparator keyvals)))

(defn sorted-set
  "Returns a new sorted set with supplied keys."
  {:added "1.0"
   :static true}
  [& keys]
  (clojure.lang.PersistentTreeSet/create keys))

(defn sorted-set-by
  "Returns a new sorted set with supplied keys,
  using the supplied comparator."
  {:added "1.1"
   :static true}
  ([comparator & keys]
   (clojure.lang.PersistentTreeSet/create comparator keys)))

;;;;;;
(defn nil?
  "Returns true if x is nil, false otherwise."
  {:tag Boolean
   :added "1.0"
   :static true
   :inline (fn [x] (list 'clojure.lang.Util/identical x nil))}
  [x] (clojure.lang.Util/identical x nil))

(def
  ^{:doc "Like defn, but the resulting function name is declared as a
macro and will be used as a macro by the compiler when it is"}
```

```

called."
:arglists
'([name doc-string? attr-map? [params*] body]
  [name doc-string? attr-map? ([params*] body)+ attr-map?])
:added "1.0"
defmacro (fn [&form &env
             name & args]
          (let [prefix (loop [p (list name) args args]
                        (let [f (first args)]
                          (if (string? f)
                              (recur (cons f p) (next args))
                            (if (map? f)
                                (recur (cons f p) (next args))
                                p))))
                fdecl (loop [fd args]
                           (if (string? (first fd))
                               (recur (next fd))
                             (if (map? (first fd))
                                 (recur (next fd))
                                 fd)))
                fdecl (if (vector? (first fdecl))
                           (list fdecl)
                         fdecl)
                add-implicit-args (fn [fd]
                           (let [args (first fd)]
                             (cons
                               (vec (cons '&form (cons '&env args)))
                               (next fd))))
                add-args (fn [acc ds]
                           (if (nil? ds)
                               acc
                               (let [d (first ds)]
                                 (if (map? d)
                                     (conj acc d)
                                     (recur
                                       (conj acc (add-implicit-args d))
                                       (next ds)))))))
                fdecl (seq (add-args [] fdecl))
                decl (loop [p prefix d fdecl]
                           (if p
                               (recur (next p) (cons (first p) d))
                             d))]
          (list 'do
                (cons 'defn decl)
                (list '. (list 'var name) '(setMacro))
                (list 'var name)))))

(. (var defmacro) (setMacro))

```

```
(defmacro when
  "Evaluates test. If logical true, evaluates body in an implicit do."
  {:added "1.0"}
  [test & body]
  (list 'if test (cons 'do body)))

(defmacro when-not
  "Evaluates test. If logical false, evaluates body in an implicit do."
  {:added "1.0"}
  [test & body]
  (list 'if test nil (cons 'do body)))

(defn false?
  "Returns true if x is the value false, false otherwise."
  {:tag Boolean,
   :added "1.0"
   :static true}
  [x] (clojure.lang.Util/identical x false))

(defn true?
  "Returns true if x is the value true, false otherwise."
  {:tag Boolean,
   :added "1.0"
   :static true}
  [x] (clojure.lang.Util/identical x true))

(defn not
  "Returns true if x is logical false, false otherwise."
  {:tag Boolean
   :added "1.0"
   :static true}
  [x] (if x false true))

(defn str
  "With no args, returns the empty string. With one arg x, returns
  x.toString(). (str nil) returns the empty string. With more than
  one arg, returns the concatenation of the str values of the args."
  {:tag String
   :added "1.0"
   :static true}
  (^String [] "")
  (^String [^Object x]
   (if (nil? x) "" (. x (toString))))
  (^String [x & ys]
   ((fn [^StringBuilder sb more]
      (if more
          (recur (. sb (append (str (first more)))) (next more))
          (str sb)))
      (new StringBuilder (str x)) ys))))
```

```
(defn symbol?
  "Return true if x is a Symbol"
  {:added "1.0"
   :static true}
  [x] (instance? clojure.lang.Symbol x))

(defn keyword?
  "Return true if x is a Keyword"
  {:added "1.0"
   :static true}
  [x] (instance? clojure.lang.Keyword x))

(defn symbol
  "Returns a Symbol with the given namespace and name."
  {:tag clojure.lang.Symbol
   :added "1.0"
   :static true}
  ([name] (if (symbol? name) name (clojure.lang.Symbol/intern name)))
  ([ns name] (clojure.lang.Symbol/intern ns name)))

(defn gensym
  "Returns a new symbol with a unique name. If a prefix string is
  supplied, the name is prefix# where # is some unique number. If
  prefix is not supplied, the prefix is 'G__'."'
  {:added "1.0"
   :static true}
  ([] (gensym "G__"))
  ([prefix-string]
   (. clojure.lang.Symbol
      (intern
       (str prefix-string (str (. clojure.lang.RT (nextID))))))))
  )

(defmacro cond
  "Takes a set of test/expr pairs. It evaluates each test one at a
  time. If a test returns logical true, cond evaluates and returns
  the value of the corresponding expr and doesn't evaluate any of the
  other tests or exprs. (cond) returns nil."
  {:added "1.0"}
  [& clauses]
  (when clauses
    (list 'if (first clauses)
          (if (next clauses)
              (second clauses)
              (throw (IllegalArgumentException.
                      "cond requires an even number of forms"))))
    (cons 'clojure.core/cond (next (next clauses)))))

(defn keyword
  "Returns a Keyword with the given namespace and name. Do not use :
```

```

in the keyword strings, it will be added automatically."
{:tag clojure.lang.Keyword
 :added "1.0"
 :static true}
([name] (cond (keyword? name) name
               (symbol? name)
               (clojure.lang.Keyword/intern ^clojure.lang.Symbol name)
               (string? name)
               (clojure.lang.Keyword/intern ^String name)))
 ([ns name] (clojure.lang.Keyword/intern ns name)))

(defn find-keyword
  "Returns a Keyword with the given namespace and name if one already
  exists. This function will not intern a new keyword. If the keyword
  has not already been interned, it will return nil. Do not use :
  in the keyword strings, it will be added automatically."
{:tag clojure.lang.Keyword
 :added "1.3"
 :static true}
([name] (cond (keyword? name) name
               (symbol? name)
               (clojure.lang.Keyword/find ^clojure.lang.Symbol name)
               (string? name)
               (clojure.lang.Keyword/find ^String name)))
 ([ns name] (clojure.lang.Keyword/find ns name)))

(defn spread
  {:private true
   :static true}
  [arglist]
  (cond
    (nil? arglist) nil
    (nil? (next arglist)) (seq (first arglist))
    :else (cons (first arglist) (spread (next arglist)))))

(defn list*
  "Creates a new list containing the items prepended to the rest, the
  last of which will be treated as a sequence."
  {:added "1.0"
   :static true}
  ([args] (seq args))
  ([a args] (cons a args))
  ([a b args] (cons a (cons b args)))
  ([a b c args] (cons a (cons b (cons c args))))
  ([a b c d & more]
   (cons a (cons b (cons c (cons d (spread more)))))))

(defn apply
  "Applies fn f to the argument list formed by prepending
  "

```

```

intervening arguments to args."
{:added "1.0"
:static true}
([`clojure.lang.IFn f args]
 (. f (applyTo (seq args))))
([`clojure.lang.IFn f x args]
 (. f (applyTo (list* x args))))
([`clojure.lang.IFn f x y args]
 (. f (applyTo (list* x y args))))
([`clojure.lang.IFn f x y z args]
 (. f (applyTo (list* x y z args))))
([`clojure.lang.IFn f a b c d & args]
 (. f (applyTo (cons a (cons b (cons c (cons d (spread args)))))))))

(defn vary-meta
"Returns an object of the same type and value as obj, with
(apply f (meta obj) args) as its metadata."
{:added "1.0"
:static true}
[obj f & args]
(with-meta obj (apply f (meta obj) args)))

(defmacro lazy-seq
"Takes a body of expressions that returns an ISeq or nil, and yields
a Seqable object that will invoke the body only the first time seq
is called, and will cache the result and return it on all subsequent
seq calls."
{:added "1.0"}
[& body]
(list 'new 'clojure.lang.LazySeq (list* `{:once true} fn* [] body)))

(defn ^:static `clojure.lang.ChunkBuffer chunk-buffer
`clojure.lang.ChunkBuffer [capacity]
(clojure.lang.ChunkBuffer. capacity))

(defn ^:static chunk-append [`clojure.lang.ChunkBuffer b x]
(.add b x))

(defn ^:static `clojure.lang.IChunk chunk [`clojure.lang.ChunkBuffer b]
(.chunk b))

(defn ^:static `clojure.lang.IChunk chunk-first
`clojure.lang.IChunk [`clojure.lang.IChunkedSeq s]
(.chunkedFirst s))

(defn ^:static `clojure.lang.ISeq chunk-rest
`clojure.lang.ISeq [`clojure.lang.IChunkedSeq s]
(.chunkedMore s))

(defn ^:static `clojure.lang.ISeq chunk-next

```

```

^clojure.lang.ISeq [^clojure.lang.IChunkedSeq s]
(.chunkedNext s))

(defn ^:static chunk-cons [chunk rest]
  (if (clojure.lang.Numbers/isZero (clojure.lang.RT/count chunk))
    rest
    (clojure.lang.ChunkedCons. chunk rest)))

(defn ^:static chunked-seq? [s]
  (instance? clojure.lang.IChunkedSeq s))

(defn concat
  "Returns a lazy seq representing the concatenation of the
  elements in the supplied colls."
{:added "1.0"
 :static true}
([] (lazy-seq nil))
([x] (lazy-seq x))
([x y]
 (lazy-seq
  (let [s (seq x)]
    (if s
      (if (chunked-seq? s)
        (chunk-cons (chunk-first s) (concat (chunk-rest s) y))
        (cons (first s) (concat (rest s) y)))
      y)))
 ([x y & zs]
  (let [cat (fn cat [xys zs]
    (lazy-seq
     (let [xys (seq xys)]
       (if xys
         (if (chunked-seq? xys)
           (chunk-cons (chunk-first xys)
                      (cat (chunk-rest xys) zs))
           (cons (first xys) (cat (rest xys) zs)))
         (when zs
           (cat (first zs) (next zs)))))))
    (cat (concat x y) zs)))))

;;;at this point all the support for syntax-quote exists;;
(defmacro delay
  "Takes a body of expressions and yields a Delay object that will
  invoke the body only the first time it is forced (with force or
  deref/@), and will cache the result and return it on all subsequent
  force calls."
{:added "1.0"}
[& body]
  (list 'new 'clojure.lang.Delay (list* `{:once true} fn* [] body)))

(defn delay?

```

```

"returns true if x is a Delay created with delay"
{:added "1.0"
 :static true}
[x] (instance? clojure.lang.Delay x))

(defn force
  "If x is a Delay, returns the (possibly cached) value of its
   expression, else returns x"
{:added "1.0"
 :static true}
[x] (. clojure.lang.Delay (force x)))

(defmacro if-not
  "Evaluates test. If logical false, evaluates and returns then expr,
   otherwise else expr, if supplied, else nil."
{:added "1.0"}
([test then] `(if-not ~test ~then nil))
([test then else]
 `'(if (not ~test) ~then ~else)))

(defn identical?
  "Tests if 2 arguments are the same object"
{:inline (fn [x y] `(. clojure.lang.Util identical ~x ~y)}
 :inline-arities #{2}
 :added "1.0"}
([x y] (clojure.lang.Util/identical x y)))

;equiv-based
(defn =
  "Equality. Returns true if x equals y, false if not. Same as
   Java x.equals(y) except it also works for nil, and compares
   numbers and collections in a type-independent manner. Clojure's
   immutable data structures define equals() (and thus =) as a value,
   not an identity, comparison."
{:inline (fn [x y] `(. clojure.lang.Util equiv ~x ~y)}
 :inline-arities #{2}
 :added "1.0"}
([x] true)
([x y] (clojure.lang.Util/equiv x y))
([x y & more]
 (if (= x y)
     (if (next more)
         (recur y (first more) (next more))
         (= y (first more)))
     false)))

;equals-based
#_(defn =
  "Equality. Returns true if x equals y, false if not. Same as Java
   x.equals(y) except it also works for nil. Boxed numbers must have
   "

```

```

same type. Clojure's immutable data structures define equals() (and
thus =) as a value, not an identity, comparison."
{:inline (fn [x y] `(. clojure.lang.Util equals ~x ~y))
 :inline-arities #{2}
 :added "1.0"}
([x] true)
([x y] (clojure.lang.Util>equals x y))
([x y & more]
 (if (= x y)
 (if (next more)
 (recur y (first more) (next more))
 (= y (first more)))
 false)))

(defn not=
  "Same as (not (= obj1 obj2))"
  {:tag Boolean
   :added "1.0"
   :static true}
  ([x] false)
  ([x y] (not (= x y)))
  ([x y & more]
   (not (apply = x y more)))))

(defn compare
  "Comparator. Returns a negative number, zero, or a positive number
when x is logically 'less than', 'equal to', or 'greater than'
y. Same as Java x.compareTo(y) except it also works for nil, and
compares numbers and collections in a type-independent manner. x
must implement Comparable"
{
  :inline (fn [x y] `(. clojure.lang.Util compare ~x ~y))
  :added "1.0"}
  [x y] (. clojure.lang.Util (compare x y)))

(defmacro and
  "Evaluates exprs one at a time, from left to right. If a form
returns logical false (nil or false), and returns that value and
doesn't evaluate any of the other expressions, otherwise it returns
the value of the last expr. (and) returns true."
  {:added "1.0"}
  ([] true)
  ([x] x)
  ([x & next]
   `(let [and# ~x]
      (if and# (and ~@next) and#)))))

(defmacro or

```

```

"Evaluates exprs one at a time, from left to right. If a form
returns a logical true value, or returns that value and doesn't
evaluate any of the other expressions, otherwise it returns the
value of the last expression. (or) returns nil."
{:added "1.0"}
([] nil)
([x] x)
([x & next]
  '(let [or# ~x]
    (if or# or# (or ~@next)))))

;;;;;;;;;; sequence fns ;;;;;;;
(defn zero?
  "Returns true if num is zero, else false"
  {
  :inline (fn [x] '(. clojure.lang.Numbers (isZero ~x)))
  :added "1.0"}
  [x] (. clojure.lang.Numbers (isZero x)))

(defn count
  "Returns the number of items in the collection. (count nil) returns
0. Also works on strings, arrays, and Java Collections and Maps"
  {
  :inline (fn [x] '(. clojure.lang.RT (count ~x)))
  :added "1.0"}
  [coll] (clojure.lang.RT/count coll))

(defn int
  "Coerce to int"
  {
  :inline (fn [x]
    '(. clojure.lang.RT
      (~(if *unchecked-math* 'uncheckedIntCast 'intCast) ~x)))
  :added "1.0"}
  [x] (. clojure.lang.RT (intCast x)))

(defn nth
  "Returns the value at the index. get returns nil if index out of
bounds, nth throws an exception unless not-found is supplied. nth
also works for strings, Java arrays, regex Matchers and Lists, and,
in O(n) time, for sequences."
  {:inline (fn [c i & nf] '(. clojure.lang.RT (nth ~c ~i ~@nf)))
  :inline-arithes #{2 3}
  :added "1.0"}
  ([coll index] (. clojure.lang.RT (nth coll index)))
  ([coll index not-found]
   (. clojure.lang.RT (nth coll index not-found)))))

(defn <
  "Returns non-nil if nums are in monotonically increasing order,

```

```

otherwise false."
{:inline (fn [x y] `(. clojure.lang.Numbers (lt ~x ~y)))
 :inline-arities #{2}
 :added "1.0"}
([x] true)
([x y] (. clojure.lang.Numbers (lt x y)))
([x y & more]
 (if (< x y)
 (if (next more)
 (recur y (first more) (next more))
 (< y (first more)))
 false)))

(defn inc'
  "Returns a number one greater than num. Supports arbitrary precision.
  See also: inc"
  {:inline (fn [x] `(. clojure.lang.Numbers (incP ~x)))
   :added "1.0"}
  [x] (. clojure.lang.Numbers (incP x)))

(defn inc
  "Returns a number one greater than num. Does not auto-promote
  longs, will throw on overflow. See also: inc'"
  {:inline (fn [x]
    `(. clojure.lang.Numbers
        (~(if *unchecked-math* 'unchecked-inc 'inc) ~x)))
   :added "1.2"}
  [x] (. clojure.lang.Numbers (inc x)))

;; reduce is defined again later after InternalReduce loads
(defn ^:private ^:static
  reduce1
  ([f coll]
   (let [s (seq coll)]
     (if s
       (reduce1 f (first s) (next s))
       (f))))
  ([f val coll]
   (let [s (seq coll)]
     (if s
       (if (chunked-seq? s)
         (recur f
               (.reduce (chunk-first s) f val)
               (chunk-next s))
         (recur f (f val (first s)) (next s)))
       val)))))

(defn reverse
  "Returns a seq of the items in coll in reverse order. Not lazy."
  {:added "1.0"

```

```

:static true}
[coll]
(reduce1 conj () coll))

;;math stuff
(defn +
  "Returns the sum of nums. (+) returns 0. Supports arbitrary precision.
  See also: +'"
{:inline (fn [x y] `(. clojure.lang.Numbers (addP ~x ~y)))
:inline-arities #{2}
:added "1.0"}
([] 0)
([x] (cast Number x))
([x y] (. clojure.lang.Numbers (addP x y)))
([x y & more]
(reduce1 +' (+' x y) more)))

(defn +
  "Returns the sum of nums. (+) returns 0. Does not auto-promote
  longs, will throw on overflow. See also: +'"
{:inline (fn [x y]
  `(. clojure.lang.Numbers
      (~(if *unchecked-math* 'unchecked_add 'add) ~x ~y)))
:inline-arities #{2}
:added "1.2"}
([] 0)
([x] (cast Number x))
([x y] (. clojure.lang.Numbers (add x y)))
([x y & more]
(reduce1 + (+' x y) more)))

(defn *
  "Returns the product of nums. (*) returns 1. Supports arbitrary
  precision. See also: *"
{:inline (fn [x y] `(. clojure.lang.Numbers (multiplyP ~x ~y)))
:inline-arities #{2}
:added "1.0"}
([] 1)
([x] (cast Number x))
([x y] (. clojure.lang.Numbers (multiplyP x y)))
([x y & more]
(reduce1 *' (*' x y) more)))

(defn *
  "Returns the product of nums. (*) returns 1. Does not auto-promote
  longs, will throw on overflow. See also: *'"
{:inline
 (fn [x y]
  `(. clojure.lang.Numbers
      (~(if *unchecked-math* 'unchecked_multiply 'multiply) ~x ~y)))}
```

```

:inline-arities #{2}
:added "1.2"
([] 1)
([x] (cast Number x))
([x y] (. clojure.lang.Numbers (multiply x y)))
([x y & more]
 (reduce1 * (* x y) more)))

(defn /
  "If no denominators are supplied, returns 1/numerator,
  else returns numerator divided by all of the denominators."
  {:inline (fn [x y] `(. clojure.lang.Numbers (divide ~x ~y)))
   :inline-arities #{2}
   :added "1.0"}
  ([x] (/ 1 x))
  ([x y] (. clojure.lang.Numbers (divide x y)))
  ([x y & more]
   (reduce1 / (/ x y) more)))

(defn -
  "If no ys are supplied, returns the negation of x, else subtracts
  the ys from x and returns the result. Supports arbitrary precision.
  See also: -"
  {:inline (fn [& args] `(. clojure.lang.Numbers (minusP ~@args)))
   :inline-arities #{1 2}
   :added "1.0"}
  ([x] (. clojure.lang.Numbers (minusP x)))
  ([x y] (. clojure.lang.Numbers (minusP x y)))
  ([x y & more]
   (reduce1 -' (-' x y) more)))

(defn -
  "If no ys are supplied, returns the negation of x, else subtracts
  the ys from x and returns the result. Does not auto-promote
  longs, will throw on overflow. See also: -'"
  {:inline
   (fn [& args]
     `(. clojure.lang.Numbers
         (^if *unchecked-math* 'unchecked_minus 'minus) ~@args)))
   :inline-arities #{1 2}
   :added "1.2"}
  ([x] (. clojure.lang.Numbers (minus x)))
  ([x y] (. clojure.lang.Numbers (minus x y)))
  ([x y & more]
   (reduce1 - (- x y) more)))

(defn <=
  "Returns non-nil if nums are in monotonically non-decreasing order,
  otherwise false."
  {:inline (fn [x y] `(. clojure.lang.Numbers (lte ~x ~y)))}

```

```

:inline-arities #{2}
:added "1.0"
([x] true)
([x y] (. clojure.lang.Numbers (lte x y)))
([x y & more]
(if (<= x y)
(if (next more)
(recur y (first more) (next more))
(<= y (first more)))
false)))

(defn >
"Returns non-nil if nums are in monotonically decreasing order,
otherwise false."
{:inline (fn [x y] `(. clojure.lang.Numbers (gt ~x ~y)))
:inline-arities #{2}
:added "1.0"}
([x] true)
([x y] (. clojure.lang.Numbers (gt x y)))
([x y & more]
(if (> x y)
(if (next more)
(recur y (first more) (next more))
(> y (first more)))
false)))

(defn >=
"Returns non-nil if nums are in monotonically non-increasing order,
otherwise false."
{:inline (fn [x y] `(. clojure.lang.Numbers (gte ~x ~y)))
:inline-arities #{2}
:added "1.0"}
([x] true)
([x y] (. clojure.lang.Numbers (gte x y)))
([x y & more]
(if (>= x y)
(if (next more)
(recur y (first more) (next more))
(>= y (first more)))
false)))

(defn ==
"Returns non-nil if nums all have the equivalent
value (type-independent), otherwise false"
{:inline (fn [x y] `(. clojure.lang.Numbers (equiv ~x ~y)))
:inline-arities #{2}
:added "1.0"}
([x] true)
([x y] (. clojure.lang.Numbers (equiv x y)))
([x y & more]

```

```

(if (== x y)
  (if (next more)
    (recur y (first more) (next more))
    (== y (first more)))
  false)))

(defn max
  "Returns the greatest of the nums."
  {:added "1.0"
   :static true}
  ([x] x)
  ([x y] (if (> x y) x y))
  ([x y & more]
   (reduce1 max (max x y) more)))

(defn min
  "Returns the least of the nums."
  {:added "1.0"
   :static true}
  ([x] x)
  ([x y] (if (< x y) x y))
  ([x y & more]
   (reduce1 min (min x y) more)))

(defn dec'
  "Returns a number one less than num. Supports arbitrary precision.
  See also: dec"
  {:inline (fn [x] `(. clojure.lang.Numbers (decP ~x)))
   :added "1.0"}
  [x] (. clojure.lang.Numbers (decP x)))

(defn dec
  "Returns a number one less than num. Does not auto-promote
  longs, will throw on overflow. See also: dec'"
  {:inline
   (fn [x]
     `(. clojure.lang.Numbers
         (~(if *unchecked-math* 'unchecked-dec 'dec) ~x)))
   :added "1.2"}
  [x] (. clojure.lang.Numbers (dec x)))

(defn unchecked-inc-int
  "Returns a number one greater than x, an int.
  Note - uses a primitive operator subject to overflow."
  {:inline (fn [x] `(. clojure.lang.Numbers (unchecked-int-inc ~x)))
   :added "1.0"}
  [x] (. clojure.lang.Numbers (unchecked-int-inc x)))

(defn unchecked-inc
  "Returns a number one greater than x, a long.

```

```

Note - uses a primitive operator subject to overflow."
{:inline (fn [x] '(. clojure.lang.Numbers (unchecked_inc ~x)))
 :added "1.0"}
[x] (. clojure.lang.Numbers (unchecked_inc x)))

(defn unchecked-dec-int
  "Returns a number one less than x, an int.
  Note - uses a primitive operator subject to overflow."
  {:inline (fn [x] '(. clojure.lang.Numbers (unchecked_int_dec ~x)))
   :added "1.0"}
  [x] (. clojure.lang.Numbers (unchecked_int_dec x)))

(defn unchecked-dec
  "Returns a number one less than x, a long.
  Note - uses a primitive operator subject to overflow."
  {:inline (fn [x] '(. clojure.lang.Numbers (unchecked_dec ~x)))
   :added "1.0"}
  [x] (. clojure.lang.Numbers (unchecked_dec x)))

(defn unchecked-negate-int
  "Returns the negation of x, an int.
  Note - uses a primitive operator subject to overflow."
  {:inline (fn [x] '(. clojure.lang.Numbers (unchecked_int_negate ~x)))
   :added "1.0"}
  [x] (. clojure.lang.Numbers (unchecked_int_negate x)))

(defn unchecked-negate
  "Returns the negation of x, a long.
  Note - uses a primitive operator subject to overflow."
  {:inline (fn [x] '(. clojure.lang.Numbers (unchecked_minus ~x)))
   :added "1.0"}
  [x] (. clojure.lang.Numbers (unchecked_minus x)))

(defn unchecked-add-int
  "Returns the sum of x and y, both int.
  Note - uses a primitive operator subject to overflow."
  {:inline (fn [x y] '(. clojure.lang.Numbers (unchecked_int_add ~x ~y)))
   :added "1.0"}
  [x y] (. clojure.lang.Numbers (unchecked_int_add x y)))

(defn unchecked-add
  "Returns the sum of x and y, both long.
  Note - uses a primitive operator subject to overflow."
  {:inline (fn [x y] '(. clojure.lang.Numbers (unchecked_add ~x ~y)))
   :added "1.0"}
  [x y] (. clojure.lang.Numbers (unchecked_add x y)))

(defn unchecked-subtract-int
  "Returns the difference of x and y, both int.
  Note - uses a primitive operator subject to overflow."

```

```

{:inline
  (fn [x y] '(. clojure.lang.Numbers (unchecked_int_subtract ~x ~y)))
  :added "1.0"}
[x y] (. clojure.lang.Numbers (unchecked_int_subtract x y)))

(defn unchecked-subtract
  "Returns the difference of x and y, both long.
  Note - uses a primitive operator subject to overflow."
  {:inline (fn [x y] '(. clojure.lang.Numbers (unchecked_minus ~x ~y)))
   :added "1.0"}
  [x y] (. clojure.lang.Numbers (unchecked_minus x y)))

(defn unchecked-multiply-int
  "Returns the product of x and y, both int.
  Note - uses a primitive operator subject to overflow."
  {:inline
   (fn [x y] '(. clojure.lang.Numbers (unchecked_int_multiply ~x ~y)))
   :added "1.0"}
  [x y] (. clojure.lang.Numbers (unchecked_int_multiply x y)))

(defn unchecked-multiply
  "Returns the product of x and y, both long.
  Note - uses a primitive operator subject to overflow."
  {:inline
   (fn [x y] '(. clojure.lang.Numbers (unchecked_multiply ~x ~y)))
   :added "1.0"}
  [x y] (. clojure.lang.Numbers (unchecked_multiply x y)))

(defn unchecked-divide-int
  "Returns the division of x by y, both int.
  Note - uses a primitive operator subject to truncation."
  {:inline
   (fn [x y] '(. clojure.lang.Numbers (unchecked_int_divide ~x ~y)))
   :added "1.0"}
  [x y] (. clojure.lang.Numbers (unchecked_int_divide x y)))

(defn unchecked-remainder-int
  "Returns the remainder of division of x by y, both int.
  Note - uses a primitive operator subject to truncation."
  {:inline
   (fn [x y] '(. clojure.lang.Numbers (unchecked_int_remainder ~x ~y)))
   :added "1.0"}
  [x y] (. clojure.lang.Numbers (unchecked_int_remainder x y)))

(defn pos?
  "Returns true if num is greater than zero, else false"
  {
  :inline (fn [x] '(. clojure.lang.Numbers (isPos ~x)))
  :added "1.0"}
  [x] (. clojure.lang.Numbers (isPos x)))

```

```
(defn neg?
  "Returns true if num is less than zero, else false"
  {
    :inline (fn [x] '(. clojure.lang.Numbers (isNeg ~x)))
    :added "1.0"
    [x] (. clojure.lang.Numbers (isNeg x)))

(defn quot
  "quotient of dividing numerator by denominator."
  {:added "1.0"
   :static true
   :inline (fn [x y] '(. clojure.lang.Numbers (quotient ~x ~y)))}
  [num div]
  (. clojure.lang.Numbers (quotient num div)))

(defn rem
  "remainder of dividing numerator by denominator."
  {:added "1.0"
   :static true
   :inline (fn [x y] '(. clojure.lang.Numbers (remainder ~x ~y)))}
  [num div]
  (. clojure.lang.Numbers (remainder num div)))

(defn rationalize
  "returns the rational value of num"
  {:added "1.0"
   :static true}
  [num]
  (. clojure.lang.Numbers (rationalize num)))

;;Bit ops

(defn bit-not
  "Bitwise complement"
  {:inline (fn [x] '(. clojure.lang.Numbers (not ~x)))
   :added "1.0"}
  [x] (. clojure.lang.Numbers not x))

(defn bit-and
  "Bitwise and"
  {:inline (fn [x y] '(. clojure.lang.Numbers (and ~x ~y)))
   :added "1.0"}
  [x y] (. clojure.lang.Numbers and x y))

(defn bit-or
  "Bitwise or"
  {:inline (fn [x y] '(. clojure.lang.Numbers (or ~x ~y)))
   :added "1.0"}
```

```
[x y] (. clojure.lang.Numbers or x y))

(defn bit-xor
  "Bitwise exclusive or"
  {:inline (fn [x y] `(. clojure.lang.Numbers (xor ~x ~y)))
   :added "1.0"}
  [x y] (. clojure.lang.Numbers xor x y))

(defn bit-and-not
  "Bitwise and with complement"
  {:added "1.0"
   :static true}
  [x y] (. clojure.lang.Numbers andNot x y))

(defn bit-clear
  "Clear bit at index n"
  {:added "1.0"
   :static true}
  [x n] (. clojure.lang.Numbers clearBit x n))

(defn bit-set
  "Set bit at index n"
  {:added "1.0"
   :static true}
  [x n] (. clojure.lang.Numbers setBit x n))

(defn bit-flip
  "Flip bit at index n"
  {:added "1.0"
   :static true}
  [x n] (. clojure.lang.Numbers flipBit x n))

(defn bit-test
  "Test bit at index n"
  {:added "1.0"
   :static true}
  [x n] (. clojure.lang.Numbers testBit x n))

(defn bit-shift-left
  "Bitwise shift left"
  {:inline (fn [x n] `(. clojure.lang.Numbers (shiftLeft ~x ~n)))
   :added "1.0"}
  [x n] (. clojure.lang.Numbers shiftLeft x n))

(defn bit-shift-right
  "Bitwise shift right"
  {:inline (fn [x n] `(. clojure.lang.Numbers (shiftRight ~x ~n)))
   :added "1.0"}
```

```
[x n] (. clojure.lang.Numbers shiftRight x n))

(defn even?
  "Returns true if n is even, throws an exception if n is not an integer"
  {:added "1.0"
   :static true}
  [n] (zero? (bit-and n 1)))

(defn odd?
  "Returns true if n is odd, throws an exception if n is not an integer"
  {:added "1.0"
   :static true}
  [n] (not (even? n)))

;;

(defn complement
  "Takes a fn f and returns a fn that takes the same arguments as f,
  has the same effects, if any, and returns the opposite truth value."
  {:added "1.0"
   :static true}
  [f]
  (fn
    ([] (not (f)))
    ([x] (not (f x)))
    ([x y] (not (f x y)))
    ([x y & zs] (not (apply f x y zs)))))

(defn constantly
  "Returns a function that takes any number of arguments and returns x."
  {:added "1.0"
   :static true}
  [x] (fn [& args] x))

(defn identity
  "Returns its argument."
  {:added "1.0"
   :static true}
  [x] x)

;;Collection stuff

;;list stuff
(defn peek
  "For a list or queue, same as first, for a vector, same as, but much
  more efficient than, last. If the collection is empty, returns nil."
  {:added "1.0"
   :static true}
  [coll] (. clojure.lang.RT (peek coll)))
```

```
(defn pop
  "For a list or queue, returns a new list/queue without the first
  item, for a vector, returns a new vector without the last item. If
  the collection is empty, throws an exception. Note - not the same
  as next/butlast."
  {:added "1.0"
   :static true}
  [coll] (. clojure.lang.RT (pop coll)))

;;map stuff

(defn contains?
  "Returns true if key is present in the given collection, otherwise
  returns false. Note that for numerically indexed collections like
  vectors and Java arrays, this tests if the numeric key is within the
  range of indexes. 'contains?' operates constant or logarithmic time;
  it will not perform a linear search for a value. See also 'some'."
  {:added "1.0"
   :static true}
  [coll key] (. clojure.lang.RT (contains coll key)))

(defn get
  "Returns the value mapped to key, not-found or nil if key not present."
  {:inline (fn [m k & nf] '(. clojure.lang.RT (get ~m ~k ~@nf)))
   :inline-arities #{2 3}
   :added "1.0"}
  ([map key]
   (. clojure.lang.RT (get map key)))
  ([map key not-found]
   (. clojure.lang.RT (get map key not-found)))))

(defn dissoc
  "dissoc[iate]. Returns a new map of the same (hashed/sorted) type,
  that does not contain a mapping for key(s)."
  {:added "1.0"
   :static true}
  ([map] map)
  ([map key]
   (. clojure.lang.RT (dissoc map key)))
  ([map key & ks]
   (let [ret (dissoc map key)]
     (if ks
       (recur ret (first ks) (next ks))
       ret)))))

(defn disj
  "disj[oin]. Returns a new set of the same (hashed/sorted) type, that
  does not contain key(s)."
  {:added "1.0"
```

```

:static true}
([set] set)
([^clojure.lang.IPersistentSet set key]
(when set
  (. set (disjoin key))))
[set key & ks]
(when set
  (let [ret (disj set key)]
    (if ks
      (recur ret (first ks) (next ks))
      ret)))))

(defn find
  "Returns the map entry for key, or nil if key not present."
  {:added "1.0"
   :static true}
  [map key] (. clojure.lang.RT (find map key)))

(defn select-keys
  "Returns a map containing only those entries in map whose key
  is in keys"
  {:added "1.0"
   :static true}
  [map keyseq]
  (loop [ret {} keys (seq keyseq)]
    (if keys
        (let [entry (. clojure.lang.RT (find map (first keys)))]
          (recur
            (if entry
              (conj ret entry)
              ret)
            (next keys)))
        ret)))

(defn keys
  "Returns a sequence of the map's keys."
  {:added "1.0"
   :static true}
  [map] (. clojure.lang.RT (keys map)))

(defn vals
  "Returns a sequence of the map's values."
  {:added "1.0"
   :static true}
  [map] (. clojure.lang.RT (vals map)))

(defn key
  "Returns the key of the map entry."
  {:added "1.0"
   :static true}

```

```
[^java.util.Map$Entry e]
(. e (getKey))

(defn val
  "Returns the value in the map entry."
{:added "1.0"
 :static true}
[^java.util.Map$Entry e]
(. e (getValue))

(defn rseq
  "Returns, in constant time, a seq of the items in rev (which
  can be a vector or sorted-map), in reverse order. If rev is
  empty returns nil"
{:added "1.0"
 :static true}
[^clojure.lang.Reversible rev]
(. rev (rseq))

(defn name
  "Returns the name String of a string, symbol or keyword."
{:tag String
 :added "1.0"
 :static true}
[x]
(if (string? x) x (. ^clojure.lang.Named x (getName)))

(defn namespace
  "Returns the namespace String of a symbol or keyword, or nil
  if not present."
{:tag String
 :added "1.0"
 :static true}
[^clojure.lang.Named x]
(. x (getNamespace))

(defmacro locking
  "Executes exprs in an implicit do, while holding the monitor of x.
  Will release the monitor of x in all circumstances."
{:added "1.0"}
[x & body]
'(let [lockee# ~x]
  (try
    (monitor-enter lockee#)
    ~@body
    (finally
      (monitor-exit lockee#)))))

(defmacro ..
  "form => fieldName-symbol or (instanceMethodName-symbol args*)
```

Expands into a member access (.) of the first member on the first argument, followed by the next member on the result, etc. For instance:

```
(.. System (getProperties) (get \"os.name\"))

expands to:

(. (. System (getProperties)) (get \"os.name\"))

but is easier to write, read, and understand.

{:added "1.0"}
([x form] '(. ~x ~form))
([x form & more] '(.. (. ~x ~form) ~@more))

(defmacro ->
  "Threads the expr through the forms. Inserts x as the
  second item in the first form, making a list of it if it is not a
  list already. If there are more forms, inserts the first form as the
  second item in second form, etc."
{:added "1.0"}
([x] x)
([x form] (if (seq? form)
                (with-meta `(~(first form) ~x ~@(next form)) (meta form))
                (list form x)))
([x form & more] `(-> (-> ~x ~form) ~@more)))

(defmacro ->>
  "Threads the expr through the forms. Inserts x as the
  last item in the first form, making a list of it if it is not a
  list already. If there are more forms, inserts the first form as the
  last item in second form, etc."
{:added "1.1"}
([x form] (if (seq? form)
                (with-meta `(~(first form) ~@(next form) ~x) (meta form))
                (list form x)))
([x form & more] `(->> (->> ~x ~form) ~@more)))

;; multimethods
(def global-hierarchy

(defmacro defmulti
  "Creates a new multimethod with the associated dispatch function.
  The docstring and attribute-map are optional.

  Options are key-value pairs and may be one of:
    :default    the default dispatch value, defaults to :default
    :hierarchy  the isa? hierarchy to use for dispatching
                defaults to the global hierarchy"

```

```

{:arglists '([name docstring? attr-map? dispatch-fn & options])
  :added "1.0"}
[mm-name & options]
(let [docstring (if (string? (first options))
                     (first options)
                     nil)
      options (if (string? (first options))
                  (next options)
                  options)
      m (if (map? (first options))
            (first options)
            {}))
      options (if (map? (first options))
                  (next options)
                  options)
      dispatch-fn (first options)
      options (next options)
      m (if docstring
            (assoc m :doc docstring)
            m)
      m (if (meta mm-name)
            (conj (meta mm-name) m)
            m)]
  (when (= (count options) 1)
    (throw (Exception.
             "The syntax for defmulti has changed.
              Example: (defmulti name dispatch-fn :default dispatch-value)")))
  (let [options (apply hash-map options)
        default (get options :default :default)
        hierarchy (get options :hierarchy #'global-hierarchy)]
    '(let [v# (def ~mm-name)]
       (when-not (and (.hasRoot v#)
                      (instance? clojure.lang.MultiFn (deref v#)))
         (def ~(with-meta mm-name m)
               (new clojure.lang.MultiFn
                   ~(name mm-name) ~dispatch-fn ~default ~hierarchy)))))))

(defmacro defmethod
  "Creates and installs a new method of multimethod associated
   with dispatch-value. "
  {:added "1.0"}
  [multifn dispatch-val & fn-tail]
  `(~(with-meta multifn {:tag 'clojure.lang.MultiFn})
     addMethod ~dispatch-val (fn ~@fn-tail)))

(defn remove-all-methods
  "Removes all of the methods of multimethod."
  {:added "1.2"
   :static true}
  [^clojure.lang.MultiFn multifn]

```

```

(.reset multifn))

(defn remove-method
  "Removes the method of multimethod associated with dispatch-value."
  {:added "1.0"
   :static true}
  [^clojure.lang.MultiFn multifn dispatch-val]
  (.multifn removeMethod dispatch-val))

(defn prefer-method
  "Causes the multimethod to prefer matches of dispatch-val-x
  over dispatch-val-y
  when there is a conflict"
  {:added "1.0"
   :static true}
  [^clojure.lang.MultiFn multifn dispatch-val-x dispatch-val-y]
  (.multifn preferMethod dispatch-val-x dispatch-val-y))

(defn methods
  "Given a multimethod, returns a map of dispatch values -> dispatch fns"
  {:added "1.0"
   :static true}
  [^clojure.lang.MultiFn multifn] (.getMethodTable multifn))

(defn get-method
  "Given a multimethod and a dispatch value, returns the dispatch fn
  that would apply to that value, or nil if none apply and no default"
  {:added "1.0"
   :static true}
  [^clojure.lang.MultiFn multifn dispatch-val]
  (.getMethod multifn dispatch-val))

(defn prefers
  "Given a multimethod, returns a map of preferred value -> set
  of other values"
  {:added "1.0"
   :static true}
  [^clojure.lang.MultiFn multifn] (.getPreferTable multifn))

;;;;;; var stuff

(defmacro ^{:private true} assert-args [fnname & pairs]
  `(do (when-not ~(first pairs)
              (throw (IllegalArgumentException.
                      ~(str fnname " requires " (second pairs))))))
       ~(let [more (nnext pairs)]
          (when more
            (list* 'assert-args fnname more)))))

(defmacro if-let

```

```

"bindings => binding-form test

If test is true, evaluates then with binding-form bound to the
value of test, if not, yields else"
{:added "1.0"}
([bindings then]
 '(if-let ~bindings ~then nil))
([bindings then else & oldform]
 (assert-args if-let
 (and (vector? bindings) (nil? oldform)) "a vector for its binding"
 (= 2 (count bindings)) "exactly 2 forms in binding vector")
 (let [form (bindings 0) tst (bindings 1)]
 '(let [temp# ~tst]
 (if temp#
 (let [~form temp#]
 ~then)
 ~else)))))

(defmacro when-let
 "bindings => binding-form test

When test is true, evaluates body with binding-form bound to
the value of test"
{:added "1.0"}
[bindings & body]
(assert-args when-let
 (vector? bindings) "a vector for its binding"
 (= 2 (count bindings)) "exactly 2 forms in binding vector")
(let [form (bindings 0) tst (bindings 1)]
'(let [temp# ~tst]
 (when temp#
 (let [~form temp#]
 ~@body)))))

(defn push-thread-bindings
 "WARNING: This is a low-level function. Prefer high-level macros like
binding where ever possible.

Takes a map of Var/value pairs. Binds each Var to the associated
value for the current thread. Each call *MUST* be accompanied by
a matching call to pop-thread-bindings wrapped in a try-finally!

(push-thread-bindings bindings)
(try
 ...
 (finally
 (pop-thread-bindings)))"
{:added "1.1"
 :static true}
[bindings]
```

```
(clojure.lang.Var/pushThreadBindings bindings))

(defn pop-thread-bindings
  "Pop one set of bindings pushed with push-binding before. It is an
  error to pop bindings without pushing before."
  {:added "1.1"
   :static true}
 []
 (clojure.lang.Var/popThreadBindings))

(defn get-thread-bindings
  "Get a map with the Var/value pairs which is currently in effect
  for the current thread."
  {:added "1.1"
   :static true}
 []
 (clojure.lang.Var/getThreadBindings))

(defmacro binding
  "binding => var-symbol init-expr

Creates new bindings for the (already-existing) vars, with the
supplied initial values, executes the exprs in an implicit do, then
re-establishes the bindings that existed before. The new bindings
are made in parallel (unlike let); all init-exprs are evaluated
before the vars are bound to their new values."
  {:added "1.0"}
  [bindings & body]
  (assert-args binding
    (vector? bindings) "a vector for its binding"
    (even? (count bindings)) "an even number of forms in binding vector")
  (let [var-size (fn [var-vals]
                  (loop [ret [] vvs (seq var-vals)]
                    (if vvs
                        (recur
                          (conj
                            (conj ret `(var ~(first vvs)))
                            (second vvs))
                          (next (next vvs)))
                        (seq ret))))]
    `(~(let []
         (push-thread-bindings (hash-map ~@(var-size bindings)))
         (try
           ^@body
           (finally
             (pop-thread-bindings)))))))

(defn with-bindings*
  "Takes a map of Var/value pairs. Installs for the given Vars the
  associated values as thread-local bindings. Then calls f with the
```

```

supplied arguments. Pops the installed bindings after f returned.
Returns whatever f returns."
{:added "1.1"
 :static true}
[binding-map f & args]
(push-thread-bindings binding-map)
(try
  (apply f args)
  (finally
    (pop-thread-bindings)))))

(defmacro with-bindings
  "Takes a map of Var/value pairs. Installs for the given Vars the
  associated values as thread-local bindings. The executes body.
  Pops the installed bindings after body was evaluated. Returns the
  value of body."
{:added "1.1"}
[binding-map & body]
'(with-bindings* ~binding-map (fn [] ~@body)))

(defn bound-fn*
  "Returns a function, which will install the same bindings in effect
  as in the thread at the time bound-fn* was called and then call f
  with any given arguments. This may be used to define a helper
  function which runs on a different thread, but needs the same
  bindings in place."
{:added "1.1"
 :static true}
[f]
(let [bindings (get-thread-bindings)]
  (fn [& args]
    (apply with-bindings* bindings f args)))))

(defmacro bound-fn
  "Returns a function defined by the given fntail, which will install
  the same bindings in effect as in the thread at the time bound-fn
  was called. This may be used to define a helper function which
  runs on a different thread, but needs the same bindings in place."
{:added "1.1"}
[& fntail]
'(bound-fn* (fn ~@fntail)))

(defn find-var
  "Returns the global var named by the namespace-qualified symbol, or
  nil if no var with that name."
{:added "1.0"
 :static true}
[sym] (. clojure.lang.Var (find sym)))

(defn binding-conveyor-fn

```



```

:continue (the default if an error-handler is given) or :fail (the
default if no error-handler is given) -- see set-error-mode! for
details."
{:added "1.0"
 :static true
}
([state & options]
 (let [a (new clojure.lang.Agent state)
       opts (apply hash-map options)]
   (setup-reference a options)
   (when (:error-handler opts)
     (.setErrorHandler a (:error-handler opts)))
   (.setErrorMode a (or (:error-mode opts)
                        (if (:error-handler opts) :continue :fail)))
   a)))

(defn send
  "Dispatch an action to an agent. Returns the agent immediately.
  Subsequently, in a thread from a thread pool, the state of the agent
  will be set to the value of:

  (apply action-fn state-of-agent args)"
  {:added "1.0"
   :static true}
  [^clojure.lang.Agent a f & args]
  (.dispatch a (binding [*agent* a] (binding-conveyor-fn f)) args false))

(defn send-off
  "Dispatch a potentially blocking action to an agent. Returns the
  agent immediately. Subsequently, in a separate thread, the state of
  the agent will be set to the value of:

  (apply action-fn state-of-agent args)"
  {:added "1.0"
   :static true}
  [^clojure.lang.Agent a f & args]
  (.dispatch a (binding [*agent* a] (binding-conveyor-fn f)) args true))

(defn release-pending-sends
  "Normally, actions sent directly or indirectly during another action
  are held until the action completes (changes the agent's
  state). This function can be used to dispatch any pending sent
  actions immediately. This has no impact on actions sent during a
  transaction, which are still held until commit. If no action is
  occurring, does nothing. Returns the number of actions dispatched."
  {:added "1.0"
   :static true}
  [] (clojure.lang.Agent/releasePendingSends))

(defn add-watch

```

```

"Alpha - subject to change.
Adds a watch function to an agent/atom/var/ref reference. The watch
fn must be a fn of 4 args: a key, the reference, its old-state, its
new-state. Whenever the reference's state might have been changed,
any registered watches will have their functions called. The watch fn
will be called synchronously, on the agent's thread if an agent,
before any pending sends if agent or ref. Note that an atom's or
ref's state may have changed again prior to the fn call, so use
old/new-state rather than derefing the reference. Note also that watch
fns may be called from multiple threads simultaneously. Var watchers
are triggered only by root binding changes, not thread-local
set!s. Keys must be unique per reference, and can be used to remove
the watch with remove-watch, but are otherwise considered opaque by
the watch mechanism."
{:added "1.0"
 :static true}
[~clojure.lang.IRef reference key fn] (.addWatch reference key fn))

(defn remove-watch
"Alpha - subject to change.
Removes a watch (set by add-watch) from a reference"
{:added "1.0"
 :static true}
[~clojure.lang.IRef reference key]
(.removeWatch reference key))

(defn agent-error
"Returns the exception thrown during an asynchronous action of the
agent if the agent is failed. Returns nil if the agent is not
failed."
{:added "1.2"
 :static true}
[~clojure.lang.Agent a] (.getError a))

(defn restart-agent
"When an agent is failed, changes the agent state to new-state and
then un-fails the agent so that sends are allowed again. If
a :clear-actions true option is given, any actions queued on the
agent that were being held while it was failed will be discarded,
otherwise those held actions will proceed. The new-state must pass
the validator if any, or restart will throw an exception and the
agent will remain failed with its old state and error. Watchers, if
any, will NOT be notified of the new state. Throws an exception if
the agent is not failed."
{:added "1.2"
 :static true
}
[~clojure.lang.Agent a, new-state & options]
(let [opts (apply hash-map options)]
(.restart a new-state (if (:clear-actions opts) true false))))
```

```
(defn set-error-handler!
  "Sets the error-handler of agent a to handler-fn. If an action
  being run by the agent throws an exception or doesn't pass the
  validator fn, handler-fn will be called with two arguments: the
  agent and the exception."
  {:added "1.2"
   :static true}
  [^clojure.lang.Agent a, handler-fn]
  (.setErrorHandler a handler-fn))

(defn error-handler
  "Returns the error-handler of agent a, or nil if there is none.
  See set-error-handler!"
  {:added "1.2"
   :static true}
  [^clojure.lang.Agent a]
  (.getErrorHandler a))

(defn set-error-mode!
  "Sets the error-mode of agent a to mode-keyword, which must be
  either :fail or :continue. If an action being run by the agent
  throws an exception or doesn't pass the validator fn, an
  error-handler may be called (see set-error-handler!), after which,
  if the mode is :continue, the agent will continue as if neither the
  action that caused the error nor the error itself ever happened.

  If the mode is :fail, the agent will become failed and will stop
  accepting new 'send' and 'send-off' actions, and any previously
  queued actions will be held until a 'restart-agent'. Deref will
  still work, returning the state of the agent before the error."
  {:added "1.2"
   :static true}
  [^clojure.lang.Agent a, mode-keyword]
  (.setErrorMode a mode-keyword))

(defn error-mode
  "Returns the error-mode of agent a. See set-error-mode!"
  {:added "1.2"
   :static true}
  [^clojure.lang.Agent a]
  (.getErrorMode a))

(defn agent-errors
  "DEPRECATED: Use 'agent-error' instead.
  Returns a sequence of the exceptions thrown during asynchronous
  actions of the agent."
  {:added "1.0"
   :deprecated "1.2"}
  [a]
```

```
(when-let [e (agent-error a)]
  (list e)))

(defn clear-agent-errors
  "DEPRECATED: Use 'restart-agent' instead.
  Clears any exceptions thrown during asynchronous actions of the
  agent, allowing subsequent actions to occur."
  {:added "1.0"
   :deprecated "1.2"}
  [^clojure.lang.Agent a] (restart-agent a (.deref a)))

(defn shutdown-agents
  "Initiates a shutdown of the thread pools that back the agent
  system. Running actions will complete, but no new actions will be
  accepted"
  {:added "1.0"
   :static true}
  [] (. clojure.lang.Agent shutdown))

(defn ref
  "Creates and returns a Ref with an initial value of x and zero or
  more options (in any order):

  :meta metadata-map

  :validator validate-fn

  :min-history (default 0)
  :max-history (default 10)

  If metadata-map is supplied, it will become the metadata on the
  ref. validate-fn must be nil or a side-effect-free fn of one
  argument, which will be passed the intended new state on any state
  change. If the new state is unacceptable, the validate-fn should
  return false or throw an exception. validate-fn will be called on
  transaction commit, when all refs have their final values.

  Normally refs accumulate history dynamically as needed to deal with
  read demands. If you know in advance you will need history you can
  set :min-history to ensure it will be available when first needed
  (instead of after a read fault). History is limited, and the limit
  can be set with :max-history."
  {:added "1.0"
   :static true
   }
  ([x] (new clojure.lang.Ref x))
  ([x & options]
   (let [r ^clojure.lang.Ref (setup-reference (ref x) options)
         opts (apply hash-map options)]
     (when (:max-history opts)
```

```

(.setMaxHistory r (:max-history opts))
(when (:min-history opts)
  (.setMinHistory r (:min-history opts)))
r)))

(defn deref
  "Also reader macro: @ref/@agent/@var/@atom/@delay/@future.
  Within a transaction, returns the in-transaction-value of ref,
  else returns the most-recently-committed value of ref. When
  applied to a var, agent or atom, returns its current state.
  When applied to a delay, forces it if not already forced. When
  applied to a future, will block if computation not complete"
{:added "1.0"
 :static true}
[~clojure.lang.IDeref ref] (.deref ref))

(defn atom
  "Creates and returns an Atom with an initial value of x and zero or
  more options (in any order):

  :meta metadata-map

  :validator validate-fn

  If metadata-map is supplied, it will become the metadata on the
  atom. validate-fn must be nil or a side-effect-free fn of one
  argument, which will be passed the intended new state on any state
  change. If the new state is unacceptable, the validate-fn should
  return false or throw an exception."
{:added "1.0"
 :static true}
([x] (new clojure.lang.Atom x))
([x & options] (setup-reference (atom x) options)))

(defn swap!
  "Atomically swaps the value of atom to be:
  (apply f current-value-of-atom args). Note that f may be called
  multiple times, and thus should be free of side effects. Returns
  the value that was swapped in."
{:added "1.0"
 :static true}
([~clojure.lang.Atom atom f] (.swap atom f))
([~clojure.lang.Atom atom f x] (.swap atom f x))
([~clojure.lang.Atom atom f x y] (.swap atom f x y))
([~clojure.lang.Atom atom f x y & args] (.swap atom f x y args)))

(defn compare-and-set!
  "Atomically sets the value of atom to newval if and only if the
  current value of the atom is identical to oldval. Returns true if
  set happened, else false"

```

```

{:added "1.0"
 :static true}
[~clojure.lang.Atom atom oldval newval]
 (.compareAndSet atom oldval newval))

(defn reset!
  "Sets the value of atom to newval without regard for the
  current value. Returns newval."
{:added "1.0"
 :static true}
[~clojure.lang.Atom atom newval] (.reset atom newval))

(defn set-validator!
  "Sets the validator-fn for a var/ref/agent/atom. validator-fn
  must be nil or a side-effect-free fn of one argument, which will
  be passed the intended new state on any state change. If the new
  state is unacceptable, the validator-fn should return false or
  throw an exception. If the current state (root value if var)
  is not acceptable to the new validator, an exception will be
  thrown and the validator will not be changed."
{:added "1.0"
 :static true}
[~clojure.lang.IRef iref validator-fn]
 (. iref (setValidator validator-fn)))

(defn get-validator
  "Gets the validator-fn for a var/ref/agent/atom."
{:added "1.0"
 :static true}
[~clojure.lang.IRef iref] (. iref (getValidator)))

(defn alter-meta!
  "Atomically sets the metadata for a namespace/var/ref/agent/atom to be:

  (apply f its-current-meta args)

  f must be free of side-effects"
{:added "1.0"
 :static true}
[~clojure.lang.IReference iref f & args] (.alterMeta iref f args))

(defn reset-meta!
  "Atomically resets the metadata for a namespace/var/ref/agent/atom"
{:added "1.0"
 :static true}
[~clojure.lang.IReference iref metadata-map]
 (.resetMeta iref metadata-map))

(defn commute
  "Must be called in a transaction. Sets the in-transaction-value of

```

```

ref to:

(apply fun in-transaction-value-of-ref args)

and returns the in-transaction-value of ref.

At the commit point of the transaction, sets the value of ref to be:

(apply fun most-recently-committed-value-of-ref args)

Thus fun should be commutative, or, failing that, you must accept
last-one-in-wins behavior. commute allows for more concurrency than
ref-set."
{:added "1.0"
 :static true}

[^clojure.lang.Ref ref fun & args]
(. ref (commute fun args))

(defn alter
  "Must be called in a transaction. Sets the in-transaction-value of
  ref to:
  (apply fun in-transaction-value-of-ref args)

  and returns the in-transaction-value of ref."
  {:added "1.0"
   :static true}
  [^clojure.lang.Ref ref fun & args]
  (. ref (alter fun args)))

(defn ref-set
  "Must be called in a transaction. Sets the value of ref.
  Returns val."
  {:added "1.0"
   :static true}
  [^clojure.lang.Ref ref val]
  (. ref (set val)))

(defn ref-history-count
  "Returns the history count of a ref"
  {:added "1.1"
   :static true}
  [^clojure.lang.Ref ref]
  (.getHistoryCount ref))

(defn ref-min-history
  "Gets the min-history of a ref, or sets it and returns the ref"
  {:added "1.1"
   :static true})

```

```
([~clojure.lang.Ref ref]
  (.getMinHistory ref))
([~clojure.lang.Ref ref n]
  (.setMinHistory ref n)))

(defn ref-max-history
  "Gets the max-history of a ref, or sets it and returns the ref"
  {:added "1.1"
   :static true}
  ([~clojure.lang.Ref ref]
   (.getMaxHistory ref))
  ([~clojure.lang.Ref ref n]
   (.setMaxHistory ref n)))

(defn ensure
  "Must be called in a transaction. Protects the ref from modification
  by other transactions. Returns the in-transaction-value of
  ref. Allows for more concurrency than (ref-set ref @ref)"
  {:added "1.0"
   :static true}
  [^clojure.lang.Ref ref]
  (. ref (touch))
  (. ref (deref)))

(defmacro sync
  "transaction-flags => TBD, pass nil for now

Runs the exprs (in an implicit do) in a transaction that encompasses
exprs and any nested calls. Starts a transaction if none is already
running on this thread. Any uncaught exception will abort the
transaction and flow out of sync. The exprs may be run more than
once, but any effects on Refs will be atomic."
  {:added "1.0"}
  [flags-ignored-for-now & body]
  `(. clojure.lang.LockingTransaction
      (runInTransaction (fn [] ~@body))))"

(defmacro io!
  "If an io! block occurs in a transaction, throws an
IllegalStateException, else runs body in an implicit do. If the
first expression in body is a literal string, will use that as the
exception message."
  {:added "1.0"}
  [& body]
  (let [message (when (string? (first body)) (first body))
        body (if message (next body) body)]
    `'(if (clojure.lang.LockingTransaction/isRunning)
        (throw
         (new IllegalStateException ~(or message "I/O in transaction")))))
```

```

(do ~@body)))

;;;;;; fn stuff ;;;;;;;;

(defn comp
  "Takes a set of functions and returns a fn that is the composition
  of those fns. The returned fn takes a variable number of args,
  applies the rightmost of fns to the args, the next
  fn (right-to-left) to the result, etc."
  {:added "1.0"
   :static true}
  ([] identity)
  ([f] f)
  ([f g]
   (fn
     ([] (f (g)))
     ([x] (f (g x)))
     ([x y] (f (g x y)))
     ([x y z] (f (g x y z)))
     ([x y z & args] (f (apply g x y z args)))))
  ([f g h]
   (fn
     ([] (f (g (h))))
     ([x] (f (g (h x))))
     ([x y] (f (g (h x y))))
     ([x y z] (f (g (h x y z))))
     ([x y z & args] (f (g (apply h x y z args))))))
  ([f1 f2 f3 & fs]
   (let [fs (reverse (list* f1 f2 f3 fs))]
     (fn [& args]
       (loop [ret (apply (first fs) args) fs (next fs)]
         (if fs
             (recur ((first fs) ret) (next fs))
             ret)))))

(defn juxtap
  "Alpha - name subject to change.
  Takes a set of functions and returns a fn that is the juxtaposition
  of those fns. The returned fn takes a variable number of args, and
  returns a vector containing the result of applying each fn to the
  args (left-to-right).
  ((juxtap a b c) x) => [(a x) (b x) (c x)]"
  {:added "1.1"
   :static true}
  ([f]
   (fn
     ([] [(f)])
     ([x] [(f x)])
     ([x y] [(f x y)])))

```

```

([x y z] [(f x y z)])
([x y z & args] [(apply f x y z args)]))

([f g]
 (fn
  ([] [(f) (g)])
  ([x] [(f x) (g x)])
  ([x y] [(f x y) (g x y)])
  ([x y z] [(f x y z) (g x y z)])
  ([x y z & args] [(apply f x y z args) (apply g x y z args)])))

([f g h]
 (fn
  ([] [(f) (g) (h)])
  ([x] [(f x) (g x) (h x)])
  ([x y] [(f x y) (g x y) (h x y)])
  ([x y z] [(f x y z) (g x y z) (h x y z)])
  ([x y z & args]
   [(apply f x y z args)
    (apply g x y z args)
    (apply h x y z args)])))

([f g h & fs]
 (let [fs (list* f g h fs)]
  (fn
   ([] (reduce1 #(conj %1 (%2)) [] fs))
   ([x] (reduce1 #(conj %1 (%2 x)) [] fs))
   ([x y] (reduce1 #(conj %1 (%2 x y)) [] fs))
   ([x y z] (reduce1 #(conj %1 (%2 x y z)) [] fs))
   ([x y z & args]
    (reduce1 #(conj %1 (apply %2 x y z args)) [] fs)))))

(defn partial
  "Takes a function f and fewer than the normal arguments to f, and
  returns a fn that takes a variable number of additional args. When
  called, the returned function calls f with args + additional args."
  {:added "1.0"
   :static true}
  ([f arg1]
   (fn [& args] (apply f arg1 args)))
  ([f arg1 arg2]
   (fn [& args] (apply f arg1 arg2 args)))
  ([f arg1 arg2 arg3]
   (fn [& args] (apply f arg1 arg2 arg3 args)))
  ([f arg1 arg2 arg3 & more]
   (fn [& args] (apply f arg1 arg2 arg3 (concat more args)))))

;;;;;; sequence fns ;;;;;;;
(defn sequence
  "Coerces coll to a (possibly empty) sequence, if it is not already
  one. Will not force a lazy seq. (sequence nil) yields ()"
  {:added "1.0"
   :static true}

```

```

[coll]
(if (seq? coll) coll
  (or (seq coll) ()))

(defn every?
  "Returns true if (pred x) is logical true for every x in coll, else
  false."
{:tag Boolean
 :added "1.0"
 :static true}
[pred coll]
(cond
  (nil? (seq coll)) true
  (pred (first coll)) (recur pred (next coll))
  :else false))

(def
^{:tag Boolean
 :doc "Returns false if (pred x) is logical true for every x in
coll, else true."
 :arglists '([pred coll])
 :added "1.0"}
not-every? (comp not every?))

(defn some
  "Returns the first logical true value of (pred x) for any x in coll,
  else nil. One common idiom is to use a set as pred, for example
  this will return :fred if :fred is in the sequence, otherwise nil:
  (some #{:fred} coll)"
{:added "1.0"
 :static true}
[pred coll]
(when (seq coll)
  (or (pred (first coll)) (recur pred (next coll)))))

(def
^{:tag Boolean
 :doc "Returns false if (pred x) is logical true for any x in coll,
else true."
 :arglists '([pred coll])
 :added "1.0"}
not-any? (comp not some))

;will be redefined later with arg checks
(defmacro dotimes
  "bindings => name n

Repeatedly executes body (presumably for side-effects) with name
bound to integers from 0 through n-1."
{:added "1.0"})

```

```
[bindings & body]
(let [i (first bindings)
      n (second bindings)]
  '(let [n# (clojure.lang.RT/longCast ~n)]
    (loop [~i 0]
      (when (< ~i n#)
        ~@body
        (recur (unchecked-inc ~i))))))

(defn map
  "Returns a lazy sequence consisting of the result of applying f to the
  set of first items of each coll, followed by applying f to the set
  of second items in each coll, until any one of the colls is
  exhausted. Any remaining items in other colls are ignored. Function
  f should accept number-of-colls arguments."
  {:added "1.0"
   :static true}
  ([f coll]
   (lazy-seq
     (when-let [s (seq coll)]
       (if (chunked-seq? s)
           (let [c (chunk-first s)
                 size (int (count c))
                 b (chunk-buffer size)]
             (dotimes [i size]
               (chunk-append b (f (.nth c i))))
             (chunk-cons (chunk b) (map f (chunk-rest s))))
             (cons (f (first s)) (map f (rest s))))))
     ([f c1 c2]
      (lazy-seq
        (let [s1 (seq c1) s2 (seq c2)]
          (when (and s1 s2)
            (cons (f (first s1) (first s2))
                  (map f (rest s1) (rest s2)))))))
     ([f c1 c2 c3]
      (lazy-seq
        (let [s1 (seq c1) s2 (seq c2) s3 (seq c3)]
          (when (and s1 s2 s3)
            (cons (f (first s1) (first s2) (first s3))
                  (map f (rest s1) (rest s2) (rest s3)))))))
     ([f c1 c2 c3 & colls]
      (let [step (fn step [cs]
                  (lazy-seq
                    (let [ss (map seq cs)]
                      (when (every? identity ss)
                        (cons (map first ss) (step (map rest ss))))))]
                  (map #(apply f %) (step (conj colls c3 c2 c1))))))

(defn mapcat
  "Returns the result of applying concat to the result of applying map
```

```

to f and colls. Thus function f should return a collection."
{:added "1.0"
 :static true}
[f & colls]
(apply concat (apply map f colls))

(defn filter
  "Returns a lazy sequence of the items in coll for which
(pred item) returns true. pred must be free of side-effects."
{:added "1.0"
 :static true}
([pred coll]
(lazy-seq
  (when-let [s (seq coll)]
    (if (chunked-seq? s)
        (let [c (chunk-first s)
              size (count c)
              b (chunk-buffer size)]
          (dotimes [i size]
            (when (pred (.nth c i))
              (chunk-append b (.nth c i))))
          (chunk-cons (chunk b) (filter pred (chunk-rest s)))))
        (let [f (first s) r (rest s)]
          (if (pred f)
              (cons f (filter pred r))
              (filter pred r))))))
)

(defn remove
  "Returns a lazy sequence of the items in coll for which
(pred item) returns false. pred must be free of side-effects."
{:added "1.0"
 :static true}
[pred coll]
(filter (complement pred) coll))

(defn take
  "Returns a lazy sequence of the first n items in coll, or all items if
there are fewer than n."
{:added "1.0"
 :static true}
[n coll]
(lazy-seq
  (when (pos? n)
    (when-let [s (seq coll)]
      (cons (first s) (take (dec n) (rest s)))))))

(defn take-while
  "Returns a lazy sequence of successive items from coll while
(pred item) returns true. pred must be free of side-effects."

```

```

{:added "1.0"
:static true}
[pred coll]
(lazy-seq
(when-let [s (seq coll)]
  (when (pred (first s))
    (cons (first s) (take-while pred (rest s)))))))

(defn drop
  "Returns a lazy sequence of all but the first n items in coll."
{:added "1.0"
:static true}
[n coll]
(let [step (fn [n coll]
  (let [s (seq coll)]
    (if (and (pos? n) s)
      (recur (dec n) (rest s))
      s))))]
  (lazy-seq (step n coll)))))

(defn drop-last
  "Return a lazy sequence of all but the last n (default 1) items
  in coll"
{:added "1.0"
:static true}
([s] (drop-last 1 s))
([n s] (map (fn [x _] x) s (drop n s)))))

(defn take-last
  "Returns a seq of the last n items in coll. Depending on the type
  of coll may be no better than linear time. For vectors,
  see also subvec."
{:added "1.1"
:static true}
[n coll]
(loop [s (seq coll), lead (seq (drop n coll))]
  (if lead
    (recur (next s) (next lead))
    s)))

(defn drop-while
  "Returns a lazy sequence of the items in coll starting from the first
  item for which (pred item) returns nil."
{:added "1.0"
:static true}
[pred coll]
(let [step (fn [pred coll]
  (let [s (seq coll)]
    (if (and s (pred (first s)))
      (recur pred (rest s))
      s))))]
  (lazy-seq (step pred coll)))))


```

```

          s)))]
(lazy-seq (step pred coll)))

(defn cycle
  "Returns a lazy (infinite!) sequence of repetitions of the items
  in coll."
{:added "1.0"
 :static true}
[coll] (lazy-seq
         (when-let [s (seq coll)]
           (concat s (cycle s)))))

(defn split-at
  "Returns a vector of [(take n coll) (drop n coll)]"
{:added "1.0"
 :static true}
[n coll]
  [(take n coll) (drop n coll)])

(defn split-with
  "Returns a vector of [(take-while pred coll) (drop-while pred coll)]"
{:added "1.0"
 :static true}
[pred coll]
  [(take-while pred coll) (drop-while pred coll)])

(defn repeat
  "Returns a lazy (infinite!, or length n if supplied) sequence of xs."
{:added "1.0"
 :static true}
([x] (lazy-seq (cons x (repeat x))))
 ([n x] (take n (repeat x)))))

(defn replicate
  "DEPRECATED: Use 'repeat' instead.
  Returns a lazy seq of n xs."
{:added "1.0"
 :deprecated "1.3"}
[n x] (take n (repeat x)))

(defn iterate
  "Returns a lazy sequence of x, (f x), (f (f x)) etc.
  f must be free of side-effects"
{:added "1.0"
 :static true}
[f x] (cons x (lazy-seq (iterate f (f x)))))

(defn range
  "Returns a lazy seq of nums from start (inclusive) to end
  (exclusive), by step, where start defaults to 0, step to 1, and end
  "

```

```

to infinity."
{:added "1.0"
:static true}
([] (range 0 Double/POSITIVE_INFINITY 1))
([end] (range 0 end 1))
([start end] (range start end 1))
([start end step]
(lazy-seq
(let [b (chunk-buffer 32)
      comp (if (pos? step) < >)]
(loop [i start]
(if (and (< (count b) 32)
        (comp i end))
(do
(chunk-append b i)
(recur (+ i step)))
(chunk-cons (chunk b)
(when (comp i end)
(range i end step))))))))
(defn merge
"Returns a map that consists of the rest of the maps conj-ed onto
the first. If a key occurs in more than one map, the mapping from
the latter (left-to-right) will be the mapping in the result."
{:added "1.0"
:static true}
[& maps]
(when (some identity maps)
(reduce1 #(conj (or %1 {}) %2) maps)))

(defn merge-with
"Returns a map that consists of the rest of the maps conj-ed onto
the first. If a key occurs in more than one map, the mapping(s)
from the latter (left-to-right) will be combined with the mapping in
the result by calling (f val-in-result val-in-latter)."
{:added "1.0"
:static true}
[f & maps]
(when (some identity maps)
(let [merge-entry (fn [m e]
(let [k (key e) v (val e)]
(if (contains? m k)
(assoc m k (f (get m k) v))
(assoc m k v))))
merge2 (fn [m1 m2]
(reduce1 merge-entry (or m1 {}) (seq m2)))
(reduce1 merge2 maps))))
```

```
(defn zipmap
  "Returns a map with the keys mapped to the corresponding vals."
  {:added "1.0"
   :static true}
  [keys vals]
  (loop [map {}]
    (ks (seq keys)
         vs (seq vals))
    (if (and ks vs)
        (recur (assoc map (first ks) (first vs))
               (next ks)
               (next vs)))
        map)))

(defmacro declare
  "defs the supplied var names with no bindings,
  useful for making forward declarations."
  {:added "1.0"}
  [& names]
  '(do
    ~@(#(list 'def (vary-meta % assoc :declared true)) names)))

(defn line-seq
  "Returns the lines of text from rdr as a lazy sequence of strings.
  rdr must implement java.io.BufferedReader."
  {:added "1.0"
   :static true}
  [^java.io.BufferedReader rdr]
  (when-let [line (.readLine rdr)]
    (cons line (lazy-seq (line-seq rdr)))))

(defn comparator
  "Returns an implementation of java.util.Comparator based upon pred."
  {:added "1.0"
   :static true}
  [pred]
  (fn [x y]
    (cond (pred x y) -1 (pred y x) 1 :else 0)))

(defn sort
  "Returns a sorted sequence of the items in coll. If no comparator is
  supplied, uses compare. comparator must
  implement java.util.Comparator."
  {:added "1.0"
   :static true}
  ([coll]
   (sort compare coll))
  ([^java.util.Comparator comp coll]
   (if (seq coll)
       (let [a (to-array coll)]
```

```

(. java.util.Arrays (sort a comp))
  (seq a))
  ()))

(defn sort-by
  "Returns a sorted sequence of the items in coll, where the sort
  order is determined by comparing (keyfn item). If no comparator is
  supplied, uses compare. comparator must
  implement java.util.Comparator."
  {:added "1.0"
   :static true}
  ([keyfn coll]
   (sort-by keyfn compare coll))
  ([keyfn ^java.util.Comparator comp coll]
   (sort (fn [x y] (. comp (compare (keyfn x) (keyfn y)))) coll)))

(defn partition
  "Returns a lazy sequence of lists of n items each, at offsets step
  apart. If step is not supplied, defaults to n, i.e. the partitions
  do not overlap. If a pad collection is supplied, use its elements
  as necessary to complete last partition upto n items. In case
  there are not enough padding elements, return a partition with
  less than n items."
  {:added "1.0"
   :static true}
  ([n coll]
   (partition n n coll))
  ([n step coll]
   (lazy-seq
    (when-let [s (seq coll)]
      (let [p (take n s)]
        (when (= n (count p))
          (cons p (partition n step (drop step s)))))))
  ([n step pad coll]
   (lazy-seq
    (when-let [s (seq coll)]
      (let [p (take n s)]
        (if (= n (count p))
            (cons p (partition n step pad (drop step s)))
            (list (take n (concat p pad)))))))

;; evaluation

(defn eval
  "Evaluates the form data structure (not text!) and returns the result."
  {:added "1.0"
   :static true}
  [form] (. clojure.lang.Compiler (eval form)))

(defmacro doseq

```

```

"Repeatedly executes body (presumably for side-effects) with
bindings and filtering as provided by \"for\". Does not retain
the head of the sequence. Returns nil."
{:added "1.0"}
[seq-exprs & body]
(assert-args doseq
  (vector? seq-exprs) "a vector for its binding"
  (even? (count seq-exprs))
    "an even number of forms in binding vector")
(let [step (fn step [recform exprs]
  (if-not exprs
    [true 'do ~@body]
    (let [k (first exprs)
      v (second exprs)]
      (if (keyword? k)
        (let [steppair (step recform (nnext exprs))
          needrec (steppair 0)
          subform (steppair 1)]
          (cond
            (= k :let) [needrec '(let ~v ~subform)]
            (= k :while)
              [false '(when ~v
                ~subform
                ~@(when needrec [recform]))]
            (= k :when)
              [false '(if ~v
                (do
                  ~subform
                  ~@(when needrec [recform]))
                ~recform)])
        (let [seq- (gensym "seq_")]
          chunk-
          (with-meta (gensym "chunk_")
            {:tag 'clojure.lang.IChunk})
          count- (gensym "count_")
          i- (gensym "i_")
          recform '(recur (next ~seq-) nil 0 0)
          steppair (step recform (nnext exprs))
          needrec (steppair 0)
          subform (steppair 1)
          recform-chunk
            '(recur ~seq- ~chunk- ~count-
              (unchecked-inc ~i-))
          steppair-chunk
            (step recform-chunk (nnext exprs))
          subform-chunk (steppair-chunk 1)]
        [true
          '(loop [~seq- (seq ~v), ~chunk- nil,
            ~count- 0, ~i- 0]
            (if (< ~i- ~count-)

```

```

(let [~k (.nth ~chunk- ~i-)]
  ~subform-chunk
  ~@(when needrec [recform-chunk]))
(when-let [~seq- (seq ~seq-)]
  (if (chunked-seq? ~seq-)
    (let [c# (chunk-first ~seq-)]
      (recur (chunk-rest ~seq-) c#
             (int (count c#)) (int 0)))
    (let [~k (first ~seq-)]
      ~subform
      ~@(when needrec [recform]))))))))))]
  (nth (step nil (seq seq-exprs)) 1)))

(defn dorun
  "When lazy sequences are produced via functions that have side
  effects, any effects other than those needed to produce the first
  element in the seq do not occur until the seq is consumed. dorun can
  be used to force any effects. Walks through the successive nexts of
  the seq, does not retain the head and returns nil."
  {:added "1.0"
   :static true}
  ([coll]
   (when (seq coll)
     (recur (next coll))))
  ([n coll]
   (when (and (seq coll) (pos? n))
     (recur (dec n) (next coll)))))

(defn doall
  "When lazy sequences are produced via functions that have side
  effects, any effects other than those needed to produce the first
  element in the seq do not occur until the seq is consumed. doall can
  be used to force any effects. Walks through the successive nexts of
  the seq, retains the head and returns it, thus causing the entire
  seq to reside in memory at one time."
  {:added "1.0"
   :static true}
  ([coll]
   (dorun coll)
   coll)
  ([n coll]
   (dorun n coll)
   coll))

(defn await
  "Blocks the current thread (indefinitely!) until all actions
  dispatched thus far, from this thread or agent, to the agent(s) have
  occurred. Will block on failed agents. Will never return if
  a failed agent is restarted with :clear-actions true."
  {:added "1.0"
   :static true}
  ([agent]
   (dorun agent)
   agent)
  ([agents]
   (dorun agents)
   agents))

```

```

:static true}
[& agents]
(io! "await in transaction"
  (when *agent*
    (throw (new Exception "Can't await in agent action")))
  (let [latch (new java.util.concurrent.CountDownLatch (count agents))
        count-down (fn [agent] (. latch (countDown)) agent)]
    (doseq [agent agents]
      (send agent count-down))
    (. latch (await)))))

(defn ^:static await1 [^clojure.lang.Agent a]
  (when (pos? (.getQueueCount a))
    (await a))
  a)

(defn await-for
  "Blocks the current thread until all actions dispatched thus
  far (from this thread or agent) to the agents have occurred, or the
  timeout (in milliseconds) has elapsed. Returns nil if returning due
  to timeout, non-nil otherwise."
  {:added "1.0"
   :static true}
  [timeout-ms & agents]
  (io! "await-for in transaction"
    (when *agent*
      (throw (new Exception "Can't await in agent action")))
    (let [latch (new java.util.concurrent.CountDownLatch (count agents))
          count-down (fn [agent] (. latch (countDown)) agent)]
      (doseq [agent agents]
        (send agent count-down))
      (. latch
        (await timeout-ms
          (. java.util.concurrent.TimeUnit MILLISECONDS))))))

(defmacro dotimes
  "bindings => name n

Repeatedly executes body (presumably for side-effects) with name
bound to integers from 0 through n-1."
  {:added "1.0"}
  [bindings & body]
  (assert-args dotimes
    (vector? bindings) "a vector for its binding"
    (= 2 (count bindings)) "exactly 2 forms in binding vector")
  (let [i (first bindings)
        n (second bindings)]
    `(let [n# (long ~n)]
       (loop [~i 0]
         (when (< ~i n#)

```

```

    ~@body
    (recur (unchecked-inc ~i))))))

#_(defn into
  "Returns a new coll consisting of to-coll with all of the items of
  from-coll conjoined."
  {:added "1.0"}
  [to from]
  (let [ret to items (seq from)]
    (if items
        (recur (conj ret (first items)) (next items))
        ret)))

;;;;;;;;
\getchunk{defn transient}

\getchunk{defn persistent!}

\getchunk{defn conj!}

\getchunk{defn assoc!}

\getchunk{defn dissoc!}

\getchunk{defn pop!}

\getchunk{defn disj!}

;redef into with batch support
(defn ^:private into1
  "Returns a new coll consisting of to-coll with all of the items of
  from-coll conjoined."
  {:added "1.0"
   :static true}
  [to from]
  (if (instance? clojure.lang.IEditableCollection to)
      (persistent! (reduce1 conj! (transient to) from))
      (reduce1 conj to from)))

(defmacro import
  "import-list => (package-symbol class-name-symbols*)

  For each name in class-name-symbols, adds a mapping from name to the
  class named by package.name to the current namespace. Use :import
  in the ns macro in preference to calling this directly."
  {:added "1.0"}
  [& import-symbols-or-lists]
  (let [specs (map #(if (and (seq? %) (= 'quote (first %))) (second %) %)
                    import-symbols-or-lists)]
    '(do ~(@map #'(list 'clojure.core/import* %)
```

```

(reduce1 (fn [v spec]
            (if (symbol? spec)
                (conj v (name spec))
                (let [p (first spec) cs (rest spec)]
                    (into1 v (map #(str p "." %) cs))))
            [] specs)))))

(defn into-array
  "Returns an array with components set to the values in aseq. The
  array's component type is type if provided, or the type of the
  first value in aseq if present, or Object. All values in aseq
  must be compatible with the component type. Class objects for
  must be compatible with the primitive types can be obtained
  using, e.g., Integer/TYPE."
  {:added "1.0"
   :static true}
  ([aseq]
   (closure.lang.RT/seqToTypedArray (seq aseq)))
  ([type aseq]
   (closure.lang.RT/seqToTypedArray type (seq aseq)))))

(defn ^{:private true}
  array [& items]
  (into-array items))

(defn class
  "Returns the Class of x"
  {:added "1.0"
   :static true}
  ^Class [^Object x] (if (nil? x) x (. x getClass)))

(defn type
  "Returns the :type metadata of x, or its Class if none"
  {:added "1.0"
   :static true}
  [x]
  (or (get (meta x) :type) (class x)))

(defn num
  "Coerce to Number"
  {:tag Number
   :inline (fn [x] '(. closure.lang.Numbers (num ~x)))
   :added "1.0"}
  [x] (. closure.lang.Numbers (num x)))

(defn long
  "Coerce to long"
  {:inline (fn [x] '(. closure.lang.RT (longCast ~x)))
   :added "1.0"}
  [^Number x] (closure.lang.RT/longCast x))

```

```
(defn float
  "Coerce to float"
  {:inline (fn [x]
    `(. clojure.lang.RT
        ~(if *unchecked-math* 'uncheckedFloatCast 'floatCast) ~x)))
  :added "1.0"}
  [^Number x] (clojure.lang.RT/floatCast x))

(defn double
  "Coerce to double"
  {:inline (fn [x] `(. clojure.lang.RT (doubleCast ~x)))
  :added "1.0"}
  [^Number x] (clojure.lang.RT/doubleCast x))

(defn short
  "Coerce to short"
  {:inline
   (fn [x]
     `(. clojure.lang.RT
         ~(if *unchecked-math* 'uncheckedShortCast 'shortCast) ~x)))
  :added "1.0"}
  [^Number x] (clojure.lang.RT/shortCast x))

(defn byte
  "Coerce to byte"
  {:inline
   (fn [x]
     `(. clojure.lang.RT
         ~(if *unchecked-math* 'uncheckedByteCast 'byteCast) ~x)))
  :added "1.0"}
  [^Number x] (clojure.lang.RT/byteCast x))

(defn char
  "Coerce to char"
  {:inline
   (fn [x]
     `(. clojure.lang.RT
         ~(if *unchecked-math* 'uncheckedCharCast 'charCast) ~x)))
  :added "1.1"}
  [x] (. clojure.lang.RT (charCast x)))

(defn boolean
  "Coerce to boolean"
  {
  :inline (fn [x] `(. clojure.lang.RT (booleanCast ~x)))
  :added "1.0"}
  [x] (clojure.lang.RT/booleanCast x))

(defn unchecked-byte
```

```

"Coerce to byte. Subject to rounding or truncation."
{:inline (fn [x] `(. clojure.lang.RT (uncheckedByteCast ~x)))
 :added "1.3"}
[~Number x] (clojure.lang.RT/uncheckedByteCast x))

(defn unchecked-short
"Coerce to short. Subject to rounding or truncation."
{:inline (fn [x] `(. clojure.lang.RT (uncheckedShortCast ~x)))
 :added "1.3"}
[~Number x] (clojure.lang.RT/uncheckedShortCast x))

(defn unchecked-char
"Coerce to char. Subject to rounding or truncation."
{:inline (fn [x] `(. clojure.lang.RT (uncheckedCharCast ~x)))
 :added "1.3"}
[x] (. clojure.lang.RT (uncheckedCharCast x)))

(defn unchecked-int
"Coerce to int. Subject to rounding or truncation."
{:inline (fn [x] `(. clojure.lang.RT (uncheckedIntCast ~x)))
 :added "1.3"}
[~Number x] (clojure.lang.RT/uncheckedIntCast x))

(defn unchecked-long
"Coerce to long. Subject to rounding or truncation."
{:inline (fn [x] `(. clojure.lang.RT (uncheckedLongCast ~x)))
 :added "1.3"}
[~Number x] (clojure.lang.RT/uncheckedLongCast x))

(defn unchecked-float
"Coerce to float. Subject to rounding."
{:inline (fn [x] `(. clojure.lang.RT (uncheckedFloatCast ~x)))
 :added "1.3"}
[~Number x] (clojure.lang.RT/uncheckedFloatCast x))

(defn unchecked-double
"Coerce to double. Subject to rounding."
{:inline (fn [x] `(. clojure.lang.RT (uncheckedDoubleCast ~x)))
 :added "1.3"}
[~Number x] (clojure.lang.RT/uncheckedDoubleCast x))

(defn number?
"Returns true if x is a Number"
{:added "1.0"
 :static true}
[x]
(instance? Number x))

(defn integer?

```

```

"Returns true if n is an integer"
{:added "1.0"
 :static true}
[n]
(or (instance? Integer n)
     (instance? Long n)
     (instance? clojure.lang.BigInt n)
     (instance? BigInteger n)
     (instance? Short n)
     (instance? Byte n)))

(defn mod
  "Modulus of num and div. Truncates toward negative infinity."
  {:added "1.0"
   :static true}
  [num div]
  (let [m (rem num div)]
    (if (or (zero? m) (pos? (* num div)))
        m
        (+ m div)))))

(defn ratio?
  "Returns true if n is a Ratio"
  {:added "1.0"
   :static true}
  [n] (instance? clojure.lang.Ratio n))

(defn numerator
  "Returns the numerator part of a Ratio."
  {:tag BigInteger
   :added "1.2"
   :static true}
  [r]
  (.numerator ^clojure.lang.Ratio r))

(defn denominator
  "Returns the denominator part of a Ratio."
  {:tag BigInteger
   :added "1.2"
   :static true}
  [r]
  (.denominator ^clojure.lang.Ratio r))

(defn decimal?
  "Returns true if n is a BigDecimal"
  {:added "1.0"
   :static true}
  [n] (instance? BigDecimal n))

(defn float?

```

```

"Returns true if n is a floating point number"
{:added "1.0"
 :static true}
[n]
(or (instance? Double n)
     (instance? Float n)))

(defn rational? [n]
  "Returns true if n is a rational number"
  {:added "1.0"
   :static true}
  (or (integer? n) (ratio? n) (decimal? n)))

(defn bigint
  "Coerce to BigInt"
  {:tag clojure.lang.BigInt
   :static true
   :added "1.3"}
  [x] (cond
        (instance? clojure.lang.BigInt x) x
        (instance? BigInteger x) (clojure.lang.BigInt/fromBigInteger x)
        (decimal? x) (bigint (.toBigInteger ^BigDecimal x))
        (ratio? x) (bigint (.bigIntegerValue ^clojure.lang.Ratio x))
        (number? x) (clojure.lang.BigInt/valueOf (long x))
        :else (bigint (BigInteger. x)))))

(defn biginteger
  "Coerce to BigInteger"
  {:tag BigInteger
   :added "1.0"
   :static true}
  [x] (cond
        (instance? BigInteger x) x
        (instance? clojure.lang.BigInt x)
        (.toBigInteger ^clojure.lang.BigInt x)
        (decimal? x) (.toBigInteger ^BigDecimal x)
        (ratio? x) (.bigIntegerValue ^clojure.lang.Ratio x)
        (number? x) (BigInteger/valueOf (long x))
        :else (BigInteger. x)))))

(defn bigdec
  "Coerce to BigDecimal"
  {:tag BigDecimal
   :added "1.0"
   :static true}
  [x] (cond
        (decimal? x) x
        (float? x) (. BigDecimal valueOf (double x))
        (ratio? x) (/ (BigDecimal. (.numerator x)) (.denominator x))
        (instance? BigInteger x) (BigDecimal. ^BigInteger x)
        :else (BigDecimal. (.toString x))))
```

```

(number? x) (BigDecimal/valueOf (long x))
:else (BigDecimal. x)))

(def ^{:dynamic {:private true} print-initialized false}

(defmulti print-method (fn [x writer] (type x)))
(defmulti print-dup (fn [x writer] (class x)))

(defn pr-on
  {:private true
   :static true}
  [x w]
  (if *print-dup*
      (print-dup x w)
      (print-method x w)))
  nil)

(defn pr
  "Prints the object(s) to the output stream that is the current value
  of *out*. Prints the object(s), separated by spaces if there is
  more than one. By default, pr and prn print in a way that objects
  can be read by the reader"
  {:dynamic true
   :added "1.0"}
  ([] nil)
  ([x]
   (pr-on x *out*))
  ([x & more]
   (pr x)
   (. *out* (append \space))
   (if-let [nmore (next more)]
       (recur (first more) nmore)
       (apply pr more)))))

(def ^{:private ^String system-newline
       (System/getProperty "line.separator")}

(defn newline
  "Writes a platform-specific newline to *out*"
  {:added "1.0"
   :static true}
  []
  (. *out* (append system-newline))
  nil)

(defn flush
  "Flushes the output stream that is the current value of
  *out*"
  {:added "1.0"
   :static true}

```

```

[]

(. *out* (flush))
nil)

(defn prn
  "Same as pr followed by (newline). Observes *flush-on-newline*"
  {:added "1.0"
   :static true}
  [& more]
  (apply pr more)
  (newline)
  (when *flush-on-newline*
    (flush)))

(defn print
  "Prints the object(s) to the output stream that is the current value
  of *out*. print and println produce output for human consumption."
  {:added "1.0"
   :static true}
  [& more]
  (binding [*print-readably* nil]
    (apply pr more)))

(defn println
  "Same as print followed by (newline)"
  {:added "1.0"
   :static true}
  [& more]
  (binding [*print-readably* nil]
    (apply prn more)))

(defn read
  "Reads the next object from stream, which must be an instance of
  java.io.PushbackReader or some derivee. stream defaults to the
  current value of *in* ."
  {:added "1.0"
   :static true}
  []
  (read *in*))
  ([stream]
   (read stream true nil))
  ([stream eof-error? eof-value]
   (read stream eof-error? eof-value false))
  ([stream eof-error? eof-value recursive?]
   (. clojure.lang.LispReader
     (read stream (boolean eof-error?) eof-value recursive?)))))

(defn read-line
  "Reads the next line from stream that is the current value of *in* ."
  {:added "1.0"

```

```

:static true}
[]
(if (instance? clojure.lang.LineNumberingPushbackReader *in*)
    (.readLine ^clojure.lang.LineNumberingPushbackReader *in*)
    (.readLine ^java.io.BufferedReader *in*)))

(defn read-string
  "Reads one object from the string s"
  {:added "1.0"
   :static true}
  [s] (clojure.lang.RT/readString s))

(defn subvec
  "Returns a persistent vector of the items in vector from
  start (inclusive) to end (exclusive). If end is not supplied,
  defaults to (count vector). This operation is O(1) and very fast, as
  the resulting vector shares structure with the original and no
  trimming is done."
  {:added "1.0"
   :static true}
  ([v start]
   (subvec v start (count v)))
  ([v start end]
   (. clojure.lang.RT (subvec v start end)))))

(defmacro with-open
  "bindings => [name init ...]

  Evaluates body in a try expression with names bound to the values
  of the inits, and a finally clause that calls (.close name) on each
  name in reverse order."
  {:added "1.0"}
  [bindings & body]
  (assert-args with-open
    (vector? bindings) "a vector for its binding"
    (even? (count bindings))
    "an even number of forms in binding vector")
  (cond
    (= (count bindings) 0) `(do ~@body)
    (symbol? (bindings 0)) `(~(let ~(subvec bindings 0 2)
                                    (try
                                      (with-open ~(subvec bindings 2) ~@body)
                                      (finally
                                        (. ~(bindings 0) close)))))
    :else (throw (IllegalArgumentException.
                  "with-open only allows Symbols in bindings"))))

(defmacro doto
  "Evaluates x then calls all of the methods and functions with the
  value of x supplied at the front of the given arguments. The forms

```

```

are evaluated in order. Returns x.

(doto (new java.util.HashMap) (.put \"a\" 1) (.put \"b\" 2))
{:added "1.0"}
[x & forms]
(let [gx (gensym)]
  '(let [~gx ~x]
    ~(@(map (fn [f]
      (if (seq? f)
        `(~(first f) ~gx ~(@next f))
        `(~f ~gx)))
      forms)
    ~gx)))

(defmacro memfn
  "Expands into code that creates a fn that expects to be passed an
  object and any args and calls the named instance method on the
  object passing the args. Use when you want to treat a Java method as
  a first-class fn."
{:added "1.0"}
[name & args]
'(fn [target# ~@args]
  (. target# (~name ~@args)))))

(defmacro time
  "Evaluates expr and prints the time it took. Returns the value of
  expr."
{:added "1.0"}
[expr]
'(let [start# (. System (nanoTime))
       ret# ~expr]
  (prn (str "Elapsed time: "
            (/ (double (- (. System (nanoTime)) start#)) 1000000.0)
            " msecs"))
  ret#))

(import '(java.lang.reflect Array))

(defn alength
  "Returns the length of the Java array. Works on arrays of all
  types."
{:inline (fn [a] `(. clojure.lang.RT (alength ~a)))
 :added "1.0"}
[array] (. clojure.lang.RT (alength array)))

(defn aclone
  "Returns a clone of the Java array. Works on arrays of known
  types."

```

```

{:inline (fn [a] '(. clojure.lang.RT (aclone ~a)))
 :added "1.0"}
[array] (. clojure.lang.RT (aclone array)))

(defnaget
  "Returns the value at the index/indices. Works on Java arrays of all
  types."
  {:inline (fn [a i] '(. clojure.lang.RT (aget ~a (int ~i))))
   :inline-arities #{2}
   :added "1.0"}
  ([array idx]
   (clojure.lang.Reflector/prepRet
    (.getComponentType (class array)) (. Array (get array idx))))
  ([array idx & idxs]
   (apply aget (aget array idx) idxs)))

(defnaset
  "Sets the value at the index/indices. Works on Java arrays of
  reference types. Returns val."
  {:inline (fn [a i v] '(. clojure.lang.RT (aset ~a (int ~i) ~v)))
   :inline-arities #{3}
   :added "1.0"}
  ([array idx val]
   (. Array (set array idx val))
   val)
  ([array idx idx2 & idxv]
   (apply aset (aget array idx) idx2 idxv)))

(defmacro
  `{:private true}
  def-aset [name method coerce]
  `(defn ~name
     {:arglists
      '([`array `idx `val] [`array `idx `idx2 & `idxv])}
     ([array# idx# val#]
      (. Array (~method array# idx# (~coerce val#)))
      val#)
     ([array# idx# idx2# & idxv#]
      (apply ~name (aget array# idx#) idx2# idxv#)))))

(def-aset
  `{:doc "Sets the value at the index/indices.
  Works on arrays of int. Returns val."
   :added "1.0"}
  aset-int setInt int)

(def-aset
  `{:doc "Sets the value at the index/indices.
  Works on arrays of long. Returns val."
   :added "1.0"})

```

```

aset-long setLong long)

(def-aset
  ^{:doc "Sets the value at the index/indices.
  Works on arrays of boolean. Returns val."
  :added "1.0"}
  aset-boolean setBoolean boolean)

(def-aset
  ^{:doc "Sets the value at the index/indices.
  Works on arrays of float. Returns val."
  :added "1.0"}
  aset-float setFloat float)

(def-aset
  ^{:doc "Sets the value at the index/indices.
  Works on arrays of double. Returns val."
  :added "1.0"}
  aset-double setDouble double)

(def-aset
  ^{:doc "Sets the value at the index/indices.
  Works on arrays of short. Returns val."
  :added "1.0"}
  aset-short setShort short)

(def-aset
  ^{:doc "Sets the value at the index/indices.
  Works on arrays of byte. Returns val."
  :added "1.0"}
  aset-byte setByte byte)

(def-aset
  ^{:doc "Sets the value at the index/indices.
  Works on arrays of char. Returns val."
  :added "1.0"}
  aset-char setChar char)

(defn make-array
  "Creates and returns an array of instances of the specified class of
  the specified dimension(s). Note that a class object is required.
  Class objects can be obtained by using their imported or
  fully-qualified name. Class objects for the primitive types can be
  obtained using, e.g., Integer/TYPE."
  {:added "1.0"
   :static true}
  ([^Class type len]
   (. Array (newInstance type (int len))))
  ([^Class type dim & more-dims]
   (let [dims (cons dim more-dims)
        ...
        ]
    (if (= 1 (count dims))
        (make-array type (first dims))
        (make-array type (first dims) (rest dims)))))))

```

```

    ^" [I" dimarray (make-array (. Integer TYPE) (count dims))]
  (dotimes [i (alength dimarray)]
    (aset-int dimarray i (nth dims i)))
  (. Array (newInstance type dimarray)))))

(defn to-array-2d
  "Returns a (potentially-ragged) 2-dimensional array of Objects
  containing the contents of coll, which can be any Collection of any
  Collection."
  {:tag "[[Ljava.lang.Object;"
   :added "1.0"
   :static true}
  [^java.util.Collection coll]
  (let
   [ret
    (make-array
     (. Class (forName "[Ljava.lang.Object;"))
     (. coll (size))))
    (loop [i 0 xs (seq coll)]
      (when xs
        (aset ret i (to-array (first xs)))
        (recur (inc i) (next xs))))
    ret))

(defn macroexpand-1
  "If form represents a macro form, returns its expansion,
  else returns form."
  {:added "1.0"
   :static true}
  [form]
  (. clojure.lang.Compiler (macroexpand1 form)))

(defn macroexpand
  "Repeatedly calls macroexpand-1 on form until it no longer
  represents a macro form, then returns it. Note neither
  macroexpand-1 nor macroexpand expand macros in subforms."
  {:added "1.0"
   :static true}
  [form]
  (let [ex (macroexpand-1 form)]
    (if (identical? ex form)
        form
        (macroexpand ex)))))

(defn create-struct
  "Returns a structure basis object."
  {:added "1.0"
   :static true}
  [& keys]
  (. clojure.lang.PersistentStructMap (createSlotMap keys)))

```

```
(defmacro defstruct
  "Same as (def name (create-struct keys...))"
  {:added "1.0"
   :static true}
  [name & keys]
  '(def ~name (create-struct ~@keys)))

(defn struct-map
  "Returns a new structmap instance with the keys of the
  structure-basis. keyvals may contain all, some or none of the basis
  keys - where values are not supplied they will default to nil.
  keyvals can also contain keys not in the basis."
  {:added "1.0"
   :static true}
  [s & inits]
  (. clojure.lang.PersistentStructMap (create s inits)))

(defn struct
  "Returns a new structmap instance with the keys of the
  structure-basis. vals must be supplied for basis keys in order -
  where values are not supplied they will default to nil."
  {:added "1.0"
   :static true}
  [s & vals]
  (. clojure.lang.PersistentStructMap (construct s vals)))

(defn accessor
  "Returns a fn that, given an instance of a structmap with the basis,
  returns the value at the key. The key must be in the basis. The
  returned function should be (slightly) more efficient than using
  get, but such use of accessors should be limited to known
  performance-critical areas."
  {:added "1.0"
   :static true}
  [s key]
  (. clojure.lang.PersistentStructMap (getAccessor s key)))

(defn load-reader
  "Sequentially read and evaluate the set of forms contained in the
  stream/file"
  {:added "1.0"
   :static true}
  [rdr] (. clojure.lang.Compiler (load rdr)))

(defn load-string
  "Sequentially read and evaluate the set of forms contained in the
  string"
  {:added "1.0"
   :static true}
```

```
[s]
(let [rdr (-> (java.io.StringReader. s)
                 (clojure.lang.LineNumberingPushbackReader.))]
  (load-reader rdr)))

(defn set
  "Returns a set of the distinct elements of coll."
  {:added "1.0"
   :static true}
  [coll] (clojure.lang.PersistentHashSet/create (seq coll)))

(defn ^{:private true
         :static true}
  filter-key [keyfn pred amap]
  (loop [ret {} es (seq amap)]
    (if es
        (if (pred (keyfn (first es)))
            (recur (assoc ret (key (first es)) (val (first es))) (next es))
            (recur ret (next es)))
        ret)))

(defn find-ns
  "Returns the namespace named by the symbol or nil if it doesn't exist."
  {:added "1.0"
   :static true}
  [sym] (clojure.lang.Namespace/find sym))

(defn create-ns
  "Create a new namespace named by the symbol if one doesn't already
  exist, returns it or the already-existing namespace of the same
  name."
  {:added "1.0"
   :static true}
  [sym] (clojure.lang.Namespace/findOrCreate sym))

(defn remove-ns
  "Removes the namespace named by the symbol. Use with caution.
  Cannot be used to remove the clojure namespace."
  {:added "1.0"
   :static true}
  [sym] (clojure.lang.Namespace/remove sym))

(defn all-ns
  "Returns a sequence of all namespaces."
  {:added "1.0"
   :static true}
  [] (clojure.lang.Namespace/all))

(defn the-ns
  "If passed a namespace, returns it. Else, when passed a symbol,
```

```

returns the namespace named by it, throwing an exception if not
found."
{:added "1.0"
 :static true}
^clojure.lang.Namespace [x]
(if (instance? clojure.lang.Namespace x)
    x
    (or (find-ns x)
        (throw (Exception. (str "No namespace: " x " found")))))

(defn ns-name
  "Returns the name of the namespace, a symbol."
{:added "1.0"
 :static true}
[ns]
(.getName (the-ns ns)))

(defn ns-map
  "Returns a map of all the mappings for the namespace."
{:added "1.0"
 :static true}
[ns]
(.getMappings (the-ns ns)))

(defn ns-unmap
  "Removes the mappings for the symbol from the namespace."
{:added "1.0"
 :static true}
[ns sym]
(.unmap (the-ns ns) sym))

;(defn export [syms]
;  (doseq [sym syms]
;    (.. *ns* (intern sym) (setExported true)))))

(defn ns-publics
  "Returns a map of the public intern mappings for the namespace."
{:added "1.0"
 :static true}
[ns]
(let [ns (the-ns ns)]
  (filter-key val
    (fn [^clojure.lang.Var v] (and (instance? clojure.lang.Var v)
                                    (= ns (.ns v))
                                    (.isPublic v)))
    (ns-map ns)))))

(defn ns-imports
  "Returns a map of the import mappings for the namespace."
{:added "1.0"

```

```

:static true}
[ns]
(filter-key val (partial instance? Class) (ns-map ns)))

(defn ns-interns
  "Returns a map of the intern mappings for the namespace."
  {:added "1.0"
   :static true}
  [ns]
  (let [ns (the-ns ns)]
    (filter-key val
      (fn [^clojure.lang.Var v] (and (instance? clojure.lang.Var v)
                                       (= ns (.ns v))))
      (ns-map ns)))))

(defn refer
  "refers to all public vars of ns, subject to filters.
  filters can include at most one each of:
  :exclude list-of-symbols
  :only list-of-symbols
  :rename map-of-fromsymbol-tosymbol"
  {:added "1.0"}
  [ns-sym & filters]
  (let [ns (or (find-ns ns-sym)
               (throw (new Exception (str "No namespace: " ns-sym)))))
        fs (apply hash-map filters)
        nspublics (ns-publics ns)
        rename (or (:rename fs) {})
        exclude (set (:exclude fs))
        to-do (or (:only fs) (keys nspublics))]
    (doseq [sym to-do]
      (when-not (exclude sym)
        (let [v (nspublics sym)]
          (when-not v
            (throw (new java.lang.IllegalAccessError
                      (if (get (ns-interns ns) sym)
                          (str sym " is not public")
                          (str sym " does not exist")))))
          (. *ns* (refer (or (rename sym) sym) v)))))))

```

```
(defn ns-refers
  "Returns a map of the refer mappings for the namespace."
  {:added "1.0"
   :static true}
  [ns]
  (let [ns (the-ns ns)]
    (filter-key val
      (fn [^clojure.lang.Var v] (and (instance? clojure.lang.Var v)
                                       (not= ns (.ns v))))
      (ns-map ns)))))

(defn alias
  "Add an alias in the current namespace to another
  namespace. Arguments are two symbols: the alias to be used, and
  the symbolic name of the target namespace. Use :as in the ns macro
  in preference to calling this directly."
  {:added "1.0"
   :static true}
  [alias namespace-sym]
  (.addAlias *ns* alias (the-ns namespace-sym)))

(defn ns-aliases
  "Returns a map of the aliases for the namespace."
  {:added "1.0"
   :static true}
  [ns]
  (.getAliases (the-ns ns)))

(defn ns-unalias
  "Removes the alias for the symbol from the namespace."
  {:added "1.0"
   :static true}
  [ns sym]
  (.removeAlias (the-ns ns) sym))

(defn take-nth
  "Returns a lazy seq of every nth item in coll."
  {:added "1.0"
   :static true}
  [n coll]
  (lazy-seq
    (when-let [s (seq coll)]
      (cons (first s) (take-nth n (drop n s)))))

(defn interleave
  "Returns a lazy seq of the first item in each coll, then the
  second etc."
  {:added "1.0"
   :static true}
  ([c1 c2]
```

```

(lazy-seq
  (let [s1 (seq c1) s2 (seq c2)]
    (when (and s1 s2)
      (cons (first s1) (cons (first s2)
                               (interleave (rest s1) (rest s2)))))))
  ([c1 c2 & colls]
   (lazy-seq
     (let [ss (map seq (conj colls c2 c1))]
       (when (every? identity ss)
         (concat (map first ss) (apply interleave (map rest ss)))))))

(defn var-get
  "Gets the value in the var object"
  {:added "1.0"
   :static true}
  [^clojure.lang.Var x] (. x (get)))

(defn var-set
  "Sets the value in the var object to val. The var must be
thread-locally bound."
  {:added "1.0"
   :static true}
  [^clojure.lang.Var x val] (. x (set val)))

(defmacro with-local-vars
  "varbinding=> symbol init-expr

Executes the exprs in a context in which the symbols are bound to
vars with per-thread bindings to the init-exprs. The symbols refer
to the var objects themselves, and must be accessed with var-get and
var-set"
  {:added "1.0"}
  [name-vals-vec & body]
  (assert-args with-local-vars
    (vector? name-vals-vec) "a vector for its binding"
    (even? (count name-vals-vec))
    "an even number of forms in binding vector")
  '(let [~@(~(interleave
    (take-nth 2 name-vals-vec)
    (repeat '(.. clojure.lang.Var create setDynamic)))])
    (. clojure.lang.Var
      (pushThreadBindings (hash-map ~@name-vals-vec)))
    (try
      ~@body
      (finally (. clojure.lang.Var (popThreadBindings))))))
  )

(defn ns-resolve
  "Returns the var or Class to which a symbol will be resolved in the
namespace (unless found in the environment), else nil. Note that
if the symbol is fully qualified, the var/Class to which it resolves
"
  )

```

```

need not be present in the namespace."
{:added "1.0"
 :static true}
([ns sym]
 (ns-resolve ns nil sym))
([ns env sym]
 (when-not (contains? env sym)
 (clojure.lang.Compiler/maybeResolveIn (the-ns ns) sym)))))

(defn resolve
 "same as (ns-resolve *ns* symbol) or (ns-resolve *ns* &env symbol)"
{:added "1.0"
 :static true}
([sym] (ns-resolve *ns* sym))
([env sym] (ns-resolve *ns* env sym)))

(defn array-map
 "Constructs an array-map."
{:added "1.0"
 :static true}
([] (. clojure.lang.PersistentArrayMap EMPTY))
(& keyvals)
 (clojure.lang.PersistentArrayMap/createWithCheck
 (to-array keyvals)))))

(defn nthnext
 "Returns the nth next of coll, (seq coll) when n is 0."
{:added "1.0"
 :static true}
[coll n]
 (loop [n n xs (seq coll)]
 (if (and xs (pos? n))
 (recur (dec n) (next xs))
 xs)))

;redefine let and loop with destructuring
(defn destructure [bindings]
 (let [bents (partition 2 bindings)
 pb (fn pb [bvec b v]
 (let [pvec
 (fn [bvec b val]
 (let [gvec (gensym "vec__")]
 (loop [ret (-> bvec (conj gvec) (conj val))
 n 0
 bs b
 seen-rest? false]
 (if (seq bs)
 (let [firstb (first bs)]
 (cond

```

```

(= firstb '&)
(recur
  (pb ret
    (second bs)
    (list 'nthnext gvec n))
  n
  (nnext bs)
  true)
(= firstb :as) (pb ret (second bs) gvec)
:else (if seen-rest?
  (throw (new Exception
    "Unsupported binding form, only :as can follow & parameter")))
  (recur
    (pb ret firstb
      (list 'nth gvec n nil))
    (inc n)
    (next bs)
    seen-rest?)))
  ret))))
pmap
(fn [bvec b v]
  (let [gmap (or (:as b) (gensym "map__"))
        defaults (:or b)]
    (loop [ret (-> bvec (conj gmap) (conj v)
          (conj gmap)
          (conj
            '(if (seq? ~gmap)
              (apply hash-map ~gmap)
              ~gmap)))
        bes (reduce1
          (fn [bes entry]
            (reduce1
              #(assoc %1 %2 ((val entry) %2))
              (dissoc bes (key entry))
              ((key entry) bes)))
            (dissoc b :as :or)
            {:keys
              #(keyword (str %)),
              :strs str,
              :syms #(list 'quote %)})]
      (if (seq bes)
        (let [bb (key (first bes))
              bk (val (first bes))
              has-default (contains? defaults bb)]
          (recur (pb ret bb
            (if has-default
              (list 'get gmap bk (defaults bb))
              (list 'get gmap bk)))
            (next bes)))
        ret))))]

```

```

(cond
  (symbol? b) (-> bvec (conj b) (conj v))
  (vector? b) (pvec bvec b v)
  (map? b) (pmap bvec b v)
  :else
  (throw
    (new Exception
      (str "Unsupported binding form: " b))))))
  process-entry (fn [bvec b] (pb bvec (first b) (second b)))
(if (every? symbol? (map first bents))
  bindings
  (reduce1 process-entry [] bents)))

(defmacro let
  "binding => binding-form init-expr

Evaluates the exprs in a lexical context in which the symbols in
the binding-forms are bound to their respective init-exprs or parts
therein."
{:added "1.0", :special-form true,
  :forms '[(let [bindings*] exprs*)]}
[bindings & body]
(assert-args let
  (vector? bindings) "a vector for its binding"
  (even? (count bindings))
  "an even number of forms in binding vector")
'(let* ~(destructure bindings) ~@body))

(defn ^{:private true}
  maybe-destructured
  [params body]
  (if (every? symbol? params)
    (cons params body)
    (loop [params params
           new-params []
           lets []]
      (if params
        (if (symbol? (first params))
          (recur (next params) (conj new-params (first params)) lets)
          (let [gparam (gensym "p__")]
            (recur (next params) (conj new-params gparam)
                  (-> lets (conj (first params) (conj gparam))))))
        '(~new-params
          (let ~lets
            ~@body))))))

;redefine fn with destructuring and pre/post conditions
(defmacro fn
  "params => positional-params* , or positional-params* & next-param
  positional-param => binding-form

```

```

next-param => binding-form
name => symbol

Defines a function"
{:added "1.0", :special-form true,
:forms
'[(fn name? [params* ] exprs*) (fn name? ([params* ] exprs*+))]}
[& sigs]
(let [name (if (symbol? (first sigs)) (first sigs) nil)
      sigs (if name (next sigs) sigs)
      sigs (if (vector? (first sigs)) (list sigs) sigs)
      psig (fn* [sig]
             (let [[params & body] sig
                 conds (when (and (next body) (map? (first body)))
                           (first body))
                  body (if conds (next body) body)
                  conds (or conds (meta params))
                  pre (:pre conds)
                  post (:post conds)
                  body (if post
                           `((let [~'%' ~(if (< 1 (count body))
                                         `(do ~@body)
                                         (first body))])
                             ~@(map (fn* [c] `(assert ~c)) post)
                             ~'%'`))
                           body)
                  body (if pre
                           (concat (map (fn* [c] `(assert ~c)) pre)
                           body)
                           body)])
                  (maybe-destructured params body)))
                  new-sigs (map psig sigs)]
        (with-meta
          (if name
            (list* 'fn* name new-sigs)
            (cons 'fn* new-sigs))
          (meta &form)))))

(defmacro loop
"Evaluates the exprs in a lexical context in which the symbols in
the binding-forms are bound to their respective init-exprs or parts
therein. Acts as a recur target."
{:added "1.0", :special-form true, :forms '[(loop [bindings*] exprs*)]}
[bindings & body]
  (assert-args loop
    (vector? bindings) "a vector for its binding"
    (even? (count bindings))
    "an even number of forms in binding vector")
  (let [db (destructure bindings)]
    (if (= db bindings)

```

```

'(loop* ~bindings ~@body)
(let [vs (take-nth 2 (drop 1 bindings))
      bs (take-nth 2 bindings)
      gs (map (fn [b] (if (symbol? b) b (gensym))) bs)
      bfs (reduce1 (fn [ret [b v g]]
                     (if (symbol? b)
                         (conj ret g v)
                         (conj ret g v b g)))
                    [] (map vector bs vs gs))]
  '(let ~bfs
     (loop* ~(vec (interleave gs gs))
            (let ~(~(vec (interleave bs gs))
                  ~@body))))))

(defmacro when-first
  "bindings => x xs
   Same as (when (seq xs) (let [x (first xs)] body))"
  {:added "1.0"}
  [& bindings & body]
  (assert-args when-first
    (vector? bindings) "a vector for its binding"
    (= 2 (count bindings)) "exactly 2 forms in binding vector")
  (let [[x xs] bindings]
    '(when (seq ~xs)
       (let [~x (first ~xs)]
         ~@body)))

(defmacro lazy-cat
  "Expands to code which yields a lazy sequence of the concatenation
  of the supplied colls. Each coll expr is not evaluated until it is
  needed.

  (lazy-cat xs ys zs) ===
    (concat (lazy-seq xs) (lazy-seq ys) (lazy-seq zs))"
  {:added "1.0"}
  [& colls]
  '(concat ~@(map #(list 'lazy-seq %) colls)))

(defmacro for
  "List comprehension. Takes a vector of one or more
  binding-form/collection-expr pairs, each followed by zero or more
  modifiers, and yields a lazy sequence of evaluations of expr.
  Collections are iterated in a nested fashion, rightmost fastest,
  and nested coll-exprs can refer to bindings created in prior
  binding-forms. Supported modifiers are: :let [binding-form expr ...],
  :while test, :when test.

  (take 100
        (for [x (range 100000000) y (range 1000000) :while (< y x)]"

```

```

[x y))"
{:added "1.0"}
[seq-exprs body-expr]
(assert-args for
  (vector? seq-exprs) "a vector for its binding"
  (even? (count seq-exprs))
  "an even number of forms in binding vector")
(let [to-groups (fn [seq-exprs]
  (reduce1 (fn [groups [k v]]
    (if (keyword? k)
        (conj (pop groups)
          (conj (peek groups) [k v]))
        (conj groups [k v])))
    [] (partition 2 seq-exprs)))
  err (fn [& msg]
    (throw
      (IllegalArgumentException.
        ^String (apply str msg))))
  emit-bind (fn emit-bind [[bind expr & mod-pairs]
    & [_ next-expr] :as next-groups]]
  (let [giter (gensym "iter_")
    gxs (gensym "s_")
    do-mod
    (fn do-mod [[[k v :as pair] & etc]]
      (cond
        (= k :let)
        `(~(let ~v ~(do-mod etc)))
        (= k :while)
        `(~(when ~v ~(do-mod etc)))
        (= k :when) `(~(if ~v
          ~(do-mod etc)
          (recur (rest ~gxs))))
        (keyword? k)
        (err "Invalid 'for' keyword " k)
        next-groups
        `(~(let
          [ityers#
            ~(emit-bind next-groups)
            fs#
            (seq (ityers# ~next-expr))]
          (if fs#
              (concat fs#
                (~giter (rest ~gxs)))
              (recur (rest ~gxs))))
        :else `(~(cons
          ~body-expr
          (~giter (rest ~gxs))))))]
      (if next-groups
        #_"not the inner-most loop"
        `(~(fn ~giter [~gxs]
```

```

(lazy-seq
  (loop [~gxs ~gxs]
    (when-first [~bind ~gxs]
      ~(do-mod mod-pairs)))))

#_"inner-most loop"
(let [gi (gensym "i__")
  gb (gensym "b__")]
  do-cmod
  (fn do-cmod [[[k v :as pair] & etc]]
    (cond
      (= k :let)
      '(let ~v ~(do-cmod etc))
      (= k :while)
      '(when ~v ~(do-cmod etc))
      (= k :when)
      '(if ~v
        ~(do-cmod etc)
        (recur
          (unchecked-inc ~gi)))
      (keyword? k)
      (err
        "Invalid 'for' keyword "
        k)
      :else
      '(do
        (chunk-append
          ~gb ~body-expr)
        (recur
          (unchecked-inc ~gi))))))
  '(fn ~giter [~gxs]
    (lazy-seq
      (loop [~gxs ~gxs]
        (when-let [~gxs (seq ~gxs)]
          (if (chunked-seq? ~gxs)
            (let [c# (chunk-first ~gxs)
              size# (int (count c#))
              ~gb (chunk-buffer size#)]
              (if
                (loop [~gi (int 0)]
                  (if (< ~gi size#)
                    (let
                      [~bind (.nth c# ~gi)]
                      ~(do-cmod mod-pairs))
                    true))
                (chunk-cons
                  (chunk ~gb)
                  (~giter (chunk-rest ~gxs)))
                (chunk-cons (chunk ~gb) nil)))
            (let [~bind (first ~gxs)]
              ~(do-mod mod-pairs)))))))))))

```

```

'(let [iter# ~(emit-bind (to-groups seq-exprs))]
  (iter# ~(second seq-exprs)))))

(defmacro comment
  "Ignores body, yields nil"
  {:added "1.0"}
  [& body])

(defmacro with-out-str
  "Evaluates exprs in a context in which *out* is bound to a fresh
  StringWriter. Returns the string created by any nested printing
  calls."
  {:added "1.0"}
  [& body]
  '(let [s# (new java.io.StringWriter)]
    (binding [*out* s#]
      ~@body
      (str s#)))

(defmacro with-in-str
  "Evaluates body in a context in which *in* is bound to a fresh
  StringReader initialized with the string s."
  {:added "1.0"}
  [s & body]
  '(with-open
    [s# (-> (java.io.StringReader. ~s)
              clojure.lang.LineNumberingPushbackReader.)]
    (binding [*in* s#]
      ~@body)))

(defn pr-str
  "pr to a string, returning it"
  {:tag String
   :added "1.0"
   :static true}
  [& xs]
  (with-out-str
    (apply pr xs)))

(defn prn-str
  "prn to a string, returning it"
  {:tag String
   :added "1.0"
   :static true}
  [& xs]
  (with-out-str
    (apply prn xs)))

(defn print-str
  "print to a string, returning it"

```

```

{:tag String
 :added "1.0"
 :static true}
[& xs]
  (with-out-str
    (apply print xs)))

(defn println-str
  "println to a string, returning it"
  {:tag String
   :added "1.0"
   :static true}
  [& xs]
  (with-out-str
    (apply println xs)))

(defmacro assert
  "Evaluates expr and throws an exception if it does not evaluate to
logical true."
  {:added "1.0"}
  [x]
  (when *assert*
    `(when-not ~x
       (throw
         (new AssertionError (str "Assert failed: " (pr-str '~x)))))))

(defn test
  "test [v] finds fn at key :test in var metadata and calls it,
presuming failure will throw exception"
  {:added "1.0"}
  [v]
  (let [f (:test (meta v))]
    (if f
        (do (f) :ok)
        :no-test)))

(defn re-pattern
  "Returns an instance of java.util.regex.Pattern, for use, e.g. in
re-matcher."
  {:tag java.util.regex.Pattern
   :added "1.0"
   :static true}
  [s] (if (instance? java.util.regex.Pattern s)
        s
        (. java.util.regex.Pattern (compile s)))))

(defn re-matcher
  "Returns an instance of java.util.regex.Matcher, for use, e.g. in
re-find."
  {:tag java.util.regex.Matcher}

```

```

:added "1.0"
:static true}
[^java.util.regex.Pattern re s]
( . re (matcher s)))

(defn re-groups
  "Returns the groups from the most recent match/find. If there are no
  nested groups, returns a string of the entire match. If there are
  nested groups, returns a vector of the groups, the first element
  being the entire match."
{:added "1.0"
 :static true}
[^java.util.regex.Matcher m]
(let [gc ( . m (groupCount))]
  (if (zero? gc)
    ( . m (group))
    (loop [ret [] c 0]
      (if (<= c gc)
        (recur (conj ret ( . m (group c))) (inc c))
        ret)))))

(defn re-seq
  "Returns a lazy sequence of successive matches of pattern in string,
  using java.util.regex.Matcher.find(), each such match processed with
  re-groups."
{:added "1.0"
 :static true}
[^java.util.regex.Pattern re s]
(let [m (re-matcher re s)]
  ((fn step []
    (when ( . m (find))
      (cons (re-groups m) (lazy-seq (step)))))))

(defn re-matches
  "Returns the match, if any, of string to pattern, using
  java.util.regex.Matcher.matches(). Uses re-groups to return the
  groups."
{:added "1.0"
 :static true}
[^java.util.regex.Pattern re s]
(let [m (re-matcher re s)]
  (when ( . m (matches))
    (re-groups m)))))

(defn re-find
  "Returns the next regex match, if any, of string to pattern, using
  java.util.regex.Matcher.find(). Uses re-groups to return the
  groups."
{:added "1.0"

```

```

:static true}
([~java.util.regex.Matcher m]
(when (. m (find))
  (re-groups m)))
([~java.util.regex.Pattern re s]
(let [m (re-matcher re s)]
  (re-find m)))

(defn rand
  "Returns a random floating point number between 0 (inclusive) and
n (default 1) (exclusive)."
{:added "1.0"
 :static true}
([] (. Math (random)))
([n] (* n (rand)))))

(defn rand-int
  "Returns a random integer between 0 (inclusive) and n (exclusive)."
{:added "1.0"
 :static true}
[n] (int (rand n)))

(defmacro defn-
  "same as defn, yielding non-public def"
{:added "1.0"}
[name & decls]
(list* 'defn (with-meta name (assoc (meta name) :private true))
      decls))

(defn tree-seq
  "Returns a lazy sequence of the nodes in a tree, via a depth-first
walk. branch? must be a fn of one arg that returns true if passed
a node that can have children (but may not). children must be a
fn of one arg that returns a sequence of the children. Will only
be called on nodes for which branch? returns true. Root is the
root node of the tree."
{:added "1.0"
 :static true}
[branch? children root]
(let [walk (fn walk [node]
            (lazy-seq
              (cons node
                    (when (branch? node)
                      (mapcat walk (children node))))))]
  (walk root)))

(defn file-seq
  "A tree seq on java.io.Files"
{:added "1.0"
 :static true}

```

```

[dir]
  (tree-seq
    (fn [^java.io.File f] (. f (isDirectory)))
    (fn [^java.io.File d] (seq (. d (listFiles))))
    dir))

(defn xml-seq
  "A tree seq on the xml elements as per xml/parse"
  {:added "1.0"
   :static true}
  [root]
  (tree-seq
    (complement string?)
    (comp seq :content)
    root))

(defn special-symbol?
  "Returns true if s names a special form"
  {:added "1.0"
   :static true}
  [s]
  (contains? (. clojure.lang.Compiler specials) s))

(defn var?
  "Returns true if v is of type clojure.lang.Var"
  {:added "1.0"
   :static true}
  [v] (instance? clojure.lang.Var v))

(defn subs
  "Returns the substring of s beginning at start inclusive, and ending
  at end (defaults to length of string), exclusive."
  {:added "1.0"
   :static true}
  (^String [^String s start] (. s (substring start)))
  (^String [^String s start end] (. s (substring start end)))))

(defn max-key
  "Returns the x for which (k x), a number, is greatest."
  {:added "1.0"
   :static true}
  ([k x] x)
  ([k x y] (if (> (k x) (k y)) x y))
  ([k x y & more]
   (reduce1 #(max-key k %1 %2) (max-key k x y) more)))

(defn min-key
  "Returns the x for which (k x), a number, is least."
  {:added "1.0"
   :static true}

```

```

([k x] x)
([k x y] (if (< (k x) (k y)) x y))
([k x y & more]
 (reduce1 #(min-key k %1 %2) (min-key k x y) more)))

(defn distinct
  "Returns a lazy sequence of the elements of coll with
  duplicates removed"
  {:added "1.0"
   :static true}
  [coll]
  (let [step (fn step [xs seen]
              (lazy-seq
               ((fn [[f :as xs] seen]
                  (when-let [s (seq xs)]
                    (if (contains? seen f)
                        (recur (rest s) seen)
                        (cons f (step (rest s) (conj seen f)))))))
                 xs seen)))
        (step coll #{}))]

(defn replace
  "Given a map of replacement pairs and a vector/collection, returns a
  vector/seq with any elements = a key in smap replaced with the
  corresponding val in smap"
  {:added "1.0"
   :static true}
  [smap coll]
  (if (vector? coll)
      (reduce1 (fn [v i]
                 (if-let [e (find smap (nth v i))]
                     (assoc v i (val e))
                     v))
                coll (range (count coll)))
      (map #(if-let [e (find smap %)] (val e) %) coll)))

(defmacro dosync
  "Runs the exprs (in an implicit do) in a transaction that encompasses
  exprs and any nested calls. Starts a transaction if none is already
  running on this thread. Any uncaught exception will abort the
  transaction and flow out of dosync. The exprs may be run more than
  once, but any effects on Refs will be atomic."
  {:added "1.0"}
  [& exprs]
  '(sync nil ~@exprs))

(defmacro with-precision
  "Sets the precision and rounding mode to be used for

```

BigDecimal operations.

```

Usage: (with-precision 10 (/ 1M 3))
or:   (with-precision 10 :rounding HALF_DOWN (/ 1M 3))

The rounding mode is one of CEILING, FLOOR, HALF_UP, HALF_DOWN,
HALF_EVEN, UP, DOWN and UNNECESSARY; it defaults to HALF_UP."
{:added "1.0"}
[precision & exprs]
  (let [[body rm] (if (= (first exprs) :rounding)
                      [(next (next exprs))
                       `((. java.math.RoundingMode ~(second exprs)))]
                      [exprs nil])])
    `(binding [*math-context* (java.math.MathContext. ~precision ~@rm)]
      ~@body))

(defn mk-bound-fn
  {:private true}
  [^clojure.lang.Sorted sc test key]
  (fn [e]
    (test(.. sc comparator (compare (. sc entryKey e) key)) 0)))

(defn subseq
  "sc must be a sorted collection, test(s) one of <, <=, > or
  >=. Returns a seq of those entries with keys ek for
  which (test(.. sc comparator (compare ek key)) 0) is true"
  {:added "1.0"
   :static true}
  ([^clojure.lang.Sorted sc test key]
   (let [include (mk-bound-fn sc test key)]
     (if (#{> >=} test)
         (when-let [[e :as s] (. sc seqFrom key true)]
           (if (include e) s (next s)))
         (take-while include (. sc seq true))))
   ([^clojure.lang.Sorted sc start-test start-key end-test end-key]
    (when-let [[e :as s] (. sc seqFrom start-key true)]
      (take-while (mk-bound-fn sc end-test end-key)
                  (if ((mk-bound-fn sc start-test start-key) e) s (next s))))))

(defn rsubseq
  "sc must be a sorted collection, test(s) one of <, <=, > or
  >=. Returns a reverse seq of those entries with keys ek for
  which (test(.. sc comparator (compare ek key)) 0) is true"
  {:added "1.0"
   :static true}
  ([^clojure.lang.Sorted sc test key]
   (let [include (mk-bound-fn sc test key)]
     (if (#{< <=} test)
         (when-let [[e :as s] (. sc seqFrom key false)]
           (if (include e) s (next s)))
         (take-while include (. sc seq true))))))

```

```

(take-while include (. sc seq false)))))

(^[clojure.lang.Sorted sc start-test start-key end-test end-key]
(when-let [[e :as s] (. sc seqFrom end-key false)]
  (take-while (mk-bound-fn sc start-test start-key)
    (if ((mk-bound-fn sc end-test end-key) e) s (next s)))))

(defn repeatedly
  "Takes a function of no args, presumably with side effects, and
  returns an infinite (or length n if supplied) lazy sequence of calls
  to it"
  {:added "1.0"
   :static true}
  ([f] (lazy-seq (cons (f) (repeatedly f))))
  ([n f] (take n (repeatedly f)))))

(defn add-classpath
  "DEPRECATED

  Adds the url (String or URL object) to the classpath per
  URLClassLoader.addURL"
  {:added "1.0"
   :deprecated "1.1"}
  [url]
  (println "WARNING: add-classpath is deprecated")
  (closure.lang.RT/addURL url))

(defn hash
  "Returns the hash code of its argument"
  {:added "1.0"
   :static true}
  [x] (. clojure.lang.Util (hash x)))

(defn interpose
  "Returns a lazy seq of the elements of coll separated by sep"
  {:added "1.0"
   :static true}
  [sep coll] (drop 1 (interleave (repeat sep) coll)))

(defmacro definline
  "Experimental - like defmacro, except defines a named function whose
  body is the expansion, calls to which may be expanded inline as if
  it were a macro. Cannot be used with variadic (&) args."
  {:added "1.0"}
  [name & decl]
  (let [[pre-args [args expr]] (split-with (comp not vector?) decl)]
    `(do
      (defn ~name ~@pre-args ~args
        ~(apply (eval (list 'fn args expr)) args)))

```

```

(alter-meta! (var ~name) assoc :inline (fn ~name ~args ~expr))
  (var ~name)))

(defn empty
  "Returns an empty collection of the same category as coll, or nil"
  {:added "1.0"
   :static true}
  [coll]
  (when (instance? clojure.lang.IPersistentCollection coll)
    (.empty ^clojure.lang.IPersistentCollection coll)))

(defmacro amap
  "Maps an expression across an array a, using an index named idx, and
  return value named ret, initialized to a clone of a, then setting
  each element of ret to the evaluation of expr, returning the new
  array ret."
  {:added "1.0"}
  [a idx ret expr]
  '(let [a# ~a
         ^ret (aclone a#)]
     (loop [~idx 0]
       (if (< ~idx (alength a#))
           (do
             (aset ^ret ~idx ^expr)
             (recur (unchecked-inc ~idx)))
           ^ret)))))

(defmacro areduce
  "Reduces an expression across an array a, using an index named idx,
  and return value named ret, initialized to init, setting ret to the
  evaluation of expr at each step, returning ret."
  {:added "1.0"}
  [a idx ret init expr]
  '(let [a# ~a]
     (loop [~idx 0 ~ret ~init]
       (if (< ~idx (alength a#))
           (recur (unchecked-inc ~idx) ~expr)
           ~ret)))))

(defn float-array
  "Creates an array of floats"
  {:inline (fn [& args] '(. clojure.lang.Numbers float_array ^@args))
   :inline-arithies #{1 2}
   :added "1.0"}
  ([size-or-seq] (. clojure.lang.Numbers float_array size-or-seq))
  ([size init-val-or-seq]
   (. clojure.lang.Numbers float_array size init-val-or-seq)))

(defn boolean-array
  "Creates an array of booleans"

```

```

{:inline (fn [& args] ‘(. clojure.lang.Numbers boolean_array ~@args))
:inline-arities #{1 2}
:added "1.1"}
([size-or-seq] (. clojure.lang.Numbers boolean_array size-or-seq))
([size init-val-or-seq]
  (. clojure.lang.Numbers boolean_array size init-val-or-seq)))

(defn byte-array
  "Creates an array of bytes"
  {:inline (fn [& args] ‘(. clojure.lang.Numbers byte_array ~@args))
  :inline-arities #{1 2}
  :added "1.1"}
  ([size-or-seq] (. clojure.lang.Numbers byte_array size-or-seq))
  ([size init-val-or-seq]
    (. clojure.lang.Numbers byte_array size init-val-or-seq)))

(defn char-array
  "Creates an array of chars"
  {:inline (fn [& args] ‘(. clojure.lang.Numbers char_array ~@args))
  :inline-arities #{1 2}
  :added "1.1"}
  ([size-or-seq] (. clojure.lang.Numbers char_array size-or-seq))
  ([size init-val-or-seq]
    (. clojure.lang.Numbers char_array size init-val-or-seq)))

(defn short-array
  "Creates an array of shorts"
  {:inline (fn [& args] ‘(. clojure.lang.Numbers short_array ~@args))
  :inline-arities #{1 2}
  :added "1.1"}
  ([size-or-seq] (. clojure.lang.Numbers short_array size-or-seq))
  ([size init-val-or-seq]
    (. clojure.lang.Numbers short_array size init-val-or-seq)))

(defn double-array
  "Creates an array of doubles"
  {:inline (fn [& args] ‘(. clojure.lang.Numbers double_array ~@args))
  :inline-arities #{1 2}
  :added "1.0"}
  ([size-or-seq] (. clojure.lang.Numbers double_array size-or-seq))
  ([size init-val-or-seq]
    (. clojure.lang.Numbers double_array size init-val-or-seq)))

(defn object-array
  "Creates an array of objects"
  {:inline (fn [arg] ‘(. clojure.lang.RT object_array ~arg))
  :inline-arities #{1}
  :added "1.2"}
  ([size-or-seq] (. clojure.lang.RT object_array size-or-seq)))

```

```
(defn int-array
  "Creates an array of ints"
  {:inline (fn [& args] '(. clojure.lang.Numbers int_array ~@args))
   :inline-arities #{1 2}
   :added "1.0"}
  ([size-or-seq] (. clojure.lang.Numbers int_array size-or-seq))
  ([size init-val-or-seq]
   (. clojure.lang.Numbers int_array size init-val-or-seq)))

(defn long-array
  "Creates an array of longs"
  {:inline (fn [& args] '(. clojure.lang.Numbers long_array ~@args))
   :inline-arities #{1 2}
   :added "1.0"}
  ([size-or-seq] (. clojure.lang.Numbers long_array size-or-seq))
  ([size init-val-or-seq]
   (. clojure.lang.Numbers long_array size init-val-or-seq)))

(definline booleans
  "Casts to boolean[]"
  {:added "1.1"}
  [xs] '(. clojure.lang.Numbers booleans ~xs))

(definline bytes
  "Casts to bytes[]"
  {:added "1.1"}
  [xs] '(. clojure.lang.Numbers bytes ~xs))

(definline chars
  "Casts to chars[]"
  {:added "1.1"}
  [xs] '(. clojure.lang.Numbers chars ~xs))

(definline shorts
  "Casts to shorts[]"
  {:added "1.1"}
  [xs] '(. clojure.lang.Numbers shorts ~xs))

(definline floats
  "Casts to float[]"
  {:added "1.0"}
  [xs] '(. clojure.lang.Numbers floats ~xs))

(definline ints
  "Casts to int[]"
  {:added "1.0"}
  [xs] '(. clojure.lang.Numbers ints ~xs))

(definline doubles
  "Casts to double[]"
```

```

{:added "1.0"}
[xs] `(. clojure.lang.Numbers doubles ~xs))

(definline longs
"Casts to long[]"
{:added "1.0"}
[xs] `(. clojure.lang.Numbers longs ~xs))

(import '(java.util.concurrent BlockingQueue LinkedBlockingQueue))

(defn seque
  "Creates a queued seq on another (presumably lazy) seq s. The queued
  seq will produce a concrete seq in the background, and can get up to
  n items ahead of the consumer. n-or-q can be an integer n buffer
  size, or an instance of java.util.concurrent BlockingQueue. Note
  that reading from a seque can block if the reader gets ahead of the
  producer."
{:added "1.0"
 :static true}
([s] (seque 100 s))
([n-or-q s]
  (let [^BlockingQueue q (if (instance? BlockingQueue n-or-q)
                            n-or-q
                            (LinkedBlockingQueue. (int n-or-q)))
        NIL (Object.) ;nil sentinel since LBQ doesn't support nils
        agt (agent (seq s))
        fill (fn [s]
               (try
                 (loop [[x & xs :as s] s]
                   (if s
                       (if (.offer q (if (nil? x) NIL x))
                           (recur xs)
                           s)
                       (.put q q))) ; q itself is eos sentinel
                   (catch Exception e
                     (.put q q)
                     (throw e))))
               drain (fn drain []
                      (lazy-seq
                        (let [x (.take q)]
                          (if (identical? x q) ;q itself is eos sentinel
                              (do @agt nil) ;touch agent just to propagate errors
                              (do
                                (send-off agt fill)
                                (cons
                                  (if (identical? x NIL) nil x
                                      (drain)))))))
               (send-off agt fill)
               (drain)))))))

```

```

(defn class?
  "Returns true if x is an instance of Class"
  {:added "1.0"
   :static true}
  [x] (instance? Class x))

(defn- is-annotation? [c]
  (and (class? c)
       (.isAssignableFrom java.lang.annotation.Annotation c)))

(defn- is-runtime-annotation? [^Class c]
  (boolean
   (and (is-annotation? c)
        (when-let [^java.lang.annotation.Retention r
                  (.getAnnotation c java.lang.annotation.Retention)]
          (= (.value r) java.lang.annotation.RetentionPolicy/RUNTIME)))))

(defn- descriptor [^Class c] (clojure.asm.Type/getDescriptor c))

(declare process-annotation)
(defn- add-annotation [^clojure.asm.AnnotationVisitor av name v]
  (cond
    (vector? v) (let [avec (.visitArray av name)]
                  (doseq [vval v]
                    (add-annotation avec "value" vval))
                  (.visitEnd avec))
    (symbol? v) (let [ev (eval v)]
                  (cond
                    (instance? java.lang.Enum ev)
                    (.visitEnum av name (descriptor (class ev)) (str ev))
                    (class? ev)
                    (.visit av name (clojure.asm.Type/getType ev))
                    :else
                    (throw (IllegalArgumentException.
                            (str "Unsupported annotation value: " v
                                 " of class " (class ev)))))))
    (seq? v) (let [[nested nv] v
                   c (resolve nested)
                   nav (.visitAnnotation av name (descriptor c))]
               (process-annotation nav nv)
               (.visitEnd nav))
    :else (.visit av name v)))

(defn- process-annotation [av v]
  (if (map? v)
      (doseq [[k v] v]
        (add-annotation av (name k) v))
      (add-annotation av "value" v)))

(defn- add-annotations

```

```

([visitor m] (add-annotations visitor m nil))
([visitor m i]
 (doseq [[k v] m]
 (when (symbol? k)
 (when-let [c (resolve k)]
 (when (is-annotation? c)
 ;this is known duck/reflective as no common base
 ;of ASM Visitors
 (let
 [av (if i
 (.visitParameterAnnotation visitor i
 (descriptor c) (is-runtime-annotation? c)))
 (.visitAnnotation visitor
 (descriptor c) (is-runtime-annotation? c)))]
 (process-annotation av v)
 (.visitEnd av))))))

(defn alter-var-root
 "Atomically alters the root binding of var v by applying f to its
 current value plus any args"
 {:added "1.0"
 :static true}
 [^clojure.lang.Var v f & args] (.alterRoot v f args))

(defn bound?
 "Returns true if all of the vars provided as arguments have
 any bound value, root or thread-local. Implies that deref'ing
 the provided vars will succeed. Returns true if no vars are provided."
 {:added "1.2"
 :static true}
 [& vars]
 (every? #(.isBound ^clojure.lang.Var %) vars))

(defn thread-bound?
 "Returns true if all of the vars provided as arguments have
 thread-local bindings. Implies that set!'ing the provided vars
 will succeed. Returns true if no vars are provided."
 {:added "1.2"
 :static true}
 [& vars]
 (every? #(.getThreadBinding ^clojure.lang.Var %) vars))

(defn make-hierarchy
 "Creates a hierarchy object for use with derive, isa? etc."
 {:added "1.0"
 :static true}
 [] {:parents {} :descendants {} :ancestors {}})

(def ^{:private true}
 global-hierarchy (make-hierarchy))

```

```
(defn not-empty
  "If coll is empty, returns nil, else coll"
  {:added "1.0"
   :static true}
  [coll] (when (seq coll) coll))

(defn bases
  "Returns the immediate superclass and direct interfaces of c, if any"
  {:added "1.0"
   :static true}
  [^Class c]
  (when c
    (let [i (.getInterfaces c)
          s (.getSuperclass c)]
      (not-empty
       (if s (cons s i) i)))))

(defn supers
  "Returns the immediate and indirect superclasses and
  interfaces of c, if any"
  {:added "1.0"
   :static true}
  [^Class class]
  (loop [ret (set (bases class)) cs ret]
    (if (seq cs)
        (let [c (first cs) bs (bases c)]
          (recur (into1 ret bs) (into1 (disj cs c) bs)))
        (not-empty ret)))))

(defn isa?
  "Returns true if (= child parent), or child is directly or
  indirectly derived from parent, either via a Java type
  inheritance relationship or a relationship established via derive.
  h must be a hierarchy obtained from make-hierarchy, if not
  supplied defaults to the global hierarchy"
  {:added "1.0"}
  ([child parent] (isa? global-hierarchy child parent))
  ([h child parent]
   (or (= child parent)
       (and (class? parent) (class? child)
            (. ^Class parent isAssignableFrom child))
       (contains? ((:ancestors h) child) parent)
       (and (class? child)
            (some #(contains? ((:ancestors h) %) parent)
                  (supers child))))
       (and (vector? parent) (vector? child)
            (= (count parent) (count child))
            (loop [ret true i 0]
              (if (or (not ret) (= i (count parent)))
```

```

        ret
        (recur (isa? h (child i) (parent i)) (inc i)))))))

(defn parents
  "Returns the immediate parents of tag, either via a Java type
  inheritance relationship or a relationship established via derive. h
  must be a hierarchy obtained from make-hierarchy, if not supplied
  defaults to the global hierarchy"
  {:added "1.0"}
  ([tag] (parents global-hierarchy tag))
  ([h tag] (not-empty
    (let [tp (get (:parents h) tag)]
      (if (class? tag)
          (into1 (set (bases tag)) tp)
          tp)))))

(defn ancestors
  "Returns the immediate and indirect parents of tag, either via a
  Java type inheritance relationship or a relationship established
  via derive. h must be a hierarchy obtained from make-hierarchy,
  if not supplied defaults to the global hierarchy"
  {:added "1.0"}
  ([tag] (ancestors global-hierarchy tag))
  ([h tag] (not-empty
    (let [ta (get (:ancestors h) tag)]
      (if (class? tag)
          (let [superclasses (set (supers tag))]
            (reduce1 into1 superclasses
              (cons ta
                (map #(get (:ancestors h) %) superclasses))))
          ta)))))

(defn descendants
  "Returns the immediate and indirect children of tag, through a
  relationship established via derive. h must be a hierarchy obtained
  from make-hierarchy, if not supplied defaults to the global
  hierarchy. Note: does not work on Java type inheritance
  relationships."
  {:added "1.0"}
  ([tag] (descendants global-hierarchy tag))
  ([h tag] (if (class? tag)
    (throw (java.lang.UnsupportedOperationException.
      "Can't get descendants of classes"))
    (not-empty (get (:descendants h) tag)))))

(defn derive
  "Establishes a parent/child relationship between parent and
  tag. Parent must be a namespace-qualified symbol or keyword and
  child can be either a namespace-qualified symbol or keyword or a
  class. h must be a hierarchy obtained from make-hierarchy, if not
  "

```

```

supplied defaults to, and modifies, the global hierarchy."
{:added "1.0"}
([tag parent]
 (assert (namespace parent))
 (assert (or (class? tag)
            (and (instance? clojure.lang.Named tag) (namespace tag)))))

(alter-var-root #'global-hierarchy derive tag parent) nil)
([h tag parent]
 (assert (not= tag parent))
 (assert (or (class? tag) (instance? clojure.lang.Named tag)))
 (assert (instance? clojure.lang.Named parent)))

(let [tp (:parents h)
      td (:descendants h)
      ta (:ancestors h)
      tf (fn [m source sources target targets]
           (reduce1
             (fn [ret k]
               (assoc ret k
                     (reduce1 conj (get targets k #{}))
                     (cons target (targets target))))))
           m (cons source (sources source)))]
  (or
    (when-not (contains? (tp tag) parent)
      (when (contains? (ta tag) parent)
        (throw (Exception. (print-str tag
                                       "already has" parent "as ancestor")))))
    (when (contains? (ta parent) tag)
      (throw (Exception. (print-str "Cyclic derivation:"
                                     parent "has" tag "as ancestor"))))
    {:parents (assoc (:parents h) tag (conj (get tp tag #{}) parent))
     :ancestors (tf (:ancestors h) tag td parent ta)
     :descendants (tf (:descendants h) parent ta tag td)})
  h)))

(declare flatten)

(defn underive
  "Removes a parent/child relationship between parent and
  tag. h must be a hierarchy obtained from make-hierarchy, if not
  supplied defaults to, and modifies, the global hierarchy."
{:added "1.0"}
([tag parent]
 (alter-var-root #'global-hierarchy underive tag parent) nil)
([h tag parent]
 (let [parentMap (:parents h)
       childsParents (if (parentMap tag)
                      (disj (parentMap tag) parent) #{})
       newParents (if (not-empty childsParents)

```

```

        (assoc parentMap tag childParents)
        (dissoc parentMap tag))
deriv-seq (flatten
            (map #(cons (key %)
                        (interpose (key %) (val %)))
                  (seq newParents))))
(if (contains? (parentMap tag) parent)
(reduce1 #(apply derive %1 %2) (make-hierarchy
                                (partition 2 deriv-seq)
                                h))))
```



```

(defn distinct?
  "Returns true if no two of the arguments are ="
  {:tag Boolean
   :added "1.0"
   :static true}
  ([x] true)
  ([x y] (not (= x y)))
  ([x y & more]
   (if (not= x y)
       (loop [s #{x y} [x & etc :as xs] more]
         (if xs
             (if (contains? s x)
                 false
                 (recur (conj s x) etc))
             true))
       false)))
```



```

(defn resultset-seq
  "Creates and returns a lazy sequence of structmaps corresponding to
  the rows in the java.sql.ResultSet rs"
  {:added "1.0"}
  [^java.sql.ResultSet rs]
  (let [rsmeta (. rs getMetaData)]
    (idxs (range 1 (inc (. rsmeta getColumnCount))))
    keys (map (comp keyword #(.toLowerCase ^String %))
              (map (fn [i] (. rsmeta getColumnLabel i))) idxs))
    check-keys
      (or (apply distinct? keys)
          (throw (Exception.
                  "ResultSet must have unique column labels"))))
    row-struct (apply create-struct keys)
    row-values
      (fn [] (map (fn [^Integer i] (. rs getObject i)) idxs)))
    rows
      (fn thisfn []
        (when (. rs (next))
          (cons
            (apply struct row-struct (row-values))))
```

```

(lazy-seq (thisfn))))]
(rows)))

(defn iterator-seq
  "Returns a seq on a java.util.Iterator. Note that most collections
  providing iterators implement Iterable and thus support seq directly."
  {:added "1.0"
   :static true}
  [iter]
  (clojure.langIteratorSeq/create iter))

(defn enumeration-seq
  "Returns a seq on a java.util.Enumeration"
  {:added "1.0"
   :static true}
  [e]
  (clojure.langEnumerationSeq/create e))

(defn format
  "Formats a string using java.lang.String.format,
  see java.util.Formatter for format
  string syntax"
  {:added "1.0"
   :static true}
  ^String [fmt & args]
  (String/format fmt (to-array args)))

(defn printf
  "Prints formatted output, as per format"
  {:added "1.0"
   :static true}
  [fmt & args]
  (print (apply format fmt args)))

(declare gen-class)

(defmacro with-loading-context [& body]
  `((fn loading# []
      (. clojure.lang.Var
          (pushThreadBindings {clojure.lang.Compiler/LOADER
              (.getClassLoader (.getClass ^Object loading#)})))
      (try
          `@body
          (finally
              (. clojure.lang.Var (popThreadBindings)))))))

(defmacro ns
  "Sets *ns* to the namespace named by name (unevaluated), creating it
  if needed. references can be zero or more of: (:refer-clojure ...)
  (:require ...) (:use ...) (:import ...) (:load ...) (:gen-class)

```

with the syntax of refer-clojure/require/use/import/load/gen-class respectively, except the arguments are unevaluated and need not be quoted. (:gen-class ...), when supplied, defaults to :name corresponding to the ns name, :main true, :impl-ns same as ns, and :init-impl-ns true. All options of gen-class are supported. The :gen-class directive is ignored when not compiling. If :gen-class is not supplied, when compiled only an nsname_init.class will be generated. If :refer-clojure is not used, a default (refer 'clojure) is used. Use of ns is preferred to individual calls to in-ns/require/use/import:

```
(ns foo.bar
  (:refer-clojure :exclude [ancestors printf])
  (:require (clojure.contrib sql sql.tests))
  (:use (my.lib this that))
  (:import (java.util Date Timer Random)
           (java.sql Connection Statement)))
{:arglists '([name docstring? attr-map? references*])
 :added "1.0"}
[name & references]
(let [process-reference
      (fn [[kname & args]]
        `(~(symbol "clojure.core" (clojure.core/name kname))
          ~@(map #(list 'quote %) args)))
      docstring (when (string? (first references)) (first references))
      references (if docstring (next references) references)
      name (if docstring
              (vary-meta name assoc :doc docstring)
              name)
      metadata (when (map? (first references)) (first references))
      references (if metadata (next references) references)
      name (if metadata
              (vary-meta name merge metadata)
              name)
      gen-class-clause
      (first (filter # (= :gen-class (first %)) references))
      gen-class-call
      (when gen-class-clause
        (list* 'gen-class :name
               (.replace (str name) \- \_)
               :impl-ns name :main true (next gen-class-clause)))
      references (remove # (= :gen-class (first %)) references)
      ;ns-effect (clojure.core/in-ns name)
      ])
  `(do
    (clojure.core/in-ns `~name)
    (with-loading-context
      ~@(when gen-class-call (list gen-class-call))
      ~@(when (and (not= name 'clojure.core)
                    (not-any? # (= :refer-clojure (first %)) references)))
```

```

'((clojure.core/refer `clojure.core)))
~@(map process-reference references)))))

(defmacro refer-clojure
  "Same as (refer 'clojure.core <filters>)"
  {:added "1.0"}
  [& filters]
  '(clojure.core/refer `clojure.core ~@filters))

(defmacro defonce
  "defs name to have the root value of the expr iff the named var
  has no root value, else expr is unevaluated"
  {:added "1.0"}
  [name expr]
  '(let [v# (def ~name)]
    (when-not (.hasRoot v#)
      (def ~name ~expr)))))

;;;; require/use/load, contributed by Stephen C. Gilardi ;;;;;;;
(defonce ^:dynamic
  ^{:private true
    :doc "A ref to a sorted set of symbols representing loaded libs"}
  *loaded-libs* (ref (sorted-set)))

(defonce ^:dynamic
  ^{:private true
    :doc "the set of paths currently being loaded by this thread"}
  *pending-paths* #{})

(defonce ^:dynamic
  ^{:private true :doc
    "True while a verbose load is pending"}
  *loading-verbosely* false)

(defn- throw-if
  "Throws an exception with a message if pred is true"
  [pred fmt & args]
  (when pred
    (let [^String message (apply format fmt args)
          exception (Exception. message)
          raw-trace (.getStackTrace exception)
          boring?
            #(not= (.getMethodName ^StackTraceElement %) "doInvoke")
          trace (into-array (drop 2 (drop-while boring? raw-trace)))]
      (.setStackTrace exception trace)
      (throw exception)))))

(defn- libspec?
  "Returns true if x is a libspec"

```

```

[x]
(or (symbol? x)
    (and (vector? x)
        (or
            (nil? (second x))
            (keyword? (second x)))))

(defn- prependss
  "Prepends a symbol or a seq to coll"
  [x coll]
  (if (symbol? x)
      (cons x coll)
      (concat x coll)))

(defn- root-resource
  "Returns the root directory path for a lib"
  {:tag String}
  [lib]
  (str \/
    (.. (name lib)
        (replace \- \_)
        (replace \. \/)))))

(defn- root-directory
  "Returns the root resource path for a lib"
  [lib]
  (let [d (root-resource lib)]
    (subs d 0 (.lastIndexOf d "/")))

(declare load)

(defn- load-one
  "Loads a lib given its name. If need-ns, ensures that the associated
  namespace exists after loading. If require, records the load so any
  duplicate loads can be skipped."
  [lib need-ns require]
  (load (root-resource lib))
  (throw-if (and need-ns (not (find-ns lib)))
            "namespace '%s' not found after loading '%s'"
            lib (root-resource lib))
  (when require
    (dosync
      (commute *loaded-libs* conj lib)))))

(defn- load-all
  "Loads a lib given its name and forces a load of any libs it
  directly or indirectly loads. If need-ns, ensures that the
  associated namespace exists after loading. If require, records
  the load so any duplicate loads can be skipped."
  [lib need-ns require]
  
```

```

(dosync
  (commute *loaded-libs* #(reduce1 conj %1 %2)
    (binding [*loaded-libs* (ref (sorted-set))]
      (load-one lib need-ns require)
      @*loaded-libs*)))

(defn- load-lib
  "Loads a lib with options"
  [prefix lib & options]
  (throw-if (and prefix (pos? (.indexOf (name lib) (int \.))))
    "lib names inside prefix lists must not contain periods")
  (let [lib (if prefix (symbol (str prefix \. lib)) lib)
    opts (apply hash-map options)
    {:keys [as reload reload-all require use verbose]} opts
    loaded (contains? @*loaded-libs* lib)
    load (cond reload-all
      load-all
      (or reload (not require) (not loaded))
      load-one)
    need-ns (or as use)
    filter-opts (select-keys opts '(:exclude :only :rename))]
    (binding [*loading-verbosely* (or *loading-verbosely* verbose)]
      (if load
        (load lib need-ns require)
        (throw-if (and need-ns (not (find-ns lib)))
          "namespace '%s' not found" lib))
      (when (and need-ns *loading-verbosely*)
        (printf "(clojure.core/in-ns '%s)\n" (ns-name *ns*)))
      (when as
        (when *loading-verbosely*
          (printf "(clojure.core/alias '%s '%s)\n" as lib))
        (alias as lib))
      (when use
        (when *loading-verbosely*
          (printf "(clojure.core/refer '%s" lib)
          (doseq [opt filter-opts]
            (printf " %s '%s" (key opt) (print-str (val opt))))
          (printf ")\n"))
        (apply refer lib (mapcat seq filter-opts))))))

(defn- load-libs
  "Loads libs, interpreting libspecs, prefix lists, and flags for
  forwarding to load-lib"
  [& args]
  (let [flags (filter keyword? args)
    opts (interleave flags (repeat true))
    args (filter (complement keyword?) args)]
  ; check for unsupported options
  (let [supported #{:as :reload :reload-all :require :use :verbose}
    unsupported (seq (remove supported flags))])

```

```

(throw-if unsupported
  (apply str "Unsupported option(s) supplied: "
         (interpose \, unsupported)))
; check a load target was specified
(throw-if (not (seq args)) "Nothing specified to load")
(doseq [arg args]
  (if (libspec? arg)
    (apply load-lib nil (prependss arg opts))
    (let [[prefix & args] arg]
      (throw-if (nil? prefix) "prefix cannot be nil")
      (doseq [arg args]
        (apply load-lib prefix (prependss arg opts)))))))
;; Public

(defn require
  "Loads libs, skipping any that are already loaded. Each argument is
  either a libspec that identifies a lib, a prefix list that identifies
  multiple libs whose names share a common prefix, or a flag that
  modifies how all the identified libs are loaded. Use :require in
  the ns macro in preference to calling this directly."

```

Libs

A 'lib' is a named set of resources in classpath whose contents define a library of Clojure code. Lib names are symbols and each lib is associated with a Clojure namespace and a Java package that share its name. A lib's name also locates its root directory within classpath using Java's package name to classpath-relative path mapping. All resources in a lib should be contained in the directory structure under its root directory. All definitions a lib makes should be in its associated namespace.

'require loads a lib by loading its root resource. The root resource path is derived from the lib name in the following manner:
 Consider a lib named by the symbol 'x.y.z; it has the root directory <classpath>/x/y/, and its root resource is <classpath>/x/y/z.clj. The root resource should contain code to create the lib's namespace (usually by using the ns macro) and load any additional lib resources.

Libspecs

A libspec is a lib name or a vector containing a lib name followed by options expressed as sequential keywords and arguments.

Recognized options: :as
 :as takes a symbol as its argument and makes that symbol an alias to the lib's namespace in the current namespace.

Prefix Lists

It's common for Clojure code to depend on several libs whose names have the same prefix. When specifying libs, prefix lists can be used to reduce repetition. A prefix list contains the shared prefix followed by libspecs with the shared prefix removed from the lib names. After removing the prefix, the names that remain must not contain any periods.

Flags

A flag is a keyword.

```
Recognized flags: :reload, :reload-all, :verbose
:reload forces loading of all the identified libs even if they are
already loaded
:reload-all implies :reload and also forces loading of all libs that
the identified libs directly or indirectly load via require or use
:verbose triggers printing information about each load, alias, and
refer
```

Example:

The following would load the libraries clojure.zip and clojure.set abbreviated as 's'.

```
(require '(clojure zip [set :as s]))
{:added "1.0"}

[& args]
(apply load-libs :require args))

(defn use
  "Like 'require, but also refers to each lib's namespace using
clojure.core/refer. Use :use in the ns macro in preference to
calling this directly.

'use accepts additional options in libspecs: :exclude, :only,
:rename. The arguments and semantics for :exclude, :only, and
:rename are the same as those documented for clojure.core/refer."
{:added "1.0"}
[& args] (apply load-libs :require :use args))

(defn loaded-libs
  "Returns a sorted set of symbols naming the currently loaded libs"
{:added "1.0"}
[] @*loaded-libs*)

(defn load
  "Loads Clojure code from resources in classpath. A path is
interpreted as classpath-relative if it begins with a slash"
```

```

or relative to the root directory for the current namespace
otherwise."
{:added "1.0"}
[& paths]
(doseq [^String path paths]
  (let [^String path (if (.startsWith path "/")
                           path
                           (str (root-directory (ns-name *ns*)) \/ path))]
    (when *loading-verbosely*
      (printf "(clojure.core/load \"%s\")\n" path)
      (flush))
    ; (throw-if (*pending-paths* path)
    ;           "cannot load '%s' again while it is loading"
    ;           path)
    (when-not (*pending-paths* path)
      (binding [*pending-paths* (conj *pending-paths* path)]
        (clojure.lang.RT/load (.substring path 1))))))

(defn compile
  "Compiles the namespace named by the symbol lib into a set of
  classfiles. The source for the lib must be in a proper
  classpath-relative directory. The output files will go into the
  directory specified by *compile-path*, and that directory too must
  be in the classpath."
{:added "1.0"}
[lib]
(binding [*compile-files* true]
  (load-one lib true true))
lib)

;;;;;; nested associative ops ;;;;;;

(defn get-in
  "Returns the value in a nested associative structure,
  where ks is a sequence of keys. Returns nil if the key is not present,
  or the not-found value if supplied."
{:added "1.2"
 :static true}
([m ks]
 (reduce1 get m ks))
([m ks not-found]
 (loop [sentinel (Object.)
        m
        ks (seq ks)]
  (if ks
      (let [m (get m (first ks) sentinel)]
        (if (identical? sentinel m)
            not-found
            (recur sentinel m (next ks))))
      m))))
```

```
(defn assoc-in
  "Associates a value in a nested associative structure, where ks
  is a sequence of keys and v is the new value and returns a new
  nested structure. If any levels do not exist, hash-maps will be
  created."
  {:added "1.0"
   :static true}
  [m [k & ks] v]
  (if ks
    (assoc m k (assoc-in (get m k) ks v))
    (assoc m k v)))

(defn update-in
  "'Updates' a value in a nested associative structure, where ks is a
  sequence of keys and f is a function that will take the old value
  and any supplied args and return the new value, and returns a new
  nested structure. If any levels do not exist, hash-maps will be
  created."
  {:added "1.0"
   :static true}
  ([m [k & ks] f & args]
   (if ks
     (assoc m k (apply update-in (get m k) ks f args))
     (assoc m k (apply f (get m k) args)))))

(defn empty?
  "Returns true if coll has no items - same as (not (seq coll)).
  Please use the idiom (seq x) rather than (not (empty? x))!"
  {:added "1.0"
   :static true}
  [coll] (not (seq coll)))

(defn coll?
  "Returns true if x implements IPersistentCollection"
  {:added "1.0"
   :static true}
  [x] (instance? clojure.lang.IPersistentCollection x))

(defn list?
  "Returns true if x implements IPersistentList"
  {:added "1.0"
   :static true}
  [x] (instance? clojure.lang.IPersistentList x))

(defn set?
  "Returns true if x implements IPersistentSet"
  {:added "1.0"
   :static true}
```

```
[x] (instance? clojure.lang.IPersistentSet x))

(defn ifn?
  "Returns true if x implements IFn. Note that many data structures
  (e.g. sets and maps) implement IFn"
  {:added "1.0"
   :static true}
  [x] (instance? clojure.lang.IFn x))

(defn fn?
  "Returns true if x implements Fn, i.e. is an object created via fn."
  {:added "1.0"
   :static true}
  [x] (instance? clojure.lang.Fn x))

(defn associative?
  "Returns true if coll implements Associative"
  {:added "1.0"
   :static true}
  [coll] (instance? clojure.lang.Associative coll))

(defn sequential?
  "Returns true if coll implements Sequential"
  {:added "1.0"
   :static true}
  [coll] (instance? clojure.lang.Sequential coll))

(defn sorted?
  "Returns true if coll implements Sorted"
  {:added "1.0"
   :static true}
  [coll] (instance? clojure.lang.Sorted coll))

(defn counted?
  "Returns true if coll implements count in constant time"
  {:added "1.0"
   :static true}
  [coll] (instance? clojure.lang.Counted coll))

(defn reversible?
  "Returns true if coll implements Reversible"
  {:added "1.0"
   :static true}
  [coll] (instance? clojure.lang.Reversible coll))

(def ^:dynamic
  ^{:doc "bound in a repl thread to the most recent value printed"
    :added "1.0"}
  *1)
```

```
(def ^:dynamic
  {:doc "bound in a repl thread to the second most recent value printed"
   :added "1.0"}
  *2)

(def ^:dynamic
  {:doc "bound in a repl thread to the third most recent value printed"
   :added "1.0"}
  *3)

(def ^:dynamic
  {:doc "bound in a repl thread to the most recent exception caught
        by the repl"
   :added "1.0"}
  *e)

(defn trampoline
  "trampoline can be used to convert algorithms requiring mutual
  recursion without stack consumption. Calls f with supplied args, if
  any. If f returns a fn, calls that fn with no arguments, and
  continues to repeat, until the return value is not a fn, then
  returns that non-fn value. Note that if you want to return a fn as a
  final value, you must wrap it in some data structure and unpack it
  after trampoline returns."
  {:added "1.0"
   :static true}
  ([f]
   (let [ret (f)]
     (if (fn? ret)
         (recur ret)
         ret)))
  ([f & args]
   (trampoline #(apply f args)))))

(defn intern
  "Finds or creates a var named by the symbol name in the namespace
  ns (which can be a symbol or a namespace), setting its root binding
  to val if supplied. The namespace must exist. The var will adopt any
  metadata from the name symbol. Returns the var."
  {:added "1.0"
   :static true}
  ([ns ^clojure.lang.Symbol name]
   (let [v (clojure.lang.Var/intern (the-ns ns) name)]
     (when (meta name) (.setMeta v (meta name)))
     v))
  ([ns name val]
   (let [v (clojure.lang.Var/intern (the-ns ns) name val)]
     (when (meta name) (.setMeta v (meta name)))
     v)))
```

```
(defmacro while
  "Repeatedly executes body while test expression is true. Presumes
  some side-effect will cause test to become false/nil. Returns nil"
  {:added "1.0"}
  [test & body]
  '(loop []
    (when ~test
      ~@body
      (recur)))))

(defn memoize
  "Returns a memoized version of a referentially transparent function.
  The memoized version of the function keeps a cache of the mapping
  from arguments to results and, when calls with the same arguments
  are repeated often, has higher performance at the expense of higher
  memory use."
  {:added "1.0"
   :static true}
  [f]
  (let [mem (atom {})]
    (fn [& args]
      (if-let [e (find @mem args)]
          (val e)
          (let [ret (apply f args)]
            (swap! mem assoc args ret)
            ret)))))

(defmacro condp
  "Takes a binary predicate, an expression, and a set of clauses.
  Each clause can take the form of either:
  test-expr result-expr
  test-expr :>> result-fn

  Note :>> is an ordinary keyword.

  For each clause, (pred test-expr expr) is evaluated. If it returns
  logical true, the clause is a match. If a binary clause matches, the
  result-expr is returned, if a ternary clause matches, its result-fn,
  which must be a unary function, is called with the result of the
  predicate as its argument, the result of that call being the return
  value of condp. A single default expression can follow the clauses,
  and its value will be returned if no clause matches. If no default
  expression is provided and no clause matches, an
  IllegalArgumentException is thrown."
  {:added "1.0"}
  [pred expr & clauses]
```

```

(let [gpred (gensym "pred_")
      gexpr (gensym "expr_")
      emit (fn emit [pred expr args]
              (let [[[a b c :as clause] more]
                    (split-at (if (= ::>> (second args)) 3 2) args)
                    n (count clause)]
                  (cond
                    (= 0 n)
                    `'(throw (IllegalArgumentException.
                               (str "No matching clause: " ~expr))))
                    (= 1 n) a
                    (= 2 n) `'(if (~pred ~a ~expr)
                                 ~b
                                 ~(emit pred expr more))
                    :else `'(if-let [p# (~pred ~a ~expr)]
                               (~c p#)
                               ~(emit pred expr more)))))
              gres (gensym "res_")]
      `(let [~gpred ~pred
            ~gexpr ~expr]
         ~(emit gpred gexpr clauses)))))

;;;;;;; var documentation ;;;;;;;;

(alter-meta! #'*agent* assoc :added "1.0")
(alter-meta! #'in-ns assoc :added "1.0")
(alter-meta! #'load-file assoc :added "1.0")

(defmacro add-doc-and-meta {:private true} [name docstring meta]
  `'(alter-meta! (var ~name) merge (assoc ~meta :doc ~docstring)))"

(add-doc-and-meta *file*
  "The path of the file being evaluated, as a String.

Evaluates to nil when there is no file, eg. in the REPL."
  {:added "1.0"})

(add-doc-and-meta *command-line-args*
  "A sequence of the supplied command line arguments, or nil if
none were supplied"
  {:added "1.0"})

(add-doc-and-meta *warn-on-reflection*
  "When set to true, the compiler will emit warnings when reflection is
needed to resolve Java method calls or field accesses.

Defaults to false."
  {:added "1.0"})

(add-doc-and-meta *compile-path*

```

```
"Specifies the directory where 'compile' will write out .class
files. This directory must be in the classpath for 'compile' to
work.

Defaults to \"classes\""
{:added "1.0"})

(add-doc-and-meta *compile-files*
  "Set to true when compiling files, false otherwise."
  {:added "1.0"})

(add-doc-and-meta *unchecked-math*
  "While bound to true, compilations of +, -, *, inc, dec and the
  coercions will be done without overflow checks. Default: false."
  {:added "1.3"})

(add-doc-and-meta *ns*
  "A clojure.lang.Namespace object representing the current namespace."
  {:added "1.0"})

(add-doc-and-meta *in*
  "A java.io.Reader object representing standard input for read
operations.

Defaults to System/in, wrapped in a LineNumberingPushbackReader"
  {:added "1.0"})

(add-doc-and-meta *out*
  "A java.io.Writer object representing standard output for print
operations.

Defaults to System/out, wrapped in an OutputStreamWriter"
  {:added "1.0"})

(add-doc-and-meta *err*
  "A java.io.Writer object representing standard error for print
operations.

Defaults to System/err, wrapped in a PrintWriter"
  {:added "1.0"})

(add-doc-and-meta *flush-on-newline*
  "When set to true, output will be flushed whenever a newline is
printed.

Defaults to true."
  {:added "1.0"})

(add-doc-and-meta *print-meta*
  "If set to logical true, when printing an object, its metadata
```

```

will also be printed in a form that can be read back by the reader.

Defaults to false."
{:added "1.0"})

(add-doc-and-meta *print-dup*
  "When set to logical true, objects will be printed in a way that
preserves their type when read in later.

Defaults to false."
{:added "1.0"})

(add-doc-and-meta *print-readably*
  "When set to logical false, strings and characters will be printed
with non-alphanumeric characters converted to the appropriate
escape sequences.

Defaults to true"
{:added "1.0"})

(add-doc-and-meta *read-eval*
  "When set to logical false, the EvalReader (#=(...)) is disabled
in the read/load in the thread-local binding.
Example:
  (binding [*read-eval* false] (read-string `#=(eval (def x 3))))"

Defaults to true"
{:added "1.0"})

\getchunk{defn future?}

\getchunk{defn future-done?}

(defmacro letfn
  "fnspec ==> (fname [params*] exprs) or (fname ([params*] exprs)+)

Takes a vector of function specs and a body, and generates a set of
bindings of functions to their names. All of the names are available
in all of the definitions of the functions, as well as the body."
{:added "1.0", :forms '[(letfn [fnspecs*] exprs*)],
 :special-form true, :url nil}
[fnspecs & body]
'(letfn* ~(vec (interleave (map first fnspecs)
                           (map #(cons 'fn %) fnspecs)))
      ~@body))

;;;;; case ;;;;;;;
(defn- shift-mask [shift mask x]
  (-> x (bit-shift-right shift) (bit-and mask)))

```

```
(defn- min-hash
  "takes a collection of keys and returns [shift mask]"
  [keys]
  (let [hashes (map hash keys)
        cnt (count keys)]
    (when-not (apply distinct? hashes)
      (throw (IllegalArgumentException. "Hashes must be distinct")))
    (or (first
          (filter
            (fn [[s m]]
              (apply distinct? (map #(shift-mask s m %) hashes)))
            (for [mask (map #(dec (bit-shift-left 1 %)) (range 1 14))
                  shift (range 0 31)]
              [shift mask])))
      (throw (IllegalArgumentException.
              "No distinct mapping found")))))

(defmacro case
  "Takes an expression, and a set of clauses.

  Each clause can take the form of either:

  test-constant result-expr

  (test-constant1 ... test-constantN) result-expr

  The test-constants are not evaluated. They must be compile-time
  literals, and need not be quoted. If the expression is equal to a
  test-constant, the corresponding result-expr is returned. A single
  default expression can follow the clauses, and its value will be
  returned if no clause matches. If no default expression is provided
  and no clause matches, an IllegalArgumentException is thrown.

  Unlike cond and condp, case does a constant-time dispatch, the
  clauses are not considered sequentially. All manner of constant
  expressions are acceptable in case, including numbers, strings,
  symbols, keywords, and (Clojure) composites thereof. Note that since
  lists are used to group multiple constants that map to the same
  expression, a vector can be used to match a list if needed. The
  test-constants need not be all of the same type."
  {:added "1.2"}

  [e & clauses]
  (let [ge (with-meta (gensym) {:tag Object})
        default (if (odd? (count clauses))
                  (last clauses)
                  '(throw (IllegalArgumentException.
                           (str "No matching clause: " ~ge))))]
    cases (partition 2 clauses))
```

```

case-map (reduce1 (fn [m [test expr]]
  (if (seq? test)
    (into1 m (zipmap test (repeat expr)))
    (assoc m test expr)))
  {} cases)
  [shift mask] (if (seq case-map) (min-hash (keys case-map)) [0 0]))

hmap (reduce1 (fn [m [test expr :as te]]
  (assoc m (shift-mask shift mask (hash test)) te))
  (sorted-map) case-map)

'(let [~ge ~e]
  ~(condp = (count clauses)
    0 default
    1 default
    '(case* ~ge ~shift ~mask
      ~(key (first hmap)) ~(key (last hmap)) ~default ~hmap
      ~(every? keyword? (keys case-map)))))

;;;;;;;;
;; helper files ;;;;;;;
(alter-meta! (find-ns 'clojure.core) assoc
  :doc "Fundamental library of the Clojure language")
(load "core_proxy")
(load "core_print")
(load "genclass")
(load "core_deftype")
(load "core/protocols")
(load "gvec")

;; redefine reduce with internal-reduce
(defn reduce
  "f should be a function of 2 arguments. If val is not supplied,
  returns the result of applying f to the first 2 items in coll, then
  applying f to that result and the 3rd item, etc. If coll contains no
  items, f must accept no arguments as well, and reduce returns the
  result of calling f with no arguments. If coll has only 1 item, it
  is returned and f is not called. If val is supplied, returns the
  result of applying f to val and the first item in coll, then
  applying f to that result and the 2nd item, etc. If coll contains no
  items, returns val and f is not called."
  {:added "1.0"}
  ([f coll]
   (if-let [s (seq coll)]
     (reduce f (first s) (next s))
     (f)))
  ([f val coll]
   (let [s (seq coll)]
     (clojure.core.protocols/internal-reduce s f val)))))

(defn into
  "Returns a new coll consisting of to-coll with all of the items of"

```

```

from-coll conjoined."
{:added "1.0"
 :static true}
[to from]
(if (instance? clojure.lang.IEditableCollection to)
  (persistent! (reduce conj! (transient to) from))
  (reduce conj to from)))

(require '[clojure.java.io :as jio])

(defn- normalize-slurp-opts
  [opts]
  (if (string? (first opts))
    (do
      (println
        "WARNING: (slurp f enc) is deprecated, use (slurp f :encoding enc).")
      [:encoding (first opts)])
    opts))

(defn slurp
  "Opens a reader on f and reads all its contents, returning a string.
  See clojure.java.io/reader for a complete list of supported arguments."
  {:added "1.0"}
  ([f & opts]
   (let [opts (normalize-slurp-opts opts)
         sb (StringBuilder.)]
     (with-open [#^java.io.Reader r (apply jio/reader f opts)]
       (loop [c (.read r)]
         (if (neg? c)
           (str sb)
           (do
             (.append sb (char c))
             (recur (.read r))))))))
  ([f content & options]
   (with-open [#^java.io.Writer w (apply jio/writer f options)]
     (.write w (str content)))))

\getchunk{defn future-call}

\getchunk{defmacro future}

\getchunk{defn future-cancel}

\getchunk{defn future-cancelled?}

```

```
(defn pmap
  "Like map, except f is applied in parallel. Semi-lazy in that the
  parallel computation stays ahead of the consumption, but doesn't
  realize the entire result unless required. Only useful for
  computationally intensive functions where the time of f dominates
  the coordination overhead."
  {:added "1.0"
   :static true}
  ([f coll]
   (let [n (+ 2 (.. Runtime getRuntime availableProcessors))
         rets (map #(future (f %)) coll)
         step (fn step [[x & xs :as vs] fs]
                (lazy-seq
                  (if-let [s (seq fs)]
                      (cons (deref x) (step xs (rest s)))
                      (map deref vs))))
                (step rets (drop n rets))))
    ([f coll & colls]
     (let [step (fn step [cs]
                   (lazy-seq
                     (let [ss (map seq cs)]
                       (when (every? identity ss)
                         (cons (map first ss) (step (map rest ss))))))]
       (pmap #(apply f %) (step (cons coll colls)))))

  (defn pcalls
    "Executes the no-arg fns in parallel, returning a lazy sequence of
    their values"
    {:added "1.0"
     :static true}
    [& fns] (pmap #(%) fns))

  (defmacro pvalues
    "Returns a lazy sequence of the values of the exprs, which are
    evaluated in parallel"
    {:added "1.0"
     :static true}
    [& exprs]
    `(pcalls ~@(map #'(list 'fn [] %) exprs)))

;; clojure version number ;;;;
(let [version-stream (.getResourceAsStream (clojure.lang.RT/baseLoader)
                                         "clojure/version.properties")
      properties
      (doto (new java.util.Properties) (.load version-stream))
      prop (fn [k] (.getProperty properties (str "clojure.version." k)))
      clojure-version
      #:major (Integer/valueOf ^String (prop "major")))
```

```

:minor      (Integer/valueOf ^String (prop "minor"))
:incremental (Integer/valueOf ^String (prop "incremental"))
:qualifier   (prop "qualifier")]
(def ^:dynamic *clojure-version*
(if (not (= (prop "interim") "false"))
(clojure.lang.RT/assoc clojure-version :interim true)
clojure-version)))

(add-doc-and-meta *clojure-version*
"The version info for Clojure core, as a map containing :major :minor
:incremental and :qualifier keys. Feature releases may increment
:minor and/or :major, bugfix releases will increment :incremental.
Possible values of :qualifier include
  \"GA\", \"SNAPSHOT\", \"RC-x\" \"BETA-x\""
{:added "1.0"})

(defn
clojure-version
"Returns clojure version as a printable string."
{:added "1.0"}
[]
(str (:major *clojure-version*)
"."
(:minor *clojure-version*)
(when-let [i (:incremental *clojure-version*)]
(str "." i))
(when-let [q (:qualifier *clojure-version*)]
(when (pos? (count q)) (str "-" q)))
(when (:interim *clojure-version*)
"-SNAPSHOT")))

\getchunk{defn promise}

\getchunk{defn deliver}

(defn flatten
"Takes any nested combination of sequential things (lists, vectors,
etc.) and returns their contents as a single, flat sequence.
(flatten nil) returns nil."
{:added "1.2"
:static true}
[x]
(filter (complement sequential?)
(rest (tree-seq sequential? seq x))))}

(defn group-by
>Returns a map of the elements of coll keyed by the result of
f on each element. The value at each key will be a vector of the
corresponding elements, in the order they appeared in coll."
{:added "1.2"

```

```

:static true}
[f coll]
(persistent!
(reduce
  (fn [ret x]
    (let [k (f x)]
      (assoc! ret k (conj (get ret k []) x))))
  (transient {}) coll)))

(defn partition-by
  "Applies f to each value in coll, splitting it each time f returns
  a new value. Returns a lazy seq of partitions."
{:added "1.2"
:static true}
[f coll]
(lazy-seq
  (when-let [s (seq coll)]
    (let [fst (first s)
          fv (f fst)
          run (cons fst (take-while #(= fv (f %)) (rest s)))]
      (cons run (partition-by f (drop (count run) s))))))

(defn frequencies
  "Returns a map from distinct items in coll to the number of times
  they appear."
{:added "1.2"
:static true}
[coll]
(persistent!
(reduce (fn [counts x]
  (assoc! counts x (inc (get counts x 0))))
  (transient {}) coll)))

(defn reductions
  "Returns a lazy seq of the intermediate values of the reduction (as
  per reduce) of coll by f, starting with init."
{:added "1.2"}
([f coll]
 (lazy-seq
   (if-let [s (seq coll)]
     (reductions f (first s) (rest s))
     (list (f))))
 ([f init coll]
  (cons init
    (lazy-seq
      (when-let [s (seq coll)]
        (reductions f (f init (first s)) (rest s)))))))

(defn rand-nth
  "Return a random element of the (sequential) collection. Will have

```

```

the same performance characteristics as nth for the given
collection."
{:added "1.2"
 :static true}
[coll]
(nth coll (rand-int (count coll)))))

(defn partition-all
  "Returns a lazy sequence of lists like partition, but may include
partitions with fewer than n items at the end."
{:added "1.2"
 :static true}
([n coll]
 (partition-all n n coll))
([n step coll]
 (lazy-seq
  (when-let [s (seq coll)]
    (cons (take n s) (partition-all n step (drop step s))))))

(defn shuffle
  "Return a random permutation of coll"
{:added "1.2"
 :static true}
[^java.util.Collection coll]
(let [al (java.util.ArrayList. coll)]
  (java.util.Collections/shuffle al)
  (clojure.lang.RT/vector (.toArray al)))))

(defn map-indexed
  "Returns a lazy sequence consisting of the result of applying f to 0
and the first item of coll, followed by applying f to 1 and the second
item in coll, etc, until coll is exhausted. Thus function f should
accept 2 arguments, index and item."
{:added "1.2"
 :static true}
[f coll]
(letfn
 [(mapi [idx coll]
        (lazy-seq
         (when-let [s (seq coll)]
           (if (chunked-seq? s)
               (let [c (chunk-first s)
                     size (int (count c))
                     b (chunk-buffer size)]
                 (dotimes [i size]
                   (chunk-append b (f (+ idx i) (.nth c i))))
                   (chunk-cons (chunk b) (mapi (+ idx size) (chunk-rest s)))))
               (cons (f idx (first s)) (mapi (inc idx) (rest s))))))]
  (mapi 0 coll)))

```

```
(defn keep
  "Returns a lazy sequence of the non-nil results of (f item). Note,
  this means false return values will be included. f must be free of
  side-effects."
  {:added "1.2"
   :static true}
  ([f coll]
   (lazy-seq
    (when-let [s (seq coll)]
      (if (chunked-seq? s)
          (let [c (chunk-first s)
                size (count c)
                b (chunk-buffer size)]
            (dotimes [i size]
              (let [x (f (.nth c i))]
                (when-not (nil? x)
                  (chunk-append b x))))
            (chunk-cons (chunk b) (keep f (chunk-rest s)))))
          (let [x (f (first s))]
            (if (nil? x)
                (keep f (rest s))
                (cons x (keep f (rest s))))))))))
  (defn keep-indexed
    "Returns a lazy sequence of the non-nil results of (f index item).
    Note, this means false return values will be included. f must be
    free of side-effects."
    {:added "1.2"
     :static true}
    ([f coll]
     (letfn [(keepi [idx coll]
              (lazy-seq
               (when-let [s (seq coll)]
                 (if (chunked-seq? s)
                     (let [c (chunk-first s)
                           size (count c)
                           b (chunk-buffer size)]
                       (dotimes [i size]
                         (let [x (f (+ idx i) (.nth c i))]
                           (when-not (nil? x)
                             (chunk-append b x))))
                       (chunk-cons (chunk b)
                                   (keepi (+ idx size) (chunk-rest s)))))
                     (let [x (f idx (first s))]
                       (if (nil? x)
                           (keepi (inc idx) (rest s))
                           (cons x (keepi (inc idx) (rest s)))))))
                 (keepi 0 coll))))]
       (defn fnil
```

```

"takes a function f, and returns a function that calls f, replacing
a nil first argument to f with the supplied value x. Higher arity
versions can replace arguments in the second and third
positions (y, z). Note that the function f can take any number of
arguments, not just the one(s) being nil-patched."
{:added "1.2"
 :static true}
([f x]
 (fn
  ([a] (f (if (nil? a) x a)))
  ([a b] (f (if (nil? a) x a) b))
  ([a b c] (f (if (nil? a) x a) b c))
  ([a b c & ds] (apply f (if (nil? a) x a) b c ds))))
([f x y]
 (fn
  ([a b] (f (if (nil? a) x a) (if (nil? b) y b)))
  ([a b c] (f (if (nil? a) x a) (if (nil? b) y b) c))
  ([a b c & ds] (apply f (if (nil? a) x a) (if (nil? b) y b) c ds))))
([f x y z]
 (fn
  ([a b] (f (if (nil? a) x a) (if (nil? b) y b)))
  ([a b c] (f (if (nil? a) x a) (if (nil? b) y b) (if (nil? c) z c)))
  ([a b c & ds]
   (apply f (if (nil? a) x a)
          (if (nil? b) y b)
          (if (nil? c) z c) ds)))))

(defn- ^{:dynamic true} assert-valid-fdecl
"A good fdecl looks like(([a] ...) ([a b] ...)) near the end of defn."
[fdecl]
(if-let [bad-args (seq (remove #(vector? %) (map first fdecl)))]
  (throw (IllegalArgumentException. (str
    "Parameter declaration " (first bad-args)
    " should be a vector"))))

(defn with-redefs-fn
"Temporarily redefines Vars during a call to func. Each val of
binding-map will replace the root value of its key which must be
a Var. After func is called with no args, the root values of all
the Vars will be set back to their old values. These temporary
changes will be visible in all threads. Useful for mocking out
functions during testing."
{:added "1.3"}
[binding-map func]
(let [root-bind (fn [m]
  (doseq [[a-var a-val] m]
    (.bindRoot ^clojure.lang.Var a-var a-val)))
  old-vals (zipmap (keys binding-map)
    (map deref (keys binding-map)))]
  (try

```

```
(root-bind binding-map)
(func)
(finally
  (root-bind old-vals)))))

(defmacro with-redefs
  "binding => var-symbol temp-value-expr

Temporarily redefines Vars while executing the body. The
temp-value-exprs will be evaluated and each resulting value will
replace in parallel the root value of its Var. After the body is
executed, the root values of all the Vars will be set back to their
old values. These temporary changes will be visible in all threads.
Useful for mocking out functions during testing."
{:added "1.3"}
[bindings & body]
'(with-redefs-fn ~(zipmap (map #'(list 'var %) (take-nth 2 bindings))
                           (take-nth 2 (next bindings)))
                           (fn [] ~@body)))
```

11.2 protocols.clj

— protocols.clj —

```
\getchunk{Clojure Copyright}

(ns clojure.core.protocols)

(defprotocol InternalReduce
  "Protocol for concrete seq types that can reduce themselves
   faster than first/next recursion. Called by clojure.core/reduce."
  (internal-reduce [seq f start]))

(extend-protocol InternalReduce
  nil
  (internal-reduce
    [s f val]
    val)

  ;; handles vectors and ranges
  clojure.lang.IChunkedSeq
  (internal-reduce
    [s f val]
    (if-let [s (seq s)]
      (if (chunked-seq? s)
```

```

(recur (chunk-next s)
       f
       (.reduce (chunk-first s) f val))
(internal-reduce s f val)
val))

clojure.lang.StringSeq
(internal-reduce
 [str-seq f val]
(let [s (.s str-seq)]
(loop [i (.i str-seq)
      val val]
(if (< i (.length s))
  (recur (inc i) (f val (.charAt s i)))
  val)))

clojure.lang.ArraySeq
(internal-reduce
 [a-seq f val]
(let [^objects arr (.array a-seq)]
(loop [i (.index a-seq)
      val val]
(if (< i (alength arr))
  (recur (inc i) (f val (aget arr i)))
  val)))

java.lang.Object
(internal-reduce
 [s f val]
(loop [cls (class s)
      s s
      f f
      val val]
(if-let [s (seq s)]
;; roll over to faster implementation if underlying seq changes type
(if (identical? (class s) cls)
  (recur cls (next s) f (f val (first s)))
  (internal-reduce s f val))
  val)))

(def arr-impl
'(internal-reduce
 [a-seq f val]
(let [arr (.array a-seq)]
(loop [i (.index a-seq)
      val val]
(if (< i (alength arr))
  (recur (inc i) (f val (aget arr i)))
  val))))
```

```
(defn- emit-array-impls*
  [syms]
  (apply
   concat
   (map
    (fn [s]
      [(symbol (str "clojure.lang.ArraySeq$ArraySeq_" s))
       arr-impl])
    syms)))

(defmacro emit-array-impls
  [& syms]
  `(extend-protocol InternalReduce
    ~@(~(emit-array-impls* syms))))
```

—————

```
(emit-array-impls int long float double byte char boolean)
```

11.3 core`deftype.clj

— core`deftype.clj —

```
\getchunk{Clojure Copyright}

(in-ns 'clojure.core)

;;;;;;;;
(defn namespace-munge
  "Convert a Clojure namespace name to a legal Java package name."
  {:added "1.2"}
  [ns]
  (.replace (str ns) \- \_))

;for now, built on gen-interface
(defmacro definterface
  [name & sigs]
  (let [tag (fn [x] (or (:tag (meta x)) Object))
        psig (fn [[name [& args]]]
               (vector name
                      (vec (map tag args))
                      (tag name)
                      (map meta args)))
        cname (with-meta
                  (symbol (str (namespace-munge *ns*) "." name)))
```

```

        (meta name))]

'(let []
  (gen-interface :name ~cname :methods ~(vec (map psig sigs)))
  (import ~cname)))

;;;;;; reify/deftype ;;;;;;;;

(defn- parse-opts [s]
  (loop [opts {} [k v & rs :as s] s]
    (if (keyword? k)
        (recur (assoc opts k v) rs)
        [opts s])))

(defn- parse-impls [specs]
  (loop [ret {} s specs]
    (if (seq s)
        (recur (assoc ret (first s) (take-while seq? (next s)))
               (drop-while seq? (next s)))
        ret)))

(defn- parse-opts+specs [opts+specs]
  (let [[opts specs] (parse-opts opts+specs)
        impls (parse-impls specs)
        interfaces (-> (map #(if (var? (resolve %))
                                (:on (deref (resolve %)))
                                %)
                               (keys impls)))
                     set
                     (disj 'Object 'java.lang.Object)
                     vec)
        methods (map (fn [[name params & body]]
                      (cons name (maybe-destructured params body)))
                     (apply concat (vals impls))))]
    (when-let [bad-opts (seq (remove #{:no-print} (keys opts)))]
      (throw (IllegalArgumentException.
              (apply print-str "Unsupported option(s) -" bad-opts))))
    [interfaces methods opts]))

(defmacro reify
  "reify is a macro with the following structure:

(reify options* specs*)

Currently there are no options.

Each spec consists of the protocol or interface name followed by zero
or more method bodies:

protocol-or-interface-or-Object
(methodName [args+] body)*

```

Methods should be supplied for all methods of the desired protocol(s) and interface(s). You can also define overrides for methods of Object. Note that the first parameter must be supplied to correspond to the target object ('this' in Java parlance). Thus methods for interfaces will take one more argument than do the interface declarations. Note also that recur calls to the method head should *not* pass the target object, it will be supplied automatically and can not be substituted.

The return type can be indicated by a type hint on the method name, and arg types can be indicated by a type hint on arg names. If you leave out all hints, reify will try to match on same name/arity method in the protocol(s)/interface(s) - this is preferred. If you supply any hints at all, no inference is done, so all hints (or default of Object) must be correct, for both arguments and return type. If a method is overloaded in a protocol/interface, multiple independent method definitions must be supplied. If overloaded with same arity in an interface you must specify complete hints to disambiguate - a missing hint implies Object.

recur works to method heads The method bodies of reify are lexical closures, and can refer to the surrounding local scope:

```
(str (let [f \"foo\"]
         (reify Object
                 (toString [this] f)))
     == \"foo\"

     (seq (let [f \"foo\"]
             (reify clojure.lang.Seqable
                     (seq [this] (seq f))))
     == (\\f \\o \\o))
     {:added "1.2"}
     [& opts+specs]
     (let [[interfaces methods] (parse-optspecs+specs opts+specs)]
         (with-meta '(reify* ^interfaces ^@methods) (meta &form)))

(defn hash-combine [x y]
  (clojure.lang.Util/hashCombine x (clojure.lang.Util/hash y)))

(defn munge [s]
  ((if (symbol? s) symbol str) (clojure.lang.Compiler/munge (str s)))

(defn- imap-cons
  [^IPersistentMap this o]
  (cond
    (instance? java.util.Map$Entry o)
    (let [^java.util.Map$Entry pair o]
        (.assoc this (.getKey pair) (.getValue pair))))
```

```

(instance? clojure.lang.IPersistentVector o)
  (let [^clojure.lang.IPersistentVector vec o]
    (.assoc this (.nth vec 0) (.nth vec 1)))
  :else (loop [this this
               o o]
    (if (seq o)
        (let [^java.util.Map$Entry pair (first o)]
          (recur (.assoc this (.getKey pair) (.getValue pair))
                 (rest o)))
        this)))

(defn- emit-defrecord
  "Do not use this directly - use defrecord"
  {:added "1.2"}
  [tagname name fields interfaces methods]
  (let [tag (keyword (str *ns*) (str tagname))
        classname
        (with-meta (symbol (str (namespace-munge *ns*) "." name))
                   (meta name))
        interfaces (vec interfaces)
        interface-set (set (map resolve interfaces))
        methodname-set (set (map first methods))
        hinted-fields fields
        fields (vec (map #(with-meta % nil) fields))
        base-fields fields
        fields (conj fields '___meta '___extmap)]
    (when (some #{:volatile-mutable :unsynchronized-mutable}
                (mapcat (comp keys meta) hinted-fields))
      (throw (IllegalArgumentException.
              (str ":volatile-mutable or :unsynchronized-mutable not "
                   "supported for record fields"))))
    (let [gs (gensym)]
      (letfn
        [(eqhash [[i m]]
           [i
            (conj m
                  '(hashCode [this#]
                             (clojure.lang.APersistentMap/mapHash this#))
                  '(equals [this# ~gs]
                           (clojure.lang.APersistentMap/mapEquals this# ~gs))))])
        (iobj [[i m]]
           [(conj i 'clojure.lang.IObj)
            (conj m '(meta [this#] '___meta)
                  '(withMeta [this# ~gs]
                      (new ~tagname ~(replace {'___meta gs} fields))))])
        (ilookup [[i m]]
           [(conj i 'clojure.lang.ILookup 'clojure.lang.IKeywordLookup)
            (conj m '(valAt [this# k#] (.valAt this# k# nil))
                  '(valAt [this# k# else#]
                          (case k# ~(mapcat (fn [fld] [(keyword fld) fld])
```

```

        base-fields)
        (get ~'__extmap k# else#)))
'(getLookupThunk [this# k#]
  (let [~'gclass (class this#)]
    (case k#
      ~@(let [hinted-target
              (with-meta 'gtarget {:tag tagname})]
          (mapcat
            (fn [fld]
              [(keyword fld)
               '(reify clojure.lang.ILookupThunk
                  (get [~'thunk ~'gtarget]
                      (if
                        (identical?
                          (class ~'gtarget) ~'gclass)
                        (. ~hinted-target
                           ~(keyword fld))
                        ~'thunk))))])
            base-fields))
        nil))))))
(imap [[i m]]
  [(conj i 'clojure.lang.IPersistentMap)
   (conj m
     '(count [this#]
             (+ ~(count base-fields) (count ~'__extmap)))
     '(empty [this#]
             (throw (UnsupportedOperationException.
                     (str "Can't create empty: " ~(str classname)))))
     '(cons [this# e#] ((var imap-cons) this# e#))
     '(equiv [this# ~gs]
             (boolean
               (or (identical? this# ~gs)
                   (when (identical? (class this#) (class ~gs))
                     (let [~'gs ~(with-meta gs {:tag tagname})]
                       (and
                         ~@(map
                             (fn [fld]
                               '(= ~fld (. ~gs ~fld)))
                             base-fields)
                         (= ~'__extmap
                             (. ~gs ~'__extmap)))))))
     '(containsKey [this# k#]
                 (not (identical? this# (.valAt this# k# this#))))
     '(entryAt [this# k#]
                (let [v# (.valAt this# k# this#)]
                  (when-not (identical? this# v#)
                    (clojure.lang.MapEntry. k# v#))))
     '(seq [this#]
           (seq
             (concatat

```

```

[~@(map
  #(list 'new 'clojure.lang.MapEntry
    (keyword %) %) base-fields)
 ~'__extmap)))
'(assoc [this# k# ^gs]
  (condp identical? k#
    ~@ (mapcat
      (fn [fld]
        [(keyword fld)
         (list* 'new tagname
           (replace {fld gs} fields))])
      base-fields)
    (new ^tagname
      ~@ (remove #'__extmap) fields)
    (assoc ~'__extmap k# ^gs))))
'(without [this# k#]
  (if
    (contains? #{}{~@ (map keyword base-fields)} k#)
    (dissoc (with-meta (into {} this#) ^__meta) k#)
    (new ^tagname ~@ (remove #'__extmap) fields)
    (not-empty (dissoc ~'__extmap k#))))))
(ijavamap [[i m]]
  [(conj i 'java.util.Map 'java.io.Serializable)
   (conj m
     '(size [this#] (.count this#))
     '(isEmpty [this#] (= 0 (.count this#)))
     '(containsValue [this# v#]
       (boolean (some #{v#} (vals this#))))
     '(get [this# k#] (.valAt this# k#))
     '(put [this# k# v#]
       (throw (UnsupportedOperationException.)))
     '(remove [this# k#]
       (throw (UnsupportedOperationException.)))
     '(putAll [this# m#]
       (throw (UnsupportedOperationException.)))
     '(clear [this#]
       (throw (UnsupportedOperationException.)))
     '(keySet [this#] (set (keys this#)))
     '(values [this#] (vals this#))
     '(entrySet [this#] (set this#))))]
  ]
(let [[i m]
  (-> [interfaces methods] eqhash iobj ilookup imap ijavamap)]
  '(deftype* ^tagname ^classname
    ~(conj hinted-fields '^__meta '__extmap)
    :implements ~(vec i)
    `@m)))))

(defmacro defrecord
  "Alpha - subject to change

```

```
(defrecord name [fields*] options* specs*)
```

Currently there are no options.

Each spec consists of a protocol or interface name followed by zero or more method bodies:

```
protocol-or-interface-or-Object
(methodName [args*] body)*
```

Dynamically generates compiled bytecode for class with the given name, in a package with the same name as the current namespace, the given fields, and, optionally, methods for protocols and/or interfaces.

The class will have the (immutable) fields named by fields, which can have type hints. Protocols/interfaces and methods are optional. The only methods that can be supplied are those declared in the protocols/interfaces. Note that method bodies are not closures, the local environment includes only the named fields, and those fields can be accessed directly.

Method definitions take the form:

```
(methodName [args*] body)
```

The argument and return types can be hinted on the arg and methodName symbols. If not supplied, they will be inferred, so type hints should be reserved for disambiguation.

Methods should be supplied for all methods of the desired protocol(s) and interface(s). You can also define overrides for methods of Object. Note that a parameter must be supplied to correspond to the target object ('this' in Java parlance). Thus methods for interfaces will take one more argument than do the interface declarations. Note also that recur calls to the method head should *not* pass the target object, it will be supplied automatically and can not be substituted.

In the method bodies, the (unqualified) name can be used to name the class (for calls to new, instance? etc).

The class will have implementations of several (clojure.lang) interfaces generated automatically: IObj (metadata support) and IPersistentMap, and all of their superinterfaces.

In addition, defrecord will define type-and-value-based equality and hashCode.

When AOT compiling, generates compiled bytecode for a class with the given name (a symbol), prepends the current ns as the package, and writes the .class file to the *compile-path* directory.

Two constructors will be defined, one taking the designated fields followed by a metadata map (nil for none) and an extension field map (nil for none), and one taking only the fields (using nil for meta and extension fields)."

```
{:added "1.2"}
```

```
[name [& fields] & opts+specs]
(let [gname name
      [interfaces methods opts] (parse-optspec+spec opts+specs)
      classname (symbol (str (namespace-munge *ns*) "." gname))
      tag (keyword (str *ns*) (str name))
      hinted-fields fields
      fields (vec (map #(with-meta % nil) fields))]
  '(let []
    ~(emit-defrecord name gname
      (vec hinted-fields) (vec interfaces) methods)
    (defmethod print-method ~classname [o# w#]
      ((var print-defrecord) o# w#))
    (import ~classname)
    #_(defn ~name
       ([~@fields] (new ~classname ~@fields nil nil))
       ([~@fields meta# extmap#]
        (new ~classname ~@fields meta# extmap#))))))
(defn- print-defrecord [o ^Writer w]
  (print-meta o w)
  (.write w "#:")
  (.write w (.getName (class o)))
  (print-map
    o
    pr-on w))

(defn- emit-deftype*
  "Do not use this directly - use deftype"
  [tagname name fields interfaces methods]
  (let [classname
        (with-meta (symbol (str (namespace-munge *ns*) "." name))
                  (meta name))]

    '(deftype* ~tagname ~classname ~fields
       :implements ~interfaces
       ~@methods)))

(defmacro deftype
  "Alpha - subject to change

  (deftype name [fields*] options* specs*)
```

Currently there are no options.

Each spec consists of a protocol or interface name followed by zero or more method bodies:

```
protocol-or-interface-or-Object
(methodName [args*] body)*
```

Dynamically generates compiled bytecode for class with the given name, in a package with the same name as the current namespace, the given fields, and, optionally, methods for protocols and/or interfaces.

The class will have the (by default, immutable) fields named by fields, which can have type hints. Protocols/interfaces and methods are optional. The only methods that can be supplied are those declared in the protocols/interfaces. Note that method bodies are not closures, the local environment includes only the named fields, and those fields can be accessed directly. Fields can be qualified with the metadata :volatile-mutable true or :unsynchronized-mutable true, at which point (set! afield aval) will be supported in method bodies. Note well that mutable fields are extremely difficult to use correctly, and are present only to facilitate the building of higher level constructs, such as Clojure's reference types, in Clojure itself. They are for experts only - if the semantics and implications of :volatile-mutable or :unsynchronized-mutable are not immediately apparent to you, you should not be using them.

Method definitions take the form:

```
(methodname [args*] body)
```

The argument and return types can be hinted on the arg and methodname symbols. If not supplied, they will be inferred, so type hints should be reserved for disambiguation.

Methods should be supplied for all methods of the desired protocol(s) and interface(s). You can also define overrides for methods of Object. Note that a parameter must be supplied to correspond to the target object ('this' in Java parlance). Thus methods for interfaces will take one more argument than do the interface declarations. Note also that recur calls to the method head should *not* pass the target object, it will be supplied automatically and can not be substituted.

In the method bodies, the (unqualified) name can be used to name the class (for calls to new, instance? etc).

When AOT compiling, generates compiled bytecode for a class with the

given name (a symbol), prepends the current ns as the package, and writes the .class file to the *compile-path* directory.

One constructors will be defined, taking the designated fields."
 {:added "1.2"}

```
[name [& fields] & opts+specs]
(let [gname name
      [interfaces methods opts] (parse-optspec+opts+specs opts+specs)
      classname (symbol (str (namespace-munge *ns*) "." gname))
      tag (keyword (str *ns*) (str name))
      hinted-fields fields
      fields (vec (map #(with-meta % nil) fields))]
  `(let []
    ~(emit-deftype* name gname
      (vec hinted-fields) (vec interfaces) methods)
    (import ~classname))))
```

;;;;;; protocols ;;;;;;

```
(defn- expand-method-impl-cache [^clojure.lang.MethodImplCache cache c f]
  (let [cs
        (into1 {})
        (remove (fn [[c e]] (nil? e))
                (map vec (partition 2 (.table cache)))))

        cs (assoc cs c (clojure.lang.MethodImplCache$Entry. c f))
        [shift mask] (min-hash (keys cs))
        table (make-array Object (* 2 (inc mask)))
        table
        (reduce1 (fn [^objects t [c e]]
                  (let [i (* 2 (int (shift-mask shift mask (hash c))))]
                      (aset t i c)
                      (aset t (inc i) e)
                      t))
                  table cs)]
        (clojure.lang.MethodImplCache.
          (.protocol cache) (.methodk cache) shift mask table)))
```

```
(defn- super-chain [^Class c]
  (when c
    (cons c (super-chain (.getSuperclass c)))))

(defn- pref
  ([] nil)
  ([a] a)
  ([^Class a ^Class b]
    (if (.isAssignableFrom a b) b a)))
```

```

(defn find-protocol-impl [protocol x]
  (if (instance? (:on-interface protocol) x)
      x
      (let [c (class x)
            impl #(get (:impls protocol) %)]
        (or (impl c)
            (and c
                  (or
                    (first (remove nil? (map impl (butlast (super-chain c))))))
                  (when-let [t
                            (reduce1 pref
                                      (filter impl (disj (supers c) Object)))]
                    (impl t))
                  (impl Object)))))))

(defn find-protocol-method [protocol methodk x]
  (get (find-protocol-impl protocol x) methodk))

(defn- protocol?
  [maybe-p]
  (boolean (:on-interface maybe-p)))

(defn- implements? [protocol atype]
  (and atype (.isAssignableFrom ^Class (:on-interface protocol) atype)))

(defn extends?
  "Returns true if atype extends protocol"
  {:added "1.2"}
  [protocol atype]
  (boolean (or (implements? protocol atype)
               (get (:impls protocol) atype)))))

(defn extender
  "Returns a collection of the types explicitly extending protocol"
  {:added "1.2"}
  [protocol]
  (keys (:impls protocol)))

(defn satisfies?
  "Returns true if x satisfies the protocol"
  {:added "1.2"}
  [protocol x]
  (boolean (find-protocol-impl protocol x)))

(defn -cache-protocol-fn
  [^clojure.lang.AFunction pf x ^Class c ^clojure.lang.IFn interf]
  (let [cache  (.__methodImplCache pf)
        f (if (.instance? c x)
             interf
             interf
             (fn []
               (let [res (apply pf x)]
                 (if (= res interf)
                     interf
                     (do (cache! (.__methodImplCache pf) x res)
                         res))))))]
    f))

```

```

        (find-protocol-method (.protocol cache) (.methodk cache) x)])
(when-not f
  (throw (IllegalArgumentException.
  (str "No implementation of method: " (.methodk cache)
       " of protocol: " (:var (.protocol cache))
       " found for class: " (if (nil? x) "nil"
                                 (.getName (class x))))))
  (set! (.__methodImplCache pf)
        (expand-method-impl-cache cache (class x) f)))
  f))

(defn- emit-method-builder [on-interface method on-method arglists]
  (let [methodk (keyword method)
        gthis (with-meta (gensym) {:tag 'clojure.lang.AFunction})
        ginterf (gensym)]
    `(fn [cache#]
      (let [~ginterf
            (fn
              (fn
                `@(~(map
                     (fn [args]
                       (let [gargs (map #(gensym (str "gf__" % "__")) args)
                            target (first gargs)]
                         `([~@gargs]
                           (. ~@(with-meta target {:tag on-interface})
                               ~(or on-method method) `@(~(rest gargs)))))
                           arglists))
                     `clojure.lang.AFunction f#
                     (fn ~gthis
                       `@(~(map
                            (fn [args]
                              (let [gargs (map #(gensym (str "gf__" % "__")) args)
                                   target (first gargs)]
                                `([~@gargs]
                                  (let [cache# (.__methodImplCache ~gthis)
                                        ff# (.fnFor cache#)
                                        (clojure.lang.Util/classOf ~target)])
                                    (if ff#
                                        (ff# `@gargs)
                                        ((-cache-protocol-fn
                                          ~gthis ~target
                                          ~on-interface ~ginterf) `@gargs)))))
                            arglists)))
                     (set! (.__methodImplCache f#) cache#)
                     f#)))))

      (defn -reset-methods [protocol]
        (doseq [[`clojure.lang.Var v build] (:method-builders protocol)]
          (let [cache
                (clojure.lang.MethodImplCache. protocol (keyword (.sym v)))]
            (.bindRoot v (build cache)))))
```

```
(defn- assert-same-protocol [protocol-var method-syms]
  (doseq [m method-syms]
    (let [v (resolve m)
          p (:protocol (meta v))]
      (when (and v (bound? v) (not= protocol-var p))
        (binding [*out* *err*]
          (println "Warning: protocol" protocol-var "is overwriting"
                  (if p
                      (str "method " (.sym v) " of protocol " (.sym p))
                      (str "function " (.sym v)))))))))

(defn- emit-protocol [name opts+sigs]
  (let [iname
        (symbol (str (munge (namespace-munge *ns*)) "." (munge name)))
        [opts sigs]
        (loop [opts {:on (list 'quote iname) :on-interface iname}
               sigs opts+sigs]
          (condp #(%1 %2) (first sigs)
            string? (recur (assoc opts :doc (first sigs)) (next sigs))
            keyword?
            (recur (assoc opts (first sigs) (second sigs))
                   (nnext sigs))
            [opts sigs])
          sigs (reduce1
                 (fn [m s]
                   (let [name-meta (meta (first s))
                         mname (with-meta (first s) nil)
                         [arglists doc]
                         (loop [as [] rs (rest s)]
                           (if (vector? (first rs))
                               (recur (conj as (first rs)) (next rs))
                               [(seq as) (first rs)])))
                     (when (some #{0} (map count arglists))
                       (throw (IllegalArgumentException.
                               (str "Protocol fn: " mname
                                    " must take at least one arg")))))
                   (assoc m (keyword mname)
                          (merge name-meta
                                 {:name (vary-meta mname assoc
                                                   :doc doc :arglists arglists)
                                  :arglists arglists
                                  :doc doc})))
                 {} sigs)
          meths (mapcat
                 (fn [sig]
                   (let [m (munge (:name sig))]
                     (map #(vector m
                                   (vec (repeat (dec (count %))'Object))
                                   'Object)

```

```

          (:arglists sig)))
          (vals sigs)])
'(do
  (defonce ~name {})
  (gen-interface :name ~iname :methods ~meths)
  (alter-meta! (var ~name) assoc :doc ~(:doc opts))
  #'assert-same-protocol (var ~name) ~(map :name (vals sigs)))
  (alter-var-root (var ~name) merge
    (assoc ~opts
      :signatures ~signatures
      :var (var ~name)
      :method-map
      ~(and (:on opts)
        (apply hash-map
          (mapcat
            (fn [s]
              [(keyword (:name s))
               (keyword (or (:on s) (:name s)))])))
          (vals sigs))))
    :method-builders
    ~(apply hash-map
      (mapcat
        (fn [s]
          [~(intern *ns*
            (with-meta ~{:name s}
              (merge ~s {:protocol (var ~name)})))
          (emit-method-builder
            (:on-interface opts)
            (:name s)
            (:on s)
            (:arglists s)))
          (vals sigs)))))))
  (-reset-methods ~name)
  ~name)))

(defmacro defprotocol
  "A protocol is a named set of named methods and their signatures:
  (defprotocol AProtocolName

    ;optional doc string
    \"A doc string for AProtocol abstraction\"

    ;method signatures
    (bar [this a b] \"bar docs\")
    (baz [this a] [this a b c] [this a b c] \"baz docs\"))

  No implementations are provided. Docs can be specified for the
  protocol overall and for each method. The above yields a set of
  polymorphic functions and a protocol object. All are
  namespace-qualified by the ns enclosing the definition. The resulting

```

functions dispatch on the type of their first argument, which is required and corresponds to the implicit target object ('this' in Java parlance). defprotocol is dynamic, has no special compile-time effect, and defines no new types or classes. Implementations of the protocol methods can be provided using extend.

defprotocol will automatically generate a corresponding interface, with the same name as the protocol, i.e. given a protocol: my.ns/Protocol, an interface: my.ns.Protocol. The interface will have methods corresponding to the protocol functions, and the protocol will automatically work with instances of the interface.

Note that you should not use this interface with deftype or reify, as they support the protocol directly:

```
(defprotocol P
  (foo [this])
  (bar-me [this] [this y]))

(deftype Foo [a b c]
P
  (foo [this] a)
  (bar-me [this] b)
  (bar-me [this y] (+ c y)))

(bar-me (Foo. 1 2 3) 42)
=> 45

(foo
  (let [x 42]
    (reify P
      (foo [this] 17)
      (bar-me [this] x)
      (bar-me [this y] x))))
=> 17"
{:added "1.2"}
[name & opts+sigs]
(emit-protocol name opts+sigs))

(defn extend
  "Implementations of protocol methods can be provided using
  the extend construct:

  (extend AType
    AP
    {:foo an-existing-fn
     :bar (fn [a b] ...)
     :baz (fn ([a]...) ([a b] ...)...)}
    BP
    {...})
```

...)

extend takes a type/class (or interface, see below), and one or more protocol + method map pairs. It will extend the polymorphism of the protocol's methods to call the supplied methods when an AType is provided as the first argument.

Method maps are maps of the keyword-ized method names to ordinary fns. This facilitates easy reuse of existing fns and fn maps, for code reuse/mixins without derivation or composition. You can extend an interface to a protocol. This is primarily to facilitate interop with the host (e.g. Java) but opens the door to incidental multiple inheritance of implementation since a class can inherit from more than one interface, both of which extend the protocol. It is TBD how to specify which impl to use. You can extend a protocol on nil.

If you are supplying the definitions explicitly (i.e. not reusing existing functions or mixin maps), you may find it more convenient to use the extend-type or extend-protocol macros.

Note that multiple independent extend clauses can exist for the same type, not all protocols need be defined in a single extend call.

See also:

```
extends?, satisfies?, extenders"
{:added "1.2"}
[atype & proto+mmmaps]
(doseq [[proto mmap] (partition 2 proto+mmmaps)]
  (when-not (protocol? proto)
    (throw (IllegalArgumentException.
            (str proto " is not a protocol"))))
  (when (implements? proto atype)
    (throw (IllegalArgumentException.
            (str atype " already directly implements "
                  (:on-interface proto) " for protocol:"
                  (:var proto)))))
  (-reset-methods
    (alter-var-root (:var proto) assoc-in [:impls atype] mmap)))
(defn- emit-impl [[p fs]]
  [p (zipmap (map #(.-> % first keyword) fs)
              (map #(cons 'fn (drop 1 %)) fs))])
(defn- emit-hinted-impl [c [p fs]]
  (let [hint (fn [specs]
               (let [specs (if (vector? (first specs))
                             (list specs)
                             specs)]
                 (map (fn [[[target & args] & body]]
                        (cons
```

```

          (apply vector
                  (vary-meta target assoc :tag c) args)
                  body))
          specs))))]
[p (zipmap (map #(-> % first keyword) fs)
            (map #(cons 'fn (hint (drop 1 %))) fs)))])

(defn- emit-extend-type [c specs]
  (let [impls (parse-impls specs)]
    `(~(extend ~c
               ~@(~(mapcat (partial emit-hinted-impl c) impls)))))

(defmacro extend-type
  "A macro that expands into an extend call. Useful when you are
  supplying the definitions explicitly inline, extend-type
  automatically creates the maps required by extend. Propagates the
  class as a type hint on the first argument of all fns.

  (extend-type MyType
    Countable
    (cnt [c] ...)
    Foo
    (bar [x y] ...)
    (baz ([x] ...) ([x y & zs] ...)))

expands into:

  (extend MyType
    Countable
    {:cnt (fn [c] ...)}
    Foo
    {:baz (fn ([x] ...) ([x y & zs] ...))
     :bar (fn [x y] ...)})
    {:added "1.2"}
    [t & specs]
    (emit-extend-type t specs))

(defn- emit-extend-protocol [p specs]
  (let [impls (parse-impls specs)]
    `(~(do
         ~@(~(map (fn [[t fs]]
                     `(~(extend-type ~t ~p ~@fs)
                       impls)))))

(defmacro extend-protocol
  "Useful when you want to provide several implementations of the same
  protocol all at once. Takes a single protocol and the implementation
  of that protocol for one or more types. Expands into calls to
  extend-type:

```

```
(extend-protocol Protocol
  AType
    (foo [x] ...)
    (bar [x y] ...)
  BType
    (foo [x] ...)
    (bar [x y] ...)
  AClass
    (foo [x] ...)
    (bar [x y] ...)
  nil
    (foo [x] ...)
    (bar [x y] ...))

expands into:

(do
  (clojure.core/extend-type AType Protocol
    (foo [x] ...)
    (bar [x y] ...))
  (clojure.core/extend-type BType Protocol
    (foo [x] ...)
    (bar [x y] ...))
  (clojure.core/extend-type AClass Protocol
    (foo [x] ...)
    (bar [x y] ...))
  (clojure.core/extend-type nil Protocol
    (foo [x] ...)
    (bar [x y] ...)))
{:added "1.2"})

[p & specs]
(emit-extend-protocol p specs))
```



11.4 core print.clj

— core print.clj —

```
\getchunk{Clojure Copyright}
(in-ns 'clojure.core)
;;;;;;;;
; printing ;;;;;;;
;
```

```
(import '(java.io Writer))

(def ^:dynamic
  {:doc "*print-length* controls how many items of each collection
the printer will print. If it is bound to logical false, there is
no limit. Otherwise, it must be bound to an integer indicating the
maximum number of items of each collection to print. If a collection
contains more items, the printer will print items up to the limit
followed by '...' to represent the remaining items. The root binding
is nil indicating no limit."
  :added "1.0"}
*print-length* nil)

(def ^:dynamic
  {:doc "*print-level* controls how many levels deep the printer will
print nested objects. If it is bound to logical false, there is no
limit. Otherwise, it must be bound to an integer indicating the
maximum level to print. Each argument to print is at level 0; if
an argument is a collection, its items are at level 1; and so on.
If an object is a collection and is at a level greater than or equal
to the value bound to *print-level*, the printer prints #' to
represent it. The root binding is nil indicating no limit."
  :added "1.0"}
*print-level* nil)

(defn- print-sequential [^String begin, print-one,
                        ^String sep, ^String end, sequence,
                        ^Writer w]
  (binding [*print-level* (and (not *print-dup*)
                               *print-level*
                               (dec *print-level*))]
    (if (and *print-level* (neg? *print-level*))
        (.write w "#")
        (do
          (.write w begin)
          (when-let [xs (seq sequence)]
            (if (and (not *print-dup*) *print-length*)
                (loop [[x & xs] xs
                      print-length *print-length*]
                  (if (zero? print-length)
                      (.write w "..."))
                  (do
                    (print-one x w)
                    (when xs
                      (.write w sep)
                      (recur xs (dec print-length))))))
                (loop [[x & xs] xs]
                  (print-one x w)
                  (when xs
                    (.write w sep)))))))
```

```

          (recur xs))))))
(.write w end)))))

(defn- print-meta [o, ^Writer w]
  (when-let [m (meta o)]
    (when (and (pos? (count m))
               (or *print-dup*
                   (and *print-meta* *print-readably*)))
            (.write w ""))
      (if (and (= (count m) 1) (:tag m))
          (pr-on (:tag m) w)
          (pr-on m w))
          (.write w " ")))

(defmethod print-method :default [o, ^Writer w]
  (print-method (vary-meta o #(dissoc % :type)) w))

(defmethod print-method nil [o, ^Writer w]
  (.write w "nil"))

(defmethod print-dup nil [o w] (print-method o w))

(defn print-ctor [o print-args ^Writer w]
  (.write w "#(")
  (.write w (.getName ^Class (class o)))
  (.write w ". ")
  (print-args o w)
  (.write w ")"))

(defmethod print-method Object [o, ^Writer w]
  (.write w "#<")
  (.write w (.get SimpleName (class o)))
  (.write w " ")
  (.write w (str o))
  (.write w ">"))

(defmethod print-method clojure.lang.Keyword [o, ^Writer w]
  (.write w (str o)))

(defmethod print-dup clojure.lang.Keyword [o w] (print-method o w))

(defmethod print-method Number [o, ^Writer w]
  (.write w (str o)))

(defmethod print-dup Number [o, ^Writer w]
  (print-ctor o
              (fn [o w]
                (print-dup (str o) w)
                w)))

```

```
(defmethod print-dup clojure.lang.Fn [o, ^Writer w]
  (print-ctor o (fn [o w]) w))

(prefer-method print-dup
  clojure.lang.IPersistentCollection clojure.lang.Fn)
(prefer-method print-dup java.util.Map clojure.lang.Fn)
(prefer-method print-dup java.util.Collection clojure.lang.Fn)

(defmethod print-method Boolean [o, ^Writer w]
  (.write w (str o)))

(defmethod print-dup Boolean [o w] (print-method o w))

(defn print-simple [o, ^Writer w]
  (print-meta o w)
  (.write w (str o)))

(defmethod print-method clojure.lang.Symbol [o, ^Writer w]
  (print-simple o w))

(defmethod print-dup clojure.lang.Symbol [o w] (print-method o w))

(defmethod print-method clojure.lang.Var [o, ^Writer w]
  (print-simple o w))

(defmethod print-dup clojure.lang.Var [^clojure.lang.Var o, ^Writer w]
  (.write w (str "#=(var " (.name (.ns o)) "/" (.sym o) ")")))

(defmethod print-method clojure.lang.ISeq [o, ^Writer w]
  (print-meta o w)
  (print-sequential "(" pr-on " " ")" o w))

(defmethod print-dup clojure.lang.ISeq [o w] (print-method o w))
(defmethod print-dup
  clojure.lang.IPersistentList [o w] (print-method o w))
(prefer-method print-method
  clojure.lang.ISeq clojure.lang.IPersistentCollection)
(prefer-method print-dup
  clojure.lang.ISeq clojure.lang.IPersistentCollection)
(prefer-method print-method clojure.lang.ISeq java.util.Collection)
(prefer-method print-dup clojure.lang.ISeq java.util.Collection)

(defmethod print-dup java.util.Collection [o, ^Writer w]
  (print-ctor o #(print-sequential "[" print-dup " " "]" %1 %2) w))

(defmethod print-dup clojure.lang.IPersistentCollection [o, ^Writer w]
  (print-meta o w)
  (.write w "#(")
```

```

(.write w (.getName ^Class (class o)))
(.write w "/create")
(print-sequential "[" print-dup " " "]" o w)
(.write w ")"))

(prefer-method print-dup
  clojure.lang.IPersistentCollection java.util.Collection)

(def ^{:tag String
       :doc "Returns escape string for char or nil if none"
       :added "1.0"}
  char-escape-string
  {\newline "\n"
   \tab "\t"
   \return "\r"
   \" "\\"
   \\ "\\\\
   \formfeed "\f"
   \backspace "\b"})
}

(defmethod print-method String [^String s, ^Writer w]
  (if (or *print-dup* *print-readably*)
    (do (.append w \")
        (dotimes [n (count s)]
          (let [c (.charAt s n)
                e (char-escape-string c)]
            (if e (.write w e) (.append w c))))
        (.append w \"))
        (.write w s))
    nil))

(defmethod print-dup String [s w] (print-method s w))

(defmethod print-method clojure.lang.IPersistentVector [v, ^Writer w]
  (print-meta v w)
  (print-sequential "[" pr-on " " "] " v w))

(defn- print-map [m print-one w]
  (print-sequential
   "{"
   (fn [e ^Writer w]
     (do (print-one (key e) w) (.append w \space) (print-one (val e) w)))
   ","
   "}")
   (seq m) w))

(defmethod print-method clojure.lang.IPersistentMap [m, ^Writer w]
  (print-meta m w)
  (print-map m pr-on w))

```

```
(defmethod print-dup java.util.Map [m, ^Writer w]
  (print-ctor m #(print-map (seq %1) print-dup %2) w))

(defmethod print-dup clojure.lang.IPersistentMap [m, ^Writer w]
  (print-meta m w)
  (.write w "#(")
  (.write w (.getName (class m)))
  (.write w "/create ")
  (print-map m print-dup w)
  (.write w ")"))

(prefer-method print-dup
  clojure.lang.IPersistentCollection java.util.Map)

(defmethod print-method clojure.lang.IPersistentSet [s, ^Writer w]
  (print-meta s w)
  (print-sequential "#{" pr-on " " "}" (seq s) w))

(def ^{:tag String
       :doc "Returns name string for char or nil if none"
       :added "1.0"}
  char-name-string
  {\newline "newline"
   \tab "tab"
   \space "space"
   \backspace "backspace"
   \formfeed "formfeed"
   \return "return"})

(defmethod print-method java.lang.Character [^Character c, ^Writer w]
  (if (or *print-dup* *print-readably*)
    (do (.append w \\)
        (let [n (char-name-string c)]
          (if n (.write w n) (.append w c))))
        (.append w c))
    nil))

(defmethod print-dup java.lang.Character [c w] (print-method c w))
(defmethod print-dup java.lang.Integer [o w] (print-method o w))
(defmethod print-dup java.lang.Double [o w] (print-method o w))
(defmethod print-dup clojure.lang.Ratio [o w] (print-method o w))
(defmethod print-dup java.math.BigDecimal [o w] (print-method o w))
(defmethod print-dup clojure.lang.BigInt [o w] (print-method o w))
(defmethod print-dup java.math.BigInteger [o w] (print-method o w))
(defmethod print-dup
  clojure.lang.PersistentHashMap [o w] (print-method o w))
(defmethod print-dup
  clojure.lang.PersistentHashSet [o w] (print-method o w))
(defmethod print-dup
  clojure.lang.PersistentVector [o w] (print-method o w))
```

```

(defmethod print-dup
  clojure.lang.LazilyPersistentVector [o w] (print-method o w))

(def primitives-classnames
  {Float/TYPE "Float/TYPE"
  Integer/TYPE "Integer/TYPE"
  Long/TYPE "Long/TYPE"
  Boolean/TYPE "Boolean/TYPE"
  Character/TYPE "Character/TYPE"
  Double/TYPE "Double/TYPE"
  Byte/TYPE "Byte/TYPE"
  Short/TYPE "Short/TYPE"})

(defmethod print-method Class [^Class c, ^Writer w]
  (.write w (.getName c)))

(defmethod print-dup Class [^Class c, ^Writer w]
  (cond
    (.isPrimitive c) (do
      (.write w "#=(identity ")
      (.write w ^String (primitives-classnames c))
      (.write w ")"))

    (.isArray c) (do
      (.write w "#=(java.lang.Class.forName \\"")
      (.write w (.getName c))
      (.write w "\")"))

    :else (do
      (.write w "#=")
      (.write w (.getName c)))))

(defmethod print-method java.math.BigDecimal [b, ^Writer w]
  (.write w (str b))
  (.write w "M"))

(defmethod print-method clojure.lang.BigInt [b, ^Writer w]
  (.write w (str b))
  (.write w "N"))

(defmethod print-method java.math.BigInteger [b, ^Writer w]
  (.write w (str b))
  (.write w "BIGINT"))

(defmethod print-method java.util.regex.Pattern [p ^Writer w]
  (.write w "#\""))
  (loop [[^Character c & r :as s]
        (seq (.pattern ^java.util.regex.Pattern p))
        qmode false]
    (when s
      (cond
        (= c \\) (let [[^Character c2 & r2] r]
                  (.write w "#\\\"")
                  (.write w c2)
                  (.write w r2)))))))

```

```

(.append w \\)
(.append w c2)
(if qmode
    (recur r2 (not= c2 \E))
    (recur r2 (= c2 \Q))))
(= c \") (do
  (if qmode
      (.write w "\\" E \\ \"\\Q")
      (.write w "\\\""))
  (recur r qmode)))
:else (do
  (.append w c)
  (recur r qmode)))
(.append w \"))

(defmethod print-dup
  java.util.regex.Pattern [p ^Writer w] (print-method p w))

(defmethod print-dup
  clojure.lang.Namespace [^clojure.lang.Namespace n ^Writer w]
  (.write w "#=(find-ns ")
  (print-dup (.name n) w)
  (.write w ")"))

(defmethod print-method clojure.lang.IDeref [o ^Writer w]
  (print-sequential
    (format "#<%s@%x%s: "
           (.getSimpleName (class o))
           (System/identityHashCode o)
           (if (and (instance? clojure.lang.Agent o)
                     (agent-error o))
               " FAILED"
               ""))
    pr-on, "", ">",
    (list
      (cond
        (and (future? o) (not (future-done? o)))
        :pending
        (and (instance? clojure.lang.IPromiseImpl o)
             (not (.hasValue o)))
        :not-delivered
        :else @o)), w)))

(def ^{:private true} print-initialized true)

```

11.5 core·proxy.clj

— core·proxy.clj —

```
\getchunk{Clojure Copyright}

(in-ns 'clojure.core)

;;;;;; proxy ;;;;;;;

(import
  '(clojure.asm ClassWriter ClassVisitor Opcodes Type)
  '(java.lang.reflect Modifier Constructor)
  '(clojure.asm.commons Method GeneratorAdapter)
  '(clojure.lang IProxy Reflector
    DynamicClassLoader IPersistentMap PersistentHashMap RT))

(defn method-sig [^java.lang.reflect.Method meth]
  [(. meth getName)
   (seq (. meth getParameterTypes)) (. meth getReturnType)])

(defn- most-specific [rtyper]
  (or (some (fn [t] (when (every? #(isa? t %) rtyper) t)) rtyper)
      (throw (Exception. "Incompatible return types"))))

(defn- group-by-sig [coll]
  "takes a collection of [msig meth] and returns a seq of maps
  from return-types to meths."
  (vals (reduce1 (fn [m [msig meth]]
                  (let [rtype (peek msig)
                        argsig (pop msig)]
                    (assoc m argsig (assoc (m argsig {}) rtype meth))))
                 {} coll)))

(defn proxy-name
  {:tag String}
  [^Class super interfaces]
  (let [inames
        (into1 (sorted-set) (map #(.getName ^Class %) interfaces))]
    (apply str (.replace (str *ns*) \- \_) ".proxy"
      (interleave (repeat "$")
      (concat
        [(.getName super)]
        (map #(subs % (inc (.lastIndexOf ^String % ".") )) inames)
        [(Integer/toHexString (hash inames))])))))

(defn- generate-proxy [^Class super interfaces]
  (let [cv (new ClassWriter (. ClassWriter COMPUTE_MAXS))]
```

```

cname (.replace (proxy-name super interfaces) \. \/)
      ;(str "closure/lang/" (gensym "Proxy__"))
ctype (. Type (getObjectType cname))
iname (fn [^Class c] (.. Type (getType c) (getInternalName)))
fmap "__closureFnMap"
totype (fn [^Class c] (. Type (getType c)))
to-types (fn [cs] (if (pos? (count cs))
                      (into-array (map totype cs))
                      (make-array Type 0)))
super-type ^Type (totype super)
imap-type ^Type (totype IPersistentMap)
ifn-type (totype clojure.lang.IFn)
obj-type (totype Object)
sym-type (totype clojure.lang.Symbol)
rt-type (totype clojure.lang.RT)
ex-type (totype java.lang.UnsupportedOperationException)
gen-bridge
(fn [^java.lang.reflect.Method meth
     ^java.lang.reflect.Method dest]
  (let [pclasses (. meth (getParameterTypes))
        ptypes (to-types pclasses)
        rtype ^Type (totype (. meth (getReturnType)))
        m (new Method (. meth (getName)) rtype ptypes)
        dtype (totype (.getDeclaringClass dest))
        dm (new Method
             (. dest (getName))
             (totype (. dest (getReturnType)))
             (to-types (. dest (getParameterTypes))))
        gen (new GeneratorAdapter
              (bit-or
               (. Opcodes ACC_PUBLIC)
               (. Opcodes ACC_BRIDGE)) m nil nil cv)]
    (. gen (visitCode))
    (. gen (loadThis))
    (dotimes [i (count ptypes)]
      (. gen (loadArg i)))
    (if (-> dest .getDeclaringClass .isInterface)
        (. gen (invokeInterface dtype dm))
        (. gen (invokeVirtual dtype dm)))
    (. gen (returnValue))
    (. gen (endMethod))))
gen-method
(fn [^java.lang.reflect.Method meth else-gen]
  (let [pclasses (. meth (getParameterTypes))
        ptypes (to-types pclasses)
        rtype ^Type (totype (. meth (getReturnType)))
        m (new Method (. meth (getName)) rtype ptypes)
        gen (new GeneratorAdapter
              (. Opcodes ACC_PUBLIC) m nil nil cv)
        else-label (. gen (newLabel)))
    (. gen (label else-label))
    (. gen (tryBlock (. gen (tryLabel)))
          (. gen (catchBlock (. gen (catchLabel)))
                (. gen (returnValue))
                (. gen (endBlock)))
          (. gen (endTryBlock)))
    (. gen (label else-label))
    (. gen (else-block (. gen (elseLabel)))
          (. gen (returnValue))
          (. gen (endBlock)))
    (. gen (endElseBlock)))
  (. gen (label else-label))
  (. gen (returnValue))
  (. gen (endMethod))))
```

```

end-label (. gen (newLabel))
decl-type
  (. Type (getType (. meth (getDeclaringClass))))]
(. gen (visitCode))
(if (> (count pclasses) 18)
  (else-gen gen m)
  (do
    (. gen (loadThis))
    (. gen (getField ctype fmap imap-type))
    (. gen (push (. meth (getName))))
      ;lookup fn in map
    (. gen (invokeStatic rt-type
      (. Method
        (getMethod "Object get(Object, Object)")))
    (. gen (dup))
    (. gen (ifNull else-label))
      ;if found
    (.checkCast gen ifn-type)
    (. gen (loadThis))
      ;box args
    (dotimes [i (count ptypes)]
      (. gen (loadArg i))
      (. clojure.lang.Compiler$HostExpr
        (emitBoxReturn nil gen (nth pclasses i)))
      ;call fn
    (. gen (invokeInterface ifn-type
      (new Method "invoke" obj-type
        (into-array (cons obj-type
          (replicate (count ptypes) obj-type)))))))
      ;unbox return
    (. gen (unbox rtype))
    (when (= (. rtype (getSort)) (. Type VOID))
      (. gen (pop)))
    (. gen (goTo end-label))
      ;else call supplied alternative generator
    (. gen (mark else-label))
    (. gen (pop))

    (else-gen gen m)

    (. gen (mark end-label))))
  (. gen (returnValue))
  (. gen (endMethod)))] ;start class definition

(. cv (visit
  (. Opcodes V1_5)
  (+ (. Opcodes ACC_PUBLIC) (. Opcodes ACC_SUPER))
  cname nil (iname super)
  (into-array (map iname (cons IProxy interfaces)))))


```

```

;add field for fn mappings
(. cv (visitField
      (+ (. Opcodes ACC_PRIVATE) (. Opcodes ACC_VOLATILE))
      fmap (. imap-type (getDescriptor)) nil nil))
;add ctors matching/calling super's
(doseq [^Constructor ctor (. super (getDeclaredConstructors))]
  (when-not (. Modifier (isPrivate (. ctor (getModifiers))))
    (let [ptypes (to-types (. ctor (getParameterTypes)))
          m (new Method "<init>" (. Type VOID_TYPE) ptypes)
          gen (new GeneratorAdapter
                (. Opcodes ACC_PUBLIC) m nil nil cv)]
      (. gen (visitCode))
      (. gen (visitField (. getDescriptor) (. getModifiers) (. getType) (. getName) (. getSignature) (. getDeclaringClass) (. getAnnotations)))
        ;call super ctor
        (. gen (loadThis))
        (. gen (dup))
        (. gen (loadArgs))
        (. gen (invokeConstructor super-type m))

        (. gen (returnValue))
        (. gen (endMethod)))))

;add IProxy methods
(let [m (. Method
          (getMethod
            "void __initClojureFnMappings(clojure.lang.IPersistentMap)")
            gen (new GeneratorAdapter (. Opcodes ACC_PUBLIC) m nil nil cv))]
  (. gen (visitCode))
  (. gen (loadThis))
  (. gen (loadArgs))
  (. gen (putField ctype fmap imap-type))

  (. gen (returnValue))
  (. gen (endMethod)))
(let [m (. Method
          (getMethod
            "void __updateClojureFnMappings(clojure.lang.IPersistentMap)")
            gen (new GeneratorAdapter (. Opcodes ACC_PUBLIC) m nil nil cv))]
  (. gen (visitCode))
  (. gen (loadThis))
  (. gen (dup))
  (. gen (getField ctype fmap imap-type))
  (. checkCast gen (totype clojure.lang.IPersistentCollection))
  (. gen (loadArgs))
  (. gen (invokeInterface (totype clojure.lang.IPersistentCollection)
    (. Method
      (getMethod
        "clojure.lang.IPersistentCollection cons(Object)")))
    (. gen (checkCast imap-type))
    (. gen (putField ctype fmap imap-type))

  (. gen (returnValue)))

```

```

(. gen (endMethod)))
(let [m (. Method
           (getMethod
             "clojure.lang.IPersistentMap __getClojureFnMappings()"))
      gen (new GeneratorAdapter (. Opcodes ACC_PUBLIC) m nil nil cv)]
  (. gen (visitCode))
  (. gen (loadThis))
  (. gen (getField ctype fmap imap-type))
  (. gen (returnValue))
  (. gen (endMethod)))

;calc set of supers' non-private instance methods
(let [[mm considered]
      (loop [mm {} considered #{} c super]
        (if c
            (let [[mm considered]
                  (loop [mm mm
                         considered considered
                         meths (concat
                                 (seq (. c (getDeclaredMethods)))
                                 (seq (. c (getMethods))))]
                (if (seq meths)
                    (let [^java.lang.reflect.Method meth
                          (first meths)
                          mods (. meth (getModifiers))
                          mk (method-sig meth)]
                      (if (or (considered mk)
                              (not
                                (or
                                  (Modifier/isPublic mods)
                                  (Modifier/isProtected mods)))
                              ;(. Modifier (isPrivate mods))
                              (. Modifier (isStatic mods))
                              (. Modifier (isFinal mods))
                              (= "finalize" (.getName meth)))
                      (recur mm
                            (conj considered mk) (next meths)))
                      (recur
                        (assoc mm mk meth)
                        (conj considered mk)
                        (next meths))))
                      [mm considered]))]
                    (recur mm considered (. c (getSuperclass)))
                    [mm considered]))
            ifaces-meths
            (into1 {}
              (for [^Class iface interfaces meth (. iface (getMethods))
                    :let [msig (method-sig meth)]
                    :when (not (considered msig))]
                {msig meth}))]
```

```

mgroups (group-by-sig (concat mm ifaces-meths))
rtypes (map #(most-specific (keys %)) mgroups)
mb (map #(vector (%1 %2) (vals (dissoc %1 %2))) mgroups rtypes)
bridge? (reduce1 into1 #{} (map second mb))
ifaces-meths (remove bridge? (vals ifaces-meths))
mm (remove bridge? (vals mm))]
;add methods matching supers', if no mapping -> call super
(doseq [[^java.lang.reflect.Method dest bridges] mb
        ^java.lang.reflect.Method meth bridges]
  (gen-bridge meth dest))
(doseq [^java.lang.reflect.Method meth mm]
  (gen-method meth
    (fn [^GeneratorAdapter gen ^Method m]
      (. gen (loadThis))
      ;push args
      (. gen (loadArgs))
      ;call super
      (. gen (visitMethodInsn
        (. Opcodes INVOKESPECIAL)
        (. super-type (getInternalName))
        (. m (getName))
        (. m (getDescriptor)))))))
;add methods matching interfaces', if no mapping -> throw
(doseq [^java.lang.reflect.Method meth ifaces-meths]
  (gen-method meth
    (fn [^GeneratorAdapter gen ^Method m]
      (. gen (throwException ex-type (. m (getName)))))))
;finish class def
(. cv (visitEnd))
[cname (. cv toByteArray)])
(defn- get-super-and-interfaces [bases]
  (if (. ^Class (first bases) (isInterface))
    [Object bases]
    [(first bases) (next bases)]))

(defn get-proxy-class
  "Takes an optional single class followed by zero or more
  interfaces. If not supplied class defaults to Object. Creates an
  returns an instance of a proxy class derived from the supplied
  classes. The resulting value is cached and used for any subsequent
  requests for the same class set. Returns a Class object."
  {:added "1.0"}
  [& bases]
  (let [[super interfaces] (get-super-and-interfaces bases)
        pname (proxy-name super interfaces)]
    (or (RT/loadClassForName pname)
        (let [[cname bytecode] (generate-proxy super interfaces)]
          cname))))

```

```

(. ^DynamicClassLoader
  (deref clojure.lang.Compiler/LOADER)
  (defineClass pname bytecode [super interfaces])))))

(defn construct-proxy
  "Takes a proxy class and any arguments for its superclass ctor and
  creates and returns an instance of the proxy."
  {:added "1.0"}
  [c & ctor-args]
  (. Reflector (invokeConstructor c (to-array ctor-args)))))

(defn init-proxy
  "Takes a proxy instance and a map of strings (which must
  correspond to methods of the proxy superclass/superinterfaces) to
  fns (which must take arguments matching the corresponding method,
  plus an additional (explicit) first arg corresponding to this, and
  sets the proxy's fn map. Returns the proxy."
  {:added "1.0"}
  [^IPProxy proxy mappings]
  (. proxy ( __initClojureFnMappings mappings))
  proxy)

(defn update-proxy
  "Takes a proxy instance and a map of strings (which must
  correspond to methods of the proxy superclass/superinterfaces) to
  fns (which must take arguments matching the corresponding method,
  plus an additional (explicit) first arg corresponding to this, and
  updates (via assoc) the proxy's fn map. nil can be passed instead of
  a fn, in which case the corresponding method will revert to the
  default behavior. Note that this function can be used to update the
  behavior of an existing instance without changing its identity.
  Returns the proxy."
  {:added "1.0"}
  [^IPProxy proxy mappings]
  (. proxy ( __updateClojureFnMappings mappings))
  proxy)

(defn proxy-mappings
  "Takes a proxy instance and returns the proxy's fn map."
  {:added "1.0"}
  [^IPProxy proxy]
  (. proxy ( __getClojureFnMappings)))

(defmacro proxy
  "class-and-interfaces - a vector of class names
  args - a (possibly empty) vector of arguments to the superclass
  constructor.

  f => (name [params*] body) or

```

(name ([params*] body) ([params+] body) ...)

Expands to code which creates a instance of a proxy class that implements the named class/interface(s) by calling the supplied fns. A single class, if provided, must be first. If not provided it defaults to Object.

The interfaces names must be valid interface types. If a method fn is not provided for a class method, the superclass methd will be called. If a method fn is not provided for an interface method, an `UnsupportedOperationException` will be thrown should it be called. Method fns are closures and can capture the environment in which proxy is called. Each method fn takes an additional implicit first arg, which is bound to 'this'. Note that while method fns can be provided to override protected methods, they have no other access to protected members, nor to super, as these capabilities cannot be proxied."

```

{:added "1.0"}
[class-and-interfaces args & fs]
(let [bases (map
             #(or (resolve %)
                  (throw (Exception. (str "Can't resolve: " %))))
             class-and-interfaces)
      [super interfaces] (get-super-and-interfaces bases)
      compile-effect
      (when *compile-files*
        (let [[cname bytecode] (generate-proxy super interfaces)]
          (clojure.lang.Compiler/writeClassFile cname bytecode)))
      pc-effect (apply get-proxy-class bases)
      pname (proxy-name super interfaces)])
;remember the class to prevent it from disappearing before use
(intern *ns* (symbol pname) pc-effect)
'(let [&pc# (get-proxy-class `@class-and-interfaces)
       p# (new `(symbol pname) `@args)]
   ;(construct-proxy pc# `@args])
  (init-proxy p#
  ~loop [fmap {} fs fs]
    (if fs
        (let [[sym & meths] (first fs)
              meths (if (vector? (first meths))
                        (list meths)
                        meths)
              meths
              (map (fn [[params & body]]
                     (cons (apply vector 'this params) body))
                   meths)]
          (if-not (contains? fmap (name sym))
            (recur
              (assoc fmap (name sym) (cons 'fn meths)) (next fs))
              (throw (IllegalArgumentException.

```

```

(str "Method '" (name sym) "' redefined")))))
fmap)))
p#)))

(defn proxy-call-with-super [call this meth]
(let [m (proxy-mappings this)]
(update-proxy this (assoc m meth nil))
(let [ret (call)]
(update-proxy this m)
ret)))

(defmacro proxy-super
"Use to call a superclass method in the body of a proxy method.
Note, expansion captures 'this"
{:added "1.0"}
[meth & args]
(proxy-call-with-super
(fn [] (. ^'this ^meth ^@args)) ^'this ^(name meth)))

(defn bean
"Takes a Java object and returns a read-only implementation of the
map abstraction based upon its JavaBean properties."
{:added "1.0"}
[^Object x]
(let [c (. x getClass))
pmap
(reduce1 (fn [m ^java.beans.PropertyDescriptor pd]
(let [name (. pd getName)
method (. pd (getReadMethod))]
(if (and method
(zero? (alength (. method (getParameterTypes)))))
(assoc m (keyword name)
(fn []
(closure.lang.Reflector/prepRet
(.getPropertyType pd) (. method (invoke x nil)))
m)))
{}) (seq (.. java.beans.Introspector
(getBeanInfo c)
(getPropertyDescriptors)))))

v (fn [k] ((pmap k)))
snapshot (fn []
(reduce1 (fn [m e]
(assoc m (key e) ((val e))))
{} (seq pmap)))
(proxy [closure.lang.APersistentMap]
[]
(containsKey [k] (contains? pmap k))
(entryAt [k]

```

```

        (when (contains? pmap k) (new clojure.lang.MapEntry k (v k)))
  (valAt ([k] (v k))
    ([k default] (if (contains? pmap k) (v k) default)))
  (cons [m] (conj (snapshot) m))
  (count [] (count pmap))
  (assoc [k v] (assoc (snapshot) k v))
  (without [k] (dissoc (snapshot) k))
  (seq [])
  ((fn thisfn [plseq]
    (lazy-seq
      (when-let [pseq (seq plseq)]
        (cons
          (new clojure.lang.MapEntry (first pseq) (v (first pseq)))
          (thisfn (rest pseq)))))) (keys pmap))))))

```

11.6 data.clj

— data.clj —

```

\getchunk{Clojure Copyright}

(ns
  ^{:author "Stuart Halloway",
    :doc "Non-core data functions."}
  clojure.data
  (:require [clojure.set :as set]))


(defn- atom-diff
  "Internal helper for diff."
  [a b]
  (if (= a b) [nil nil a] [a b nil]))


;; for big things a sparse vector class would be better
(defn- vectorize
  "Convert an associative-by-numeric-index collection into
  an equivalent vector, with nil for any missing keys"
  [m]
  (when (seq m)
    (reduce
      (fn [result [k v]] (assoc result k v))
      (vec (repeat (apply max (keys m)) nil)))
      m)))

(declare diff)

```

```

(defprotocol {:added "1.3"} EqualityPartition
  "Implementation detail. Subject to change."
  (equality-partition [x]
    "Implementation detail. Subject to change.))

(defprotocol {:added "1.3"} Diff
  "Implementation detail. Subject to change."
  (diff-similar [a b]
    "Implementation detail. Subject to change.))

(extend nil
  Diff
  {:diff-similar atom-diff})

(extend Object
  Diff
  {:diff-similar atom-diff}
  EqualityPartition
  {:equality-partition
    (fn [x] (if (.. x getClass isArray) :sequential :atom))}

(defn- diff-associative
  "Diff associative things a and b, comparing only keys in ks."
  [a b ks]
  (reduce
    (fn [diff1 diff2]
      (map merge diff1 diff2))
    [nil nil nil])
  (map
    (fn [k] (map #(when % {k %}) (diff (get a k) (get b k))))
    ks)))

(extend-protocol EqualityPartition
  nil
  (equality-partition [x] :atom)

  java.util.Set
  (equality-partition [x] :set)

  java.util.List
  (equality-partition [x] :sequential)

  java.util.Map
  (equality-partition [x] :map))

(extend-protocol Diff
  java.util.Set
  (diff-similar [a b]
    [(not-empty (set/difference a b))])

```

```

(not-empty (set/difference b a))
(not-empty (set/intersection a b))]

java.util.List
(diff-similar [a b]
  (vec (map vectorize (diff-associative
    (if (vector? a) a (vec a))
    (if (vector? b) b (vec b))
    (range (max (count a) (count b)))))))

java.util.Map
(diff-similar [a b]
  (diff-associative a b (set/union (keys a) (keys b)))))

(defn diff
  "Recursively compares a and b, returning a tuple of
  [things-only-in-a things-only-in-b things-in-both].
  Comparison rules:
  * Maps are subdiffed where keys match and values differ.
  * Sets are never subdiffed.
  * All sequential things are treated as associative collections
    by their indexes, with results returned as vectors.
  * Everything else (including strings!) is treated as
    an atom and compared for equality."
  {:added "1.3"}
  [a b]
  (if (= (equality-partition a) (equality-partition b))
    (diff-similar a b)
    (atom-diff a b)))

```

11.7 genclass.clj

— genclass.clj —

```

\getchunk{Clojure Copyright}

(in-ns 'clojure.core)

(import '(java.lang.reflect Modifier Constructor)
        '(clojure.asm ClassWriter ClassVisitor Opcodes Type)
        '(clojure.asm.commons Method GeneratorAdapter)
        '(clojure.lang IPersistentMap))

```

```

;(defn method-sig [^java.lang.reflect.Method meth]
;  [(. meth (getName)) (seq (. meth (getParameterTypes)))])

(defn- non-private-methods [^Class c]
  (loop [mm {}]
    considered #{}
    c c]
  (if c
    (let [[mm considered]
          (loop [mm mm
                 considered considered
                 meths (seq (concat
                             (seq (. c (getDeclaredMethods)))
                             (seq (. c (getMethods)))))]
      (if meths
          (let [^java.lang.reflect.Method meth (first meths)
                mods (. meth (getModifiers))
                mk (method-sig meth)]
            (if (or (considered mk)
                    (not (or (Modifier/isPublic mods)
                              (Modifier/isProtected mods)))
                    ;(. Modifier (isPrivate mods))
                    (. Modifier (isStatic mods))
                    (. Modifier (isFinal mods))
                    (= "finalize" (. getName meth)))
                (recur mm (conj considered mk) (next meths)))
            (recur (assoc mm mk meth)
                  (conj considered mk)
                  (next meths))))
          [mm considered]))]
      (recur mm considered (. c (getSuperclass())))
      mm)))
    (recur mm considered (. c (getSuperclass())))
    mm))

(defn- ctor-sigs [^Class super]
  (for [^Constructor ctor (. super (getDeclaredConstructors))
        :when (not (. Modifier (isPrivate (. ctor (getModifiers)))))]
    (apply vector (. ctor (getParameterTypes)))))

(defn- escape-class-name [^Class c]
 (.. (.getSimpleName c)
      (replace "[]" "<>"))

(defn- overload-name [mname pclasses]
  (if (seq pclasses)
    (apply str mname (interleave (repeat \-)
                                 (map escape-class-name pclasses)))
    (str mname "-void")))

(defn- ^java.lang.reflect.Field find-field [^Class c f]
  (let [start-class c]

```

```

(loop [c c]
  (if (= c Object)
      (throw (new Exception
                    (str "field, " f ", not defined in class,
                          " start-class ", or its ancestors")))
      (let [dflds (.getDeclaredFields c)
            rfld (first
                  (filter
                    #(= f (.getName ^java.lang.reflect.Field %))
                    dflds))])
        (or rfld (recur (.getSuperclass c)))))))

;(distinct (map first(keys
;  (mapcat non-private-methods [Object IPersistentMap]))))

(def ^{:private true} prim->class
  {'int Integer/TYPE
   'long Long/TYPE
   'float Float/TYPE
   'double Double/TYPE
   'void Void/TYPE
   'short Short/TYPE
   'boolean Boolean/TYPE
   'byte Byte/TYPE
   'char Character/TYPE})

(defn- ^Class the-class [x]
  (cond
    (class? x) x
    (contains? prim->class x) (prim->class x)
    :else (let [strx (str x)]
             (clojure.lang.RT/classForName
               (if (some #{\\" \[\]} strx)
                   strx
                   (str "java.lang." strx))))))

;; someday this can be made codepoint aware
(defn- valid-java-method-name
  [^String s]
  (= s (clojure.lang.Compiler/munge s)))

(defn- validate-generate-class-options
  [{:keys [methods]}]
  (let [[mname]
        (remove valid-java-method-name
                (map (comp str first) methods))]
    (when mname
      (throw (IllegalArgumentException.
                    (str "Not a valid method name: " mname))))))

```

```
(defn- generate-class [options-map]
  (validate-generate-class-options options-map)
  (let [default-options
        {:prefix "-"
         :load-impl-ns true
         :impl-ns (ns-name *ns*)}
        {:keys [name extends implements constructors methods main
                factory state init exposes
                exposes-methods prefix load-impl-ns impl-ns post-init]}
        (merge default-options options-map)
        name-meta (meta name)
        name (str name)
        super (if extends (the-class extends) Object)
        interfaces (map the-class implements)
        supers (cons super interfaces)
        ctor-sig-map
        (or constructors (zipmap (ctor-sigs super) (ctor-sigs super)))
        cv (new ClassWriter (. ClassWriter COMPUTE_MAXS))
        cname (. name (replace "." "/"))
        pkg-name name
        impl-pkg-name (str impl-ns)
        impl-cname (.. impl-pkg-name (replace "." "/") (replace \- \_))
        ctype (. Type (getObjectType cname))
        iname (fn [^Class c] (.. Type (getType c) (getInternalName)))
        totype (fn [^Class c] (. Type (getType c)))
        to-types (fn [cs] (if (pos? (count cs))
                               (into-array (map totype cs))
                               (make-array Type 0)))
        obj-type ^Type (totype Object)
        arg-types (fn [n] (if (pos? n)
                               (into-array (replicate n obj-type))
                               (make-array Type 0)))
        super-type ^Type (totype super)
        init-name (str init)
        post-init-name (str post-init)
        factory-name (str factory)
        state-name (str state)
        main-name "main"
        var-name (fn [s] (clojure.lang.Compiler/munge (str s "__var")))
        class-type (totype Class)
        rt-type (totype clojure.lang.RT)
        var-type ^Type (totype clojure.lang.Var)
        ifn-type (totype clojure.lang.IFn)
        iseq-type (totype clojure.lang.ISeq)
        ex-type (totype java.lang.UnsupportedOperationException)
        all-sigs
        (distinct
         (concat
          (map #(let[[m p] (key %)] {m [p]})
                (mapcat non-private-methods supers))
          (map (fn [[m p]] {(str m) [p]}) methods)))
        sigs-by-name (apply merge-with concat {}) all-sigs)
```

```

overloads (into1 {}) (filter (fn [[m s]] (next s)) sigs-by-name))
var-fields (concat (when init [init-name])
                    (when post-init [post-init-name])
                    (when main [main-name])
                    ;(when exposes-methods
                    ;    (map str (vals exposes-methods)))
                    (distinct
                     (concat
                      (keys sigs-by-name)
                      (mapcat
                       (fn [[m s]]
                         (map
                          #(overload-name m (map the-class %))
                          s))
                         overloads)
                      (mapcat
                       (comp (partial map str) vals val)
                       exposes))))
emit-get-var (fn [^GeneratorAdapter gen v]
               (let [false-label (. gen newLabel)
                     end-label (. gen newLabel)]
                 (. gen getStatic ctype (var-name v) var-type)
                 (. gen dup)
                 (. gen invokeVirtual var-type
                   (. Method (getMethod "boolean isBound()")))
                 (. gen ifZCmp
                   (. GeneratorAdapter EQ) false-label)
                 (. gen invokeVirtual var-type
                   (. Method (getMethod "Object get()")))
                 (. gen goTo end-label)
                 (. gen mark false-label)
                 (. gen pop)
                 (. gen visitInsn (. Opcodes ACONST_NULL))
                 (. gen mark end-label)))
emit-unsupported
(fn [^GeneratorAdapter gen ^Method m]
  (. gen (throwException ex-type
    (str (. m (getName)) " (""
          impl-pkg-name "/" prefix (.getName m)
          " not defined?)))))
emit-forwarding-method
(fn [name pclasses rclass as-static else-gen]
  (let [mname (str name)
        pmetas (map meta pclasses)
        pclasses (map the-class pclasses)
        rclass (the-class rclass)
        ptypes (to-types pclasses)
        rtype ^Type (totype rclass)
        m (new Method mname rtype ptypes)
        is-overload (seq (overloads mname)))
    
```

```

gen (new GeneratorAdapter
  (+
    (. Opcodes ACC_PUBLIC)
    (if as-static (. Opcodes ACC_STATIC 0))
    m nil nil cv)
  found-label (. gen (newLabel))
  else-label (. gen (newLabel))
  end-label (. gen (newLabel)))
(add-annotations gen (meta name))
(dotimes [i (count pmetas)]
  (add-annotations gen (nth pmetas i) i))
(. gen (visitCode))
(if (> (count pclasses) 18)
  (else-gen gen m)
  (do
    (when is-overload
      (emit-get-var gen (overload-name mname pclasses))
      (. gen (dup))
      (. gen (ifNonNull found-label))
      (. gen (pop)))
    (emit-get-var gen mname)
    (. gen (dup))
    (. gen (ifNull else-label))
    (when is-overload
      (. gen (mark found-label)))
    ;if found
    (. checkCast gen ifn-type)
    (when-not as-static
      (. gen (loadThis)))
    ;box args
    (dotimes [i (count ptypes)]
      (. gen (loadArg i))
      (. clojure.lang.Compiler$HostExpr
        (emitBoxReturn nil gen (nth pclasses i))))
    ;call fn
    (. gen
      (invokeInterface ifn-type
        (new Method "invoke" obj-type
          (to-types (replicate (+ (count ptypes)
            (if as-static 0 1))
          Object)))))

    ;(into-array
    ; (cons obj-type
    ;   (replicate (count ptypes) obj-type))))))

    ;unbox return
    (. gen (unbox rtype))
    (when (= (. rtype (getSort)) (. Type VOID))
      (. gen (pop)))
    (. gen (goTo end-label)))
  )
)

```

```

;else call supplied alternative generator
(. gen (mark else-label))
(. gen (pop))

(else-gen gen m)

(. gen (mark end-label)))
(. gen (returnValue))
(. gen (endMethod)))
]

;start class definition
(. cv (visit
(. Opcodes V1_5)
(+ (. Opcodes ACC_PUBLIC) (. Opcodes ACC_SUPER))
cname nil (iname super)
(when-let [ifc (seq interfaces)]
(into-array (map iname ifc)))))

; class annotations
(add-annotations cv name-meta)

;static fields for vars
(doseq [v var-fields]
(. cv (visitField
(+ (. Opcodes ACC_PRIVATE)
(. Opcodes ACC_FINAL)
(. Opcodes ACC_STATIC))
(var-name v)
(var-type getDescriptor)
nil nil)))

;instance field for state
(when state
(. cv (visitField (+ (. Opcodes ACC_PUBLIC) (. Opcodes ACC_FINAL))
state-name
(obj-type getDescriptor)
nil nil)))

;static init to set up var fields and load init
(let [gen (new GeneratorAdapter
(+ (. Opcodes ACC_PUBLIC) (. Opcodes ACC_STATIC))
(Method getMethod "void <clinit> ()")
nil nil cv)]
(. gen (visitCode))
(doseq [v var-fields]
(. gen push impl-pkg-name)
(. gen push (str prefix v))
(. gen (invokeStatic var-type
(. Method
getMethod

```

```

    "clojure.lang.Var internPrivate(String,String))))))
(. gen putStatic ctype (var-name v) var-type))

(when load-impl-ns
  (. gen push "clojure.core")
  (. gen push "load")
  (. gen
    (invokeStatic rt-type
      (. Method
        (getMethod "clojure.lang.Var var(String,String)))))
    (. gen push (str "/" impl-cname))
  (. gen
    (invokeInterface ifn-type
      (new Method "invoke" obj-type (to-types [Object])))))
; (. gen push (str (.replace impl-pkg-name \- \_) "__init"))
; (. gen
;   (invokeStatic class-type
;     (. Method (getMethod "Class forName(String))))))
; (. gen pop)

  (. gen (returnValue))
  (. gen (endMethod)))

;ctors
(doseq [[pclasses super-pclasses] ctor-sig-map]
  (let [pclasses (map the-class pclasses)
        super-pclasses (map the-class super-pclasses)
        ptypes (to-types pclasses)
        super-ptypes (to-types super-pclasses)
        m (new Method "<init>" (. Type VOID_TYPE) ptypes)
        super-m
        (new Method "<init>" (. Type VOID_TYPE) super-ptypes)
        gen
        (new GeneratorAdapter (. Opcodes ACC_PUBLIC) m nil nil cv)
        no-init-label (. gen newLabel)
        end-label (. gen newLabel)
        no-post-init-label (. gen newLabel)
        end-post-init-label (. gen newLabel)
        nth-method (. Method (getMethod "Object nth(Object,int)"))
        local (. gen newLocal obj-type)]
    (. gen (visitCode))

    (if init
        (do
          (emit-get-var gen init-name)
          (. gen dup)
          (. gen ifNull no-init-label)
          (.checkCast gen ifn-type)
            ;box init args
        (dotimes [i (count pclasses)])

```

```

(. gen (loadArg i))
(. clojure.lang.Compiler$HostExpr
  (emitBoxReturn nil gen (nth pclasses i))))
;call init fn
(. gen (invokeInterface ifn-type
  (new Method "invoke" obj-type
    (arg-types (count ptypes)))))

;expecting [[super-ctor-args] state] returned
(. gen dup)
(. gen push (int 0))
(. gen (invokeStatic rt-type nth-method))
(. gen storeLocal local)

(. gen (loadThis))
(. gen dupX1)
(dotimes [i (count super-pclasses)]
  (. gen loadLocal local)
  (. gen push (int i))
  (. gen (invokeStatic rt-type nth-method))
  (. clojure.lang.Compiler$HostExpr
    (emitUnboxArg nil gen (nth super-pclasses i))))
  (. gen (invokeConstructor super-type super-m))

(if state
  (do
    (. gen push (int 1))
    (. gen (invokeStatic rt-type nth-method))
    (. gen (putField ctype state-name obj-type)))
  (. gen pop))

  (. gen goTo end-label)
  ;no init found
  (. gen mark no-init-label)
  (. gen (throwException ex-type
    (str impl-pkg-name "/" prefix init-name
      " not defined")))
  (. gen mark end-label))
(if (= pclasses super-pclasses)
  (do
    (. gen (loadThis))
    (. gen (loadArgs))
    (. gen (invokeConstructor super-type super-m)))
  (throw (new Exception
    ":init not specified, but ctor and super ctor args differ")))

(when post-init
  (emit-get-var gen post-init-name)
  (. gen dup)
  (. gen ifNull no-post-init-label)
  (.checkCast gen ifn-type)

```

```

(. gen (loadThis))
;box init args
(dotimes [i (count pclasses)]
  (. gen (loadArg i))
  (. clojure.lang.Compiler$HostExpr
    (emitBoxReturn nil gen (nth pclasses i))))
;call init fn
(. gen (invokeInterface ifn-type
  (new Method "invoke" obj-type
    (arg-types (inc (count ptypes)))))))
(. gen pop)
(. gen goTo end-post-init-label)
;no init found
(. gen mark no-post-init-label)
(. gen (throwException ex-type
  (str impl-pkg-name "/" prefix post-init-name
    " not defined")))
(. gen mark end-post-init-label))

(. gen (returnValue))
(. gen (endMethod))
;factory
(when factory
  (let [fm (new Method factory-name ctype ptypes)
        gen (new GeneratorAdapter
          (+ (. Opcodes ACC_PUBLIC) (. Opcodes ACC_STATIC))
          fm nil nil cv)]
    (. gen (visitCode))
    (. gen newInstance ctype)
    (. gen dup)
    (. gen (loadArgs))
    (. gen (invokeConstructor ctype m))
    (. gen (returnValue))
    (. gen (endMethod)))))

;add methods matching supers', if no fn -> call super
(let [mm (non-private-methods super)]
  (doseq [^java.lang.reflect.Method meth (vals mm)]
    (emit-forwarding-method
      (.getName meth)
      (.getParameterTypes meth)
      (.getReturnType meth)
      false
      (fn [^GeneratorAdapter gen ^Method m]
        (. gen (loadThis))
        ;push args
        (. gen (loadArgs))
        ;call super
        (. gen (visitMethodInsn
          (. Opcodes INVOKEESPECIAL)

```

```

(. super-type (getInternalName))
(. m (getName))
(. m (getDescriptor))))))
;add methods matching interfaces', if no fn -> throw
(reduce1 (fn [mm ^java.lang.reflect.Method meth]
(if (contains? mm (method-sig meth))
mm
(do
(emit-forwarding-method
(.getName meth)
(.getParameterTypes meth)
(.getReturnType meth)
false
	emit-unsupported)
(assoc mm (method-sig meth) meth)))
mm (mapcat #(.getMethods ^Class %) interfaces))
;extra methods
(doseq [[mname pclasses rclass :as msig] methods]
(emit-forwarding-method mname pclasses rclass
(:static (meta msig))
emit-unsupported))
;expose specified overridden superclass methods
(doseq [[local-mname ^java.lang.reflect.Method m]
(reduce1
(fn [ms [[name _ _] m]]
(if (contains? exposes-methods (symbol name))
(conj ms [((symbol name) exposes-methods) m])
ms) [] (seq mm)))
(let [ptypes (to-types (.getParameterTypes m))
rtype (totype (.getReturnType m))
exposer-m (new Method (str local-mname) rtype ptypes)
target-m (new Method (.getName m) rtype ptypes)
gen (new GeneratorAdapter
(. Opcodes ACC_PUBLIC) exposer-m nil nil cv)]
(. gen (loadThis))
(. gen (loadArgs))
(. gen (visitMethodInsn (. Opcodes INVOKESTATIC)
(. super-type (getInternalName))
(. target-m (getName))
(. target-m (getDescriptor)))))

(. gen (returnValue))
(. gen (endMethod)))))
;main
(when main
(let [m (. Method getMethod "void main (String[])")
gen (new GeneratorAdapter
(+ (. Opcodes ACC_PUBLIC) (. Opcodes ACC_STATIC))
m nil nil cv)
no-main-label (. gen newLabel)
end-label (. gen newLabel)])

```

```

(. gen (visitCode))

(emit-get-var gen main-name)
(. gen dup)
(. gen ifNull no-main-label)
(.checkCast gen ifn-type)
(. gen loadArgs)
(. gen (invokeStatic rt-type
  (. Method (getMethod "clojure.lang.ISeq seq(Object)")))
(. gen (invokeInterface ifn-type (new Method "applyTo" obj-type
  (into-array [iseq-type])))))
(. gen pop)
(. gen goTo end-label)
;no main found
(. gen mark no-main-label)
(. gen (throwException ex-type
  (str impl-pkg-name "/" prefix main-name " not defined")))
(. gen mark end-label)
(. gen (returnValue))
(. gen (endMethod)))
;field exposers
(doseq [[f {getter :get setter :set}] exposes]
  (let [fld (find-field super (str f))
    ftype (totype (.getType fld))
    static? (Modifier/isStatic (.getModifiers fld))
    acc (+ Opcodes/ACC_PUBLIC (if static? Opcodes/ACC_STATIC 0))]
  (when getter
    (let [m (new Method (str getter) ftype (to-types []))
      gen (new GeneratorAdapter acc m nil nil cv)]
      (. gen (visitCode))
      (if static?
        (. gen getStatic ctype (str f) ftype)
        (do
          (. gen loadThis)
          (. gen getField ctype (str f) ftype)))
      (. gen (returnValue))
      (. gen (endMethod))))
  (when setter
    (let [m (new Method
      (str setter) Type/VOID_TYPE (into-array [ftype]))
      gen (new GeneratorAdapter acc m nil nil cv)]
      (. gen (visitCode))
      (if static?
        (do
          (. gen loadArgs)
          (. gen putStatic ctype (str f) ftype))
        (do
          (. gen loadThis)
          (. gen loadArgs)
          (. gen putField ctype (str f) ftype))))
```

```

        (. gen (returnValue))
        (. gen (endMethod))))))
;finish class def
(. cv (visitEnd))
[cname (. cv (toByteArray))])

(defmacro gen-class
  "When compiling, generates compiled bytecode for a class with the
  given package-qualified :name (which, as all names in these
  parameters, can be a string or symbol), and writes the .class file
  to the *compile-path* directory. When not compiling, does
  nothing. The gen-class construct contains no implementation, as the
  implementation will be dynamically sought by the generated class in
  functions in an implementing Clojure namespace. Given a generated
  class org.mydomain.MyClass with a method named mymethod, gen-class
  will generate an implementation that looks for a function named by
  (str prefix mymethod) (default prefix: \"-\") in a
  Clojure namespace specified by :impl-ns
  (defaults to the current namespace). All inherited methods,
  generated methods, and init and main functions (see :methods, :init,
  and :main below) will be found similarly prefixed. By default, the
  static initializer for the generated class will attempt to load the
  Clojure support code for the class as a resource from the classpath,
  e.g. in the example case, `org/mydomain/MyClass__init.class`. This
  behavior can be controlled by :load-impl-ns"

```

Note that methods with a maximum of 18 parameters are supported.

In all subsequent sections taking types, the primitive types can be referred to by their Java names (int, float etc), and classes in the java.lang package can be used without a package qualifier. All other classes must be fully qualified.

Options should be a set of key/value pairs, all except for :name are optional:

:name cname

The package-qualified name of the class to be generated

:extends aclass

Specifies the superclass, the non-private methods of which will be overridden by the class. If not provided, defaults to Object.

:implements [interface ...]

One or more interfaces, the methods of which will be implemented by the class.

```
:init name
```

If supplied, names a function that will be called with the arguments to the constructor. Must return [[superclass-constructor-args] state] If not supplied, the constructor args are passed directly to the superclass constructor and the state will be nil

```
:constructors {[param-types] [super-param-types], ...}
```

By default, constructors are created for the generated class which match the signature(s) of the constructors for the superclass. This parameter may be used to explicitly specify constructors, each entry providing a mapping from a constructor signature to a superclass constructor signature. When you supply this, you must supply an :init specifier.

```
:post-init name
```

If supplied, names a function that will be called with the object as the first argument, followed by the arguments to the constructor. It will be called every time an object of this class is created, immediately after all the inherited constructors have completed. It's return value is ignored.

```
:methods [ [name [param-types] return-type], ...]
```

The generated class automatically defines all of the non-private methods of its superclasses/interfaces. This parameter can be used to specify the signatures of additional methods of the generated class. Static methods can be specified with ^{:static true} in the signature's metadata. Do not repeat superclass/interface signatures here.

```
:main boolean
```

If supplied and true, a static public main function will be generated. It will pass each string of the String[] argument as a separate argument to a function called (str prefix main).

```
:factory name
```

If supplied, a (set of) public static factory function(s) will be created with the given name, and the same signature(s) as the constructor(s).

```
:state name
```

If supplied, a public final instance field with the given name will be created. You must supply an :init function in order to provide a value for the state. Note that, though final, the state can be a ref

or agent, supporting the creation of Java objects with transactional or asynchronous mutation semantics.

```
:exposes {:protected-field-name {:get name :set name}, ...}
```

Since the implementations of the methods of the generated class occur in Clojure functions, they have no access to the inherited protected fields of the superclass. This parameter can be used to generate public getter/setter methods exposing the protected field(s) for use in the implementation.

```
:exposes-methods {:super-method-name exposed-name, ...}
```

It is sometimes necessary to call the superclass' implementation of an overridden method. Those methods may be exposed and referred in the new method implementation by a local name.

```
:prefix string
```

Default: \"-\" Methods called e.g. Foo will be looked up in vars called prefixFoo in the implementing ns.

```
:impl-ns name
```

Default: the name of the current ns. Implementations of methods will be looked up in this namespace.

```
:load-impl-ns boolean
```

Default: true. Causes the static initializer for the generated class to reference the load code for the implementing namespace. Should be true when implementing-ns is the default, false if you intend to load the code via some other method."

```
{:added "1.0"}
```

```
[& options]
```

```
(when *compile-files*
```

```
  (let [options-map (into1 {} (map vec (partition 2 options)))
```

```
    [cname bytecode] (generate-class options-map)]
```

```
    (clojure.lang.Compiler/writeClassFile cname bytecode))))
```

```
;;;;;;;;;; gen-interface ;;;;;;;;
;; based on original contribution by Chris Houser
```

```
(defn- ^Type asm-type
```

"Returns an asm Type object for c, which may be a primitive class (such as Integer/TYPE), any other class (such as Double), or a fully-qualified class name given as a string or symbol (such as 'java.lang.String")

```
[c]
```

```

(if (or (instance? Class c) (prim->class c))
  (Type/getType (the-class c))
  (let [strx (str c)]
    (Type/getObjectType
      (.replace (if (some #{\.} strx)
                  strx
                  (str "java.lang." strx))
                  ". " "/"))))

(defn- generate-interface
  [{:keys [name extends methods]}]
  (let [iname (.replace (str name) ". " "/")
        cv (ClassWriter. ClassWriter/COMPUTE_MAXS)]
    (. cv visit Opcodes/V1_5 (+ Opcodes/ACC_PUBLIC
                                 Opcodes/ACC_ABSTRACT
                                 Opcodes/ACC_INTERFACE)
      iname nil "java/lang/Object"
      (when (seq extends)
        (into-array (map #(.getInternalName (asm-type %)) extends))))
      (add-annotations cv (meta name))
      (doseq [[mname pclasses rclass pmetas] methods]
        (let [mv (. cv visitMethod
                    (+ Opcodes/ACC_PUBLIC Opcodes/ACC_ABSTRACT)
                    (str mname)
                    (Type/getMethodDescriptor (asm-type rclass)
                                              (if pclasses
                                                (into-array Type (map asm-type pclasses))
                                                (make-array Type 0)))
                    nil nil)]
          (add-annotations mv (meta mname))
          (dotimes [i (count pmetas)]
            (add-annotations mv (nth pmetas i) i))
          (. mv visitEnd)))
        (. cv visitEnd)
        [iname (. cv toByteArray)])))

(defmacro gen-interface
  "When compiling, generates compiled bytecode for an interface with
  the given package-qualified :name (which, as all names in these
  parameters, can be a string or symbol), and writes the .class file
  to the *compile-path* directory. When not compiling, does nothing."
  
```

In all subsequent sections taking types, the primitive types can be referred to by their Java names (int, float etc), and classes in the java.lang package can be used without a package qualifier. All other classes must be fully qualified.

Options should be a set of key/value pairs, all except for :name are optional:

```

:name cname

The package-qualified name of the class to be generated

:extends [interface ...]

One or more interfaces, which will be extended by this interface.

:methods [ [name [param-types] return-type], ...]

This parameter is used to specify the signatures of the methods of
the generated interface. Do not repeat superinterface signatures
here."
{:added "1.0"}

[& options]
(let [options-map (apply hash-map options)
      [cname bytecode] (generate-interface options-map)]
  (if *compile-files*
      (clojure.lang.Compiler/writeClassFile cname bytecode)
      (.defineClass ^DynamicClassLoader
                    (deref clojure.lang.Compiler/LOADER)
                    (str (:name options-map)) bytecode options)))))

(comment

(defn gen-and-load-class
  "Generates and immediately loads the bytecode for the specified
  class. Note that a class generated this way can be loaded only once
  - the JVM supports only one class with a given name per
  classloader. Subsequent to generation you can import it into any
  desired namespaces just like any other class. See gen-class for a
  description of the options."
  {:added "1.0"}

[& options]
(let [options-map (apply hash-map options)
      [cname bytecode] (generate-class options-map)]
  (.. (clojure.lang.RT/getRootClassLoader)
      (.defineClass cname bytecode options)))))

)

```

11.8 gvec.clj

— gvec.clj —

```
\getchunk{Clojure Copyright}

;;; a generic vector implementation for vectors of primitives

(in-ns 'clojure.core)

;(set! *warn-on-reflection* true)

(deftype VecNode [edit arr])

(def EMPTY-NODE (VecNode. nil (object-array 32)))

(definterface IVecImpl
  (^int tailoff [])
  (arrayFor [^int i])
  (pushTail [^int level
            ^clojure.core.VecNode parent
            ^clojure.core.VecNode tailnode])
  (popTail [^int level node])
  (newPath [edit ^int level node])
  (doAssoc [^int level node ^int i val]))

(definterface ArrayManager
  (array [^int size])
  (^int alength [arr])
  (aclone [arr])
  (aget [arr ^int i])
  (aset [arr ^int i val]))

(deftype ArrayChunk [^clojure.core.ArrayManager am arr
                     ^int off ^int end]

  clojure.lang.Indexed
  (nth [_ i] (.aget am arr (+ off i)))

  (count [_] (- end off))

  clojure.lang.IChunk
  (dropFirst [_]
    (if (= off end)
        (throw (IllegalStateException. "dropFirst of empty chunk"))
        (new ArrayChunk am arr (inc off) end)))

  (reduce [_ f init]
```

```

(loop [ret init i off]
  (if (< i end)
      (recur (f ret (.aget am arr i)) (inc i))
      ret)))
)

(deftype VecSeq [^clojure.core.ArrayManager am
                 ^clojure.core.IVecImpl vec anode
                 ^int i ^int offset]
  :no-print true

  clojure.core.protocols.InternalReduce
  (internal-reduce
   [_ f val]
   (loop [result val
          aidx offset]
     (if (< aidx (count vec))
         (let [node (.arrayFor vec aidx)
               result
               (loop [result result node-idx (bit-and 0x1f aidx)]
                 (if (< node-idx (.alength am node))
                     (recur (f result (.aget am node node-idx))
                           (inc node-idx))
                     result))]
           (recur result (bit-and 0xffe0 (+ aidx 32))))
         result)))
   (recur result (bit-and 0xffe0 (+ aidx 32)))))

clojure.lang.ISeq
(first [_] (.aget am anode offset))
(next [this]
  (if (< (inc offset) (.alength am anode))
      (new VecSeq am vec anode i (inc offset))
      (.chunkedNext this)))
(more [this]
  (let [s (.next this)]
    (or s (clojure.lang.PersistentList/EMPTY))))
(cons [this o]
  (clojure.lang.Cons. o this))
(count [this]
  (loop [i 1
         s (next this)]
    (if s
        (if (instance? clojure.lang.Counted s)
            (+ i (.count s))
            (recur (inc i) (next s)))
        i)))
(equiv [this o]
  (cond
   (identical? this o) true
   (or (instance? clojure.lang.Sequential o)
       (instance? clojure.lang.ChunkedSeq o))
   (.equiv this o))))
```

```

(instance? java.util.List o))
(loop [me this
      you (seq o)]
  (if (nil? me)
      (nil? you)
      (and (clojure.lang.Util/equiv (first me) (first you))
            (recur (next me) (next you)))))

:else false))
(empty [])
clojure.lang.PersistentList/EMPTY)

clojure.lang.Seqable
(seq [this] this)

clojure.lang.IChunkedSeq
(chunkedFirst [_] (ArrayChunk. am anode offset (.alength am anode)))
(chunkedNext [_]
  (let [nexti (+ i (.alength am anode))]
    (when (< nexti (count vec))
      (new VecSeq am vec (.arrayFor vec nexti) nexti 0))))
(chunkedMore [this]
  (let [s (.chunkedNext this)]
    (or s (clojure.lang.PersistentList/EMPTY)))))

(defmethod print-method ::VecSeq [v w]
  ((get (methods print-method) clojure.lang.ISeq) v w))

(deftype Vec [^clojure.core.ArrayManager am
            ^int cnt
            ^int shift
            ^clojure.core.VecNode root tail _meta]
  Object
  (equals [this o]
    (cond
      (identical? this o) true
      (or (instance? clojure.lang.IPersistentVector o)
          (instance? java.util.RandomAccess o))
      (and (= cnt (count o))
           (loop [i (int 0)]
             (cond
               (= i cnt) true
               (.equals (.nth this i) (nth o i)) (recur (inc i))
               :else false)))
      (or (instance? clojure.lang.Sequential o)
          (instance? java.util.List o))
      (.equals (seq this) (seq o))
      :else false)))

;todo - cache

```

```

(hashCode [this]
  (loop [hash (int 1) i (int 0)]
    (if (= i cnt)
        hash
        (let [val (.nth this i)]
          (recur (unchecked-add-int (unchecked-multiply-int 31 hash)
                                     (closure.lang.Util/hash val))
                 (inc i)))))

closure.lang.Counted
(count [_] cnt)

closure.lang.IMeta
(meta [_] _meta)

closure.lang.IObj
(withMeta [_ m] (new Vec am cnt shift root tail m))

closure.lang.Indexed
(nth [this i]
  (let [a (.arrayFor this i)]
    (.aget am a (bit-and i (int 0x1f)))))
(nth [this i not-found]
  (let [z (int 0)]
    (if (and (>= i z) (< i (.count this)))
        (.nth this i)
        not-found)))

closure.lang.IPersistentCollection
(cons [this val]
  (if (< (- cnt (.tailoff this)) (int 32))
      (let [new-tail (.array am (inc (.alength am tail)))]
        (System/arraycopy tail 0 new-tail 0 (.alength am tail))
        (.aset am new-tail (.alength am tail) val)
        (new Vec am (inc cnt) shift root new-tail (meta this)))
      (let [tail-node (VecNode. (.edit root) tail)]
        (if (> (bit-shift-right cnt (int 5))
                  (bit-shift-left (int 1) shift)) ;overflow root?
            (let [new-root (VecNode. (.edit root) (object-array 32))]
              (doto ^objects (.arr new-root)
                (aset 0 root)
                (aset 1 (.newPath this (.edit root) shift tail-node)))
              (new Vec am
                  (inc cnt)
                  (+ shift (int 5))
                  new-root
                  (let [tl (.array am 1)] (.aset am tl 0 val) tl)
                  (meta this)))
            (new Vec am
                  (inc cnt)))
        ))))

```

```

shift
(.pushTail this shift root tail-node)
(let [tl (.array am 1)]
  (.aset am tl 0 val) tl) (meta this)))))

(empty [] (new Vec am 0 5 EMPTY-NODE (.array am 0) nil))
(equiv [this o]
  (cond
    (or (instance? clojure.lang.IPersistentVector o)
        (instance? java.util.RandomAccess o))
    (and (= cnt (count o))
         (loop [i (int 0)]
           (cond
             (= i cnt) true
             (= (.nth this i) (nth o i)) (recur (inc i))
             :else false)))
    (or (instance? clojure.lang.Sequential o)
        (instance? java.util.List o))
    (clojure.lang.Util/equiv (seq this) (seq o))
    :else false))

clojure.lang.IPersistentStack
(peek [this]
  (when (> cnt (int 0))
    (.nth this (dec cnt)))))

(pop [this]
  (cond
    (zero? cnt)
    (throw (IllegalStateException. "Can't pop empty vector"))
    (= 1 cnt)
    (new Vec am 0 5 EMPTY-NODE (.array am 0) (meta this)))
    (> (- cnt (.tailoff this)) 1)
    (let [new-tail (.array am (dec (.alength am tail)))]
      (System/arraycopy tail 0 new-tail 0 (.alength am new-tail))
      (new Vec am (dec cnt) shift root new-tail (meta this)))
    :else
    (let [new-tail (.arrayFor this (- cnt 2))
          new-root ^clojure.core.VecNode (.popTail this shift root)]
      (cond
        (nil? new-root)
        (new Vec am (dec cnt) shift EMPTY-NODE new-tail (meta this))
        (and (> shift 5) (nil? (aget ^objects (.arr new-root) 1)))
        (new Vec am
              (dec cnt)
              (- shift 5)
              (aget ^objects (.arr new-root) 0)
              new-tail
              (meta this)))
      :else

```

```

(new Vec am
  (dec cnt)
  shift
  new-root
  new-tail
  (meta this)))))

clojure.lang.IPersistentVector
(assocN [this i val]
  (cond
    (and (<= (int 0) i) (< i cnt))
      (if (>= i (.tailoff this))
        (let [new-tail (.array am (.alength am tail))]
          (System/arraycopy tail 0 new-tail 0 (.alength am tail))
          (.aset am new-tail (bit-and i (int 0x1f)) val)
          (new Vec am cnt shift root new-tail (meta this)))
        (new Vec am cnt shift
          (.doAssoc this shift root i val) tail (meta this)))
      (= i cnt) (.cons this val)
      :else (throw (IndexOutOfBoundsException.)))))

clojure.lang.Reversible
(rseq [this]
  (if (> (.count this) 0)
    (clojure.lang.APersistentVector$RSeq. this (dec (.count this)))
    nil))

clojure.lang.Associative
(assoc [this k v]
  (if (clojure.lang.Util/isInteger k)
    (.assocN this k v)
    (throw (IllegalArgumentException. "Key must be integer"))))
(containsKey [this k]
  (and (clojure.lang.Util/isInteger k)
    (<= 0 (int k))
    (< (int k) cnt)))
(entryAt [this k]
  (if (.containsKey this k)
    (clojure.lang.MapEntry. k (.nth this (int k)))
    nil))

clojure.lang.ILookup
(valAt [this k not-found]
  (if (clojure.lang.Util/isInteger k)
    (let [i (int k)]
      (if (and (>= i 0) (< i cnt))
        (.nth this i)
        not-found)
      not-found)))

```

```
(valAt [this k] (.valAt this k nil))

clojure.lang.IFn
(invocation [this k]
(if (clojure.lang.Util/isInteger k)
(let [i (int k)]
(if (and (>= i 0) (< i cnt))
(.nth this i)
(throw (IndexOutOfBoundsException.)))
(throw (IllegalArgumentException. "Key must be integer")))

clojure.lang.Seqable
(seq [this]
(if (zero? cnt)
nil
(VecSeq. am this (.arrayFor this 0) 0 0)))

clojure.lang.Sequential ;marker, no methods

clojure.core.IVecImpl
(tailoff []
(- cnt (.alength am tail)))

(arrayFor [this i]
(if (and (<= (int 0) i) (< i cnt))
(if (>= i (.tailoff this))
tail
(loop [node root level shift]
(if (zero? level)
(.arr node)
(recur
(aget ^objects (.arr node)
(bit-and (bit-shift-right i level) (int 0x1f)))
(- level (int 5))))))
(throw (IndexOutOfBoundsException.)))

(pushTail [this level parent tailnode]
(let [subidx (bit-and (bit-shift-right (dec cnt) level) (int 0x1f))
parent ^clojure.core.VecNode parent
ret (VecNode. (.edit parent) (aclone ^objects (.arr parent)))
node-to-insert
(if (= level (int 5))
tailnode
(let [child (aget ^objects (.arr parent) subidx)]
(if child
(.pushTail this (- level (int 5)) child tailnode)
(.newPath this (.edit root)
(- level (int 5)) tailnode))))]
(aset ^objects (.arr ret) subidx node-to-insert)
```

```

    ret))

(popTail [this level node]
  (let [node ^clojure.core.VecNode node
        subidx (bit-and
                  (bit-shift-right (- cnt (int 2)) level)
                  (int 0x1f))]
    (cond
      (> level 5)
      (let [new-child
            (.popTail this (- level 5)
                      (aget ^objects (.arr node) subidx))]
        (if (and (nil? new-child) (zero? subidx))
            nil
            (let [arr (aclone ^objects (.arr node))]
              (aset arr subidx new-child)
              (VecNode. (.edit root) arr))))
      (zero? subidx) nil
      :else (let [arr (aclone ^objects (.arr node))]
              (aset arr subidx nil)
              (VecNode. (.edit root) arr)))))

(newPath [this edit ^int level node]
  (if (zero? level)
      node
      (let [ret (VecNode. edit (object-array 32))]
        (aset ^objects (.arr ret) 0
              (.newPath this edit (- level (int 5)) node))
        ret)))

(doAssoc [this level node i val]
  (let [node ^clojure.core.VecNode node]
    (if (zero? level)
        ;on this branch, array will need val type
        (let [arr (.aclone am (.arr node))]
          (.aset am arr (bit-and i (int 0x1f)) val)
          (VecNode. (.edit node) arr))
        (let [arr (aclone ^objects (.arr node))
              subidx (bit-and (bit-shift-right i level) (int 0x1f))]
          (aset arr subidx
                (.doAssoc this (- level (int 5)) (aget arr subidx) i val))
          (VecNode. (.edit node) arr)))))

java.lang.Comparable
(compareTo [this o]
  (if (identical? this o)
      0
      (let [#^clojure.lang.IPersistentVector v
            (cast clojure.lang.IPersistentVector o)
            vcnt (.count v)]

```

```

(cond
  (< cnt vcnt) -1
  (> cnt vcnt) 1
  :else
    (loop [i (int 0)]
      (if (= i cnt)
          0
          (let [comp (clojure.lang.Util/compare (.nth this i)
                                              (.nth v i))]
            (if (= 0 comp)
                (recur (inc i))
                comp))))))

java.lang.Iterable
(iterator [this]
  (let [i (java.util.concurrent.atomic.AtomicInteger. 0)]
    (reify java.util.Iterator
      (hasNext [] (< (.get i) cnt))
      (next [] (.nth this (dec (.incrementAndGet i))))
      (remove [] (throw (UnsupportedOperationException.)))))

  java.util.Collection
  (contains [this o] (boolean (some #(= % o) this)))
  (containsAll [this c] (every? #(=.contains this %) c))
  (isEmpty [__] (zero? cnt))
  (toArray [this] (into-array Object this))
  (toArray [this arr]
    (if (>= (count arr) cnt)
        (do
          (dotimes [i cnt]
            (aset arr i (.nth this i)))
          arr)
        (into-array Object this)))
  (size [__] cnt)
  (add [__ o] (throw (UnsupportedOperationException.)))
  (addAll [__ c] (throw (UnsupportedOperationException.)))
  (clear [__] (throw (UnsupportedOperationException.)))
  (^boolean remove [__ o] (throw (UnsupportedOperationException.)))
  (removeAll [__ c] (throw (UnsupportedOperationException.)))
  (retainAll [__ c] (throw (UnsupportedOperationException.)))

  java.util.List
  (get [this i] (.nth this i))
  (indexOf [this o]
    (loop [i (int 0)]
      (cond
        (== i cnt) -1
        (= o (.nth this i)) i
        :else (recur (inc i)))))
  (lastIndexOf [this o]

```

```

(loop [i (dec cnt)]
  (cond
    (< i 0) -1
    (= o (.nth this i)) i
    :else (recur (dec i))))
(listIterator [this] (.listIterator this 0))
(listIterator [this i]
  (let [i (java.util.concurrent.atomic.AtomicInteger. i)]
    (reify java.util.ListIterator
      (hasNext [_] (< (.get i) cnt))
      (hasPrevious [_] (pos? i))
      (next [_] (.nth this (dec (.incrementAndGet i)))))
      (nextIndex [_] (.get i))
      (previous [_] (.nth this (.decrementAndGet i)))
      (previousIndex [_] (dec (.get i)))
      (add [_ e] (throw (UnsupportedOperationException.)))
      (remove [_] (throw (UnsupportedOperationException.)))
      (set [_ e] (throw (UnsupportedOperationException.))))
      (subList [this a z] (subvec this a z))
      (add [_ i o] (throw (UnsupportedOperationException.)))
      (addAll [_ i c] (throw (UnsupportedOperationException.)))
      (^Object remove [_ ^int i] (throw (UnsupportedOperationException.)))
      (set [_ i e] (throw (UnsupportedOperationException.)))))
    )
  )

(defmethod print-method ::Vec [v w]
  ((get (methods print-method) clojure.lang.IPersistentVector) v w))

(defmacro mk-am {:private true} [t]
  (let [garr (gensym)
        tgarr (with-meta garr {:tag (symbol (str t "s"))})]
    '(reify clojure.core.ArrayManager
       (array [_ size#] (symbol (str t "-array")) size#))
       (alength [_ ^garr] (alength ^tgarr))
       (aclone [_ ^garr] (aclone ^tgarr))
       (aget [_ ^garr i#] (aget ^tgarr i#))
       (aset [_ ^garr i# val#] (aset ^tgarr i# (t val#)))))

(def ^{:private true} ams
  {:int (mk-am int)
   :long (mk-am long)
   :float (mk-am float)
   :double (mk-am double)
   :byte (mk-am byte)
   :short (mk-am short)
   :char (mk-am char)
   :boolean (mk-am boolean}})

(defn vector-of
  "Creates a new vector of a single primitive type t, where t is one"

```

```

of :int :long :float :double :byte :short :char or :boolean. The
resulting vector complies with the interface of vectors in general,
but stores the values unboxed internally."
{:added "1.2"}
[t]
(let [am ^clojure.core.ArrayManager (ams t)]
  (Vec. am 0 5 EMPTY-NODE (.array am 0) nil)))

```

11.9 inspector.clj

— inspector.clj —

```

\getchunk{Clojure Copyright}

(ns ^{:doc "Graphical object inspector for Clojure data structures."
      :author "Rich Hickey"}
  clojure.inspector
  (:import
    (java.awt BorderLayout)
    (java.awt.event ActionEvent ActionListener)
    (javax.swing.tree TreeModel)
    (javax.swing.table TableModel AbstractTableModel)
    (javax.swing JPanel JTree JTable JScrollPane JFrame
      JToolBar JButton SwingUtilities)))

```

```

(defn atom? [x]
  (not (coll? x)))

(defn collection-tag [x]
  (cond
    (instance? java.util.Map$Entry x) :entry
    (instance? java.util.Map x) :map
    (sequential? x) :seq
    :else :atom))

(defmulti is-leaf collection-tag)
(defmulti get-child (fn [parent index] (collection-tag parent)))
(defmulti get-child-count collection-tag)

(defmethod is-leaf :default [node]
  (atom? node))
(defmethod get-child :default [parent index]
  (nth parent index))
(defmethod get-child-count :default [parent]
  (count parent))

```

```

(defmethod is-leaf :entry [e]
  (is-leaf (val e)))
(defmethod get-child :entry [e index]
  (get-child (val e) index))
(defmethod get-child-count :entry [e]
  (count (val e)))

(defmethod is-leaf :map [m]
  false)
(defmethod get-child :map [m index]
  (nth (seq m) index))

(defn tree-model [data]
  (proxy [TreeModel] []
    (getRoot [] data)
    (addTreeModelListener [treeModelListener])
    (getChild [parent index]
      (get-child parent index))
    (getChildCount [parent]
      (get-child-count parent))
    (isLeaf [node]
      (is-leaf node))
    (valueForPathChanged [path newValue])
    (getIndexOfChild [parent child]
      -1)
    (removeTreeModelListener [treeModelListener])))

(defn old-table-model [data]
  (let [row1 (first data)
        colcnt (count row1)
        cnt (count data)
        vals (if (map? row1) vals identity)]
    (proxy [TableModel] []
      (addTableModelListener [tableModelListener])
      (getColumnClass [columnIndex] Object)
      (getColumnCount [] colcnt)
      (getColumnName [columnIndex]
        (if (map? row1)
          (name (nth (keys row1) columnIndex))
          (str columnIndex)))
      (getRowCount [] cnt)
      (getValueAt [rowIndex columnIndex]
        (nth (vals (nth data rowIndex)) columnIndex))
      (isCellEditable [rowIndex columnIndex] false)
      (removeTableModelListener [tableModelListener]))))

(defn inspect-tree
  "creates a graphical (Swing) inspector on the supplied hierarchical"

```

```

data"
{:added "1.0"}
[data]
(doto (JFrame. "Clojure Inspector")
  (.add ( JScrollPane. (JTree. (tree-model data))))
  (.setSize 400 600)
  (.setVisible true)))

(defn inspect-table
  "creates a graphical (Swing) inspector on the supplied regular
  data, which must be a sequential data structure of data structures
  of equal length"
{:added "1.0"}
[data]
(doto (JFrame. "Clojure Inspector")
  (.add ( JScrollPane. (JTable. (old-table-model data))))
  (.setSize 400 600)
  (.setVisible true)))

(defmulti list-provider class)

(defmethod list-provider :default [x]
  {:nrows 1
   :get-value (fn [i] x) :get-label (fn [i] (.getName (class x)))}

(defmethod list-provider java.util.List [c]
  (let [v (if (vector? c) c (vec c))]
    {:nrows (count v)
     :get-value (fn [i] (v i))
     :get-label (fn [i] i)}))

(defmethod list-provider java.util.Map [c]
  (let [v (vec (sort (map (fn [[k v]] (vector k v)) c)))]
    {:nrows (count v)
     :get-value (fn [i] ((v i) 1))
     :get-label (fn [i] ((v i) 0))}))

(defn list-model [provider]
  (let [{:keys [nrows get-value get-label]} provider]
    (proxy [AbstractTableModel] []
      (getColumnCount [] 2)
      (getRowCount [] nrows)
      (getValueAt [rowIndex columnIndex]
        (cond
          (= 0 columnIndex) (get-label rowIndex)
          (= 1 columnIndex) (print-str (get-value rowIndex)))))))

(defmulti table-model class)

```

```
(defmethod table-model :default [x]
  (proxy [AbstractTableModel] []
    (getRowCount [] 1)
    (getColumnCount [] 2)
    (getValueAt [rowIndex columnIndex]
      (if (zero? columnIndex)
        (class x)
        x))))
  ;(defn make-inspector [x]
  ;  (agent {:frame frame :data x :parent nil :index 0}))

(defn inspect
  "creates a graphical (Swing) inspector on the supplied object"
  {:added "1.0"}
  [x]
  (doto (JFrame. "Clojure Inspector")
    (.add
      (doto ( JPanel. (BorderLayout.))
        (.add (doto (JToolBar.)
          (.add (JButton. "Back"))
          (.addSeparator)
          (.add (JButton. "List"))
          (.add (JButton. "Table"))
          (.add (JButton. "Bean"))
          (.add (JButton. "Line"))
          (.add (JButton. "Bar"))
          (.addSeparator)
          (.add (JButton. "Prev"))
          (.add (JButton. "Next"))))
        BorderLayout/NORTH)
        (.add
          ( JScrollPane.
            (doto (JTable. (list-model (list-provider x)))
              (.setAutoResizeMode JTable/AUTO_RESIZE_LAST_COLUMN)))
          BorderLayout/CENTER)))
      (.setSize 400 400)
      (.setVisible true)))

  (comment
    (load-file "src/inspector.clj")
    (refer 'inspector)
    (inspect-tree {:a 1 :b 2 :c [1 2 3 {:d 4 :e 5 :f [6 7 8]}]})
    (inspect-table [[1 2 3] [4 5 6] [7 8 9] [10 11 12]])
  ))
```

11.10 browse.clj

— browse.clj —

```
\getchunk{Clojure Copyright}

(ns
  ^{:author "Christophe Grand",
    :doc "Start a web browser from Clojure"}
  clojure.java.browser
  (:require [clojure.java.shell :as sh])
  (:import (java.net URI)))

(defn- macosx? []
  (-> "os.name" System/getProperty .toLowerCase
       (.startsWith "mac os x")))

(def ^:dynamic *open-url-script* (when (macosx?) "/usr/bin/open"))

(defn- open-url-in-browser
  "Opens url (a string) in the default system web browser. May not
  work on all platforms. Returns url on success, nil if not
  supported."
  [url]
  (try
    (when (clojure.lang.Reflector/invokeStaticMethod "java.awt.Desktop"
      "isDesktopSupported" (to-array nil))
      (-> (clojure.lang.Reflector/invokeStaticMethod "java.awt.Desktop"
        "getDesktop" (to-array nil))
          (.browse (URI. url)))
      url)
    (catch ClassNotFoundException e
      nil)))

(defn- open-url-in-swing
  "Opens url (a string) in a Swing window."
  [url]
  ; the implementation of this function resides in another
  ; namespace to be loaded "on demand"
  ; this fixes a bug on mac os x where the process turns into a GUI app
  ; see http://code.google.com/p/clojure-contrib/issues/detail?id=32
  (require 'clojure.java.browser-ui)
  ((find-var 'clojure.java.browser-ui/open-url-in-swing) url))

(defn browse-url
```

```
"Open url in a browser"
{:added "1.2"}
[url]
(or (open-url-in-browser url)
     (when *open-url-script* (sh/sh *open-url-script* (str url)) true)
     (open-url-in-swing url)))
```

11.11 browse.ui.clj

— browse.ui.clj —

```
\getchunk{Clojure Copyright}

(ns
  ^{:author "Christophe Grand",
    :doc "Helper namespace for clojure.java.browser.
          Prevents console apps from becoming GUI unnecessarily."}
  clojure.java.browser-ui)

(defn- open-url-in-swing
  [url]
  (let [htmpane (javax.swing.JEditorPane. url)]
    (.setEditable htmpane false)
    (.addHyperlinkListener htmpane
      (proxy [javax.swing.event.HyperlinkListener] []
        (hyperlinkUpdate [#^javax.swing.event.HyperlinkEvent e]
          (when
            (= (.getEventType e)
                (. javax.swing.event.HyperlinkEvent$EventType ACTIVATED))
            (if (instance?
                  javax.swing.text.html.HTMLFrameHyperlinkEvent e)
                (-> htmpane .getDocument
                  (.processHTMLFrameHyperlinkEvent e)))
              (.setPage htmpane (.getURL e)))))))
    (doto (javax.swing.JFrame.)
      (.setContentPane (javax.swing.JScrollPane. htmpane))
      (.setBounds 32 32 700 900)
      (.show))))
```

11.12 io.clj

— io.clj —

```
\getchunk{Clojure Copyright}

(ns
  ^{:author "Stuart Sierra, Chas Emerick, Stuart Halloway",
    :doc "This file defines polymorphic I/O utility functions
          for Clojure."}
  clojure.java.io
  (:import
    (java.io Reader InputStreamReader PushbackReader
              BufferedReader File OutputStream
              OutputStreamWriter BufferedWriter Writer
              FileInputStream FileOutputStream ByteArrayOutputStream
              StringReader ByteArrayInputStream
              BufferedInputStream BufferedOutputStream
              CharArrayReader Closeable)
    (java.net URI URL MalformedURLException Socket)))

(def
  ^{:doc "Type object for a Java primitive byte array."
    :private true
    }
  byte-array-type (class (make-array Byte/TYPE 0)))

(def
  ^{:doc "Type object for a Java primitive char array."
    :private true}
  char-array-type (class (make-array Character/TYPE 0)))

(defprotocol ^{:added "1.2"} Coercions
  "Coerce between various 'resource-namish' things."
  (^{:tag java.io.File, :added "1.2"}
    as-file [x] "Coerce argument to a file.")
  (^{:tag java.net.URL, :added "1.2"}
    as-url [x] "Coerce argument to a URL.))

(extend-protocol Coercions
  nil
  (as-file []) nil
  (as-url []) nil)

  String
  (as-file [s] (File. s))
  (as-url [s] (URL. s))
```

```

File
(as-file [f] f)
(as-url [f] (.toURL f))

URL
(as-url [u] u)
(as-file [u]
  (if (= "file" (.getProtocol u))
    (as-file (.getPath u))
    (throw (IllegalArgumentException. (str "Not a file: " u)))))

URI
(as-url [u] (.toURL u))
(as-file [u] (as-file (as-url u)))

(defprotocol {:added "1.2"} IOFactory
  "Factory functions that create ready-to-use, buffered versions of
  the various Java I/O stream types, on top of anything that can
  be unequivocally converted to the requested kind of stream.

Common options include

:append      true to open stream in append mode
:encoding   string name of encoding to use, e.g. \"UTF-8\".

Callers should generally prefer the higher level API provided by
reader, writer, input-stream, and output-stream."
({:added "1.2"} make-reader [x opts]
  "Creates a BufferedReader. See also IOFactory docs.")
({:added "1.2"} make-writer [x opts]
  "Creates a BufferedWriter. See also IOFactory docs.")
({:added "1.2"} make-input-stream [x opts]
  "Creates a BufferedInputStream. See also IOFactory docs.")
({:added "1.2"} make-output-stream [x opts]
  "Creates a BufferedOutputStream. See also IOFactory docs."))

(defn ^Reader reader
  "Attempts to coerce its argument into an open java.io.Reader.
  Default implementations always return a java.io.BufferedReader.

  Default implementations are provided for Reader, BufferedReader,
  InputStream, File, URI, URL, Socket, byte arrays, character arrays,
  and String.

  If argument is a String, it tries to resolve it first as a URI, then
  as a local file name. URIs with a 'file' protocol are converted to
  local file names.

  Should be used inside with-open to ensure the Reader is properly
  closed."

```

```

{:added "1.2"}
[x & opts]
(make-reader x (when opts (apply hash-map opts)))))

(defn ^Writer writer
  "Attempts to coerce its argument into an open java.io.Writer.
  Default implementations always return a java.io.BufferedWriter.

  Default implementations are provided for Writer, BufferedWriter,
  OutputStream, File, URI, URL, Socket, and String.

  If the argument is a String, it tries to resolve it first as a URI,
  then as a local file name. URIs with a 'file' protocol are
  converted to local file names.

  Should be used inside with-open to ensure the Writer is properly
  closed."
{:added "1.2"}
[x & opts]
(make-writer x (when opts (apply hash-map opts)))))

(defn ^InputStream input-stream
  "Attempts to coerce its argument into an open java.io.InputStream.
  Default implementations always return a java.io.BufferedReader.

  Default implementations are defined for OutputStream, File, URI, URL,
  Socket, byte array, and String arguments.

  If the argument is a String, it tries to resolve it first as a URI,
  then as a local file name. URIs with a 'file' protocol are
  converted to local file names.

  Should be used inside with-open to ensure the InputStream is
  properly closed."
{:added "1.2"}
[x & opts]
(make-input-stream x (when opts (apply hash-map opts)))))

(defn ^OutputStream output-stream
  "Attempts to coerce its argument into an open java.io.OutputStream.
  Default implementations always return a java.io.BufferedOutputStream.

  Default implementations are defined for OutputStream, File, URI, URL,
  Socket, and String arguments.

  If the argument is a String, it tries to resolve it first as a URI,
  then as a local file name. URIs with a 'file' protocol are
  converted to local file names.

  Should be used inside with-open to ensure the OutputStream is

```

```

properly closed."
{:added "1.2"}
[x & opts]
(make-output-stream x (when opts (apply hash-map opts)))

(defn- ^Boolean append? [opts]
  (boolean (:append opts)))

(defn- ^String encoding [opts]
  (or (:encoding opts) "UTF-8"))

(defn- buffer-size [opts]
  (or (:buffer-size opts) 1024))

(def default-streams-impl
  {:make-reader
   (fn [x opts] (make-reader (make-input-stream x opts) opts))
   :make-writer
   (fn [x opts] (make-writer (make-output-stream x opts) opts))
   :make-input-stream
   (fn [x opts]
     (throw (IllegalArgumentException.
             (str "Cannot open <" (pr-str x) "> as an InputStream."))))
   :make-output-stream
   (fn [x opts]
     (throw (IllegalArgumentException.
             (str "Cannot open <" (pr-str x) "> as an OutputStream."))))})

(defn- inputstream->reader
  [^InputStream is opts]
  (make-reader (InputStreamReader. is (encoding opts)) opts))

(defn- outputstream->writer
  [^OutputStream os opts]
  (make-writer (OutputStreamWriter. os (encoding opts)) opts))

(extend BufferedInputStream
  IFactory
  (assoc default-streams-impl
    :make-input-stream (fn [x opts] x)
    :make-reader inputstream->reader))

(extend InputStream
  IFactory
  (assoc default-streams-impl
    :make-input-stream (fn [x opts] (BufferedInputStream. x))
    :make-reader inputstream->reader))

(extend Reader
  IFactory

```

```

  (assoc default-streams-impl
        :make-reader (fn [x opts] (BufferedReader. x)))))

(extend BufferedReader
  IOFactory
  (assoc default-streams-impl
        :make-reader (fn [x opts] x)))

(extend Writer
  IOFactory
  (assoc default-streams-impl
        :make-writer (fn [x opts] (BufferedWriter. x)))))

(extend BufferedWriter
  IOFactory
  (assoc default-streams-impl
        :make-writer (fn [x opts] x)))

(extend OutputStream
  IOFactory
  (assoc default-streams-impl
        :make-output-stream (fn [x opts] (BufferedOutputStream. x))
        :make-writer outputstream->writer))

(extend BufferedOutputStream
  IOFactory
  (assoc default-streams-impl
        :make-output-stream (fn [x opts] x)
        :make-writer outputstream->writer))

(extend File
  IOFactory
  (assoc default-streams-impl
        :make-input-stream
        (fn [^File x opts] (make-input-stream (FileInputStream. x) opts)))
        :make-output-stream
        (fn [^File x opts]
          (make-output-stream (FileOutputStream. x (append? opts)) opts)))))

(extend URL
  IOFactory
  (assoc default-streams-impl
        :make-input-stream (fn [^URL x opts]
                            (make-input-stream
                              (if (= "file" (.getProtocol x))
                                  (FileInputStream. (.getPath x))
                                  (.openStream x)) opts)))
        :make-output-stream
        (fn [^URL x opts]
          (if (= "file" (.getProtocol x))
              (.openStream x)))))
```

```

(make-output-stream (File. (.getPath x)) opts)
(throw (IllegalArgumentException.
        (str "Can not write to non-file URL <" x ">")))))

(extend URI
IOFactory
(assoc default-streams-impl
:make-input-stream
(fn [^URI x opts] (make-input-stream (.toURL x) opts))
:make-output-stream
(fn [^URI x opts] (make-output-stream (.toURL x) opts)))))

(extend String
IOFactory
(assoc default-streams-impl
:make-input-stream (fn [^String x opts]
(try
(make-input-stream (URL. x) opts)
(catch MalformedURLException e
(make-input-stream (File. x) opts))))
:make-output-stream (fn [^String x opts]
(try
(make-output-stream (URL. x) opts)
(catch MalformedURLException err
(make-output-stream (File. x) opts))))))

(extend Socket
IOFactory
(assoc default-streams-impl
:make-input-stream
(fn [^Socket x opts]
(make-input-stream (.getInputStream x) opts))
:make-output-stream
(fn [^Socket x opts]
(make-output-stream (.getOutputStream x) opts)))))

(extend byte-array-type
IOFactory
(assoc default-streams-impl
:make-input-stream
(fn [x opts] (make-input-stream (ByteArrayInputStream. x) opts)))))

(extend char-array-type
IOFactory
(assoc default-streams-impl
:make-reader (fn [x opts] (make-reader (CharArrayReader. x) opts)))))

(extend Object
IOFactory
default-streams-impl)

```

```

(defmulti
  #^{:doc "Internal helper for copy"
    :private true
    :arglists '([input output opts])}
  do-copy
  (fn [input output opts] [(type input) (type output)]))

(defmethod do-copy [InputStream OutputStream]
  [#^InputStream input #^OutputStream output opts]
  (let [buffer (make-array Byte/TYPE (buffer-size opts))]
    (loop []
      (let [size (.read input buffer)]
        (when (pos? size)
          (do (.write output buffer 0 size)
              (recur)))))))

(defmethod do-copy [InputStream Writer]
  [#^InputStream input #^Writer output opts]
  (let [#"^B" buffer (make-array Byte/TYPE (buffer-size opts))]
    (loop []
      (let [size (.read input buffer)]
        (when (pos? size)
          (let [chars
                (.toCharArray (String. buffer 0 size (encoding opts)))]
            (do (.write output chars)
                (recur)))))))

(defmethod do-copy [InputStream File]
  [#^InputStream input #^File output opts]
  (with-open [out (FileOutputStream. output)]
    (do-copy input out opts)))

(defmethod do-copy [Reader OutputStream]
  [#^Reader input #^OutputStream output opts]
  (let [#"^C" buffer (make-array Character/TYPE (buffer-size opts))]
    (loop []
      (let [size (.read input buffer)]
        (when (pos? size)
          (let [bytes
                (.getBytes (String. buffer 0 size) (encoding opts))]
            (do (.write output bytes)
                (recur)))))))

(defmethod do-copy [Reader Writer] [#^Reader input #^Writer output opts]
  (let [#"^C" buffer (make-array Character/TYPE (buffer-size opts))]
    (loop []
      (let [size (.read input buffer)]
        (when (pos? size)
          (do (.write output buffer 0 size)
              (recur)))))))

```

```

        (recur))))))

(defmethod do-copy [Reader File] [#^Reader input #^File output opts]
  (with-open [out (FileOutputStream. output)]
    (do-copy input out opts)))

(defmethod do-copy [File OutputStream]
  [#^File input #^OutputStream output opts]
  (with-open [in (FileInputStream. input)]
    (do-copy in output opts)))

(defmethod do-copy [File Writer] [#^File input #^Writer output opts]
  (with-open [in (FileInputStream. input)]
    (do-copy in output opts)))

(defmethod do-copy [File File] [#^File input #^File output opts]
  (with-open [in (FileInputStream. input)
             out (FileOutputStream. output)]
    (do-copy in out opts)))

(defmethod do-copy [String OutputStream]
  [#^String input #^OutputStream output opts]
  (do-copy (StringReader. input) output opts))

(defmethod do-copy [String Writer] [#^String input #^Writer output opts]
  (do-copy (StringReader. input) output opts))

(defmethod do-copy [String File] [#^String input #^File output opts]
  (do-copy (StringReader. input) output opts))

(defmethod do-copy [char-array-type OutputStream]
  [input #^OutputStream output opts]
  (do-copy (CharArrayReader. input) output opts))

(defmethod do-copy [char-array-type Writer] [input #^Writer output opts]
  (do-copy (CharArrayReader. input) output opts))

(defmethod do-copy [char-array-type File] [input #^File output opts]
  (do-copy (CharArrayReader. input) output opts))

(defmethod do-copy [byte-array-type OutputStream]
  [#^"[B" input #^OutputStream output opts]
  (do-copy (ByteArrayInputStream. input) output opts))

(defmethod do-copy [byte-array-type Writer]
  [#^"[B" input #^Writer output opts]
  (do-copy (ByteArrayInputStream. input) output opts))

(defmethod do-copy [byte-array-type File]
  [#^"[B" input #^Writer output opts]

```

```
(do-copy (ByteArrayInputStream. input) output opts))

(defn copy
  "Copies input to output. Returns nil or throws IOException.
  Input may be an InputStream, Reader, File, byte[], or String.
  Output may be an OutputStream, Writer, or File.

Options are key/value pairs and may be one of

:buffer-size  buffer size to use, default is 1024.
:encoding      encoding to use if converting between
               byte and char streams.

Does not close any streams except those it opens itself
(on a File)."
{:added "1.2"}
[input output & opts]
(do-copy input output (when opts (apply hash-map opts)))

(defn ^String as-relative-path
  "Take an as-file-able thing and return a string if it is
  a relative path, else IllegalArgumentException."
{:added "1.2"}
[x]
(let [^File f (as-file x)]
  (if (.isAbsolute f)
    (throw (IllegalArgumentException.
            (str f " is not a relative path")))
    (.getPath f)))

(defn ^File file
  "Returns a java.io.File, passing each arg to as-file. Multiple-arg
  versions treat the first argument as parent and subsequent args as
  children relative to the parent."
{:added "1.2"}
([arg]
 (as-file arg))
([parent child]
 (File. ^File (as-file parent) ^String (as-relative-path child)))
([parent child & more]
 (reduce file (file parent child) more)))

(defn delete-file
  "Delete file f. Raise an exception if it fails unless silently is
  true."
{:added "1.2"}
[f & [silently]]
(or (.delete (file f))
    silently
    (throw (java.io.IOException. (str "Couldn't delete " f)))))
```

```
(defn make-parents
  "Given the same arg(s) as for file, creates all parent directories of
  the file they represent."
  {:added "1.2"}
  [f & more]
  (.mkdirs (.getParentFile ^File (apply file f more)))))

(defn ^URL resource
  "Returns the URL for a named resource. Use the context class loader
  if no loader is specified."
  {:added "1.2"}
  ([n] (resource n (.getContextClassLoader (Thread/currentThread))))
  ([n ^ClassLoader loader] (.getResource loader n)))
  _____
```

11.13 javadoc.clj

— javadoc.clj —

```
\getchunk{Clojure Copyright}

(ns
  ^{:author "Christophe Grand, Stuart Sierra",
    :doc "A repl helper to quickly open javadocs."}
  clojure.java.javadoc
  (:use [clojure.java.browser :only (browse-url)] )
  (:import
    (java.io File)))

(def ^:dynamic *feeling-lucky-url*
  "http://www.google.com/search?btnI=I%27m%20Feeling%20Lucky&q=allinurl:")
(def ^:dynamic *feeling-lucky* true)

(def ^:dynamic *local-javadocs* (ref (list)))

(def ^:dynamic *core-java-api*
  (if (= "1.5" (System/getProperty "java.specification.version"))
    "http://java.sun.com/j2se/1.5.0/docs/api/"
    "http://java.sun.com/javase/6/docs/api/"))

(def ^:dynamic *remote-javadocs*
  (ref (sorted-map
        "java." *core-java-api*
        "javax." *core-java-api*
        "org.ietf.jgss." *core-java-api*))
```

```

"org.omg." *core-java-api*
"org.w3c.dom." *core-java-api*
"org.xml.sax." *core-java-api*
"org.apache.commons.codec."
  "http://commons.apache.org/codec/api-release/"
"org.apache.commons.io."
  "http://commons.apache.org/io/api-release/"
"org.apache.commons.lang."
  "http://commons.apache.org/lang/api-release/"))

(defn add-local-javadoc
  "Adds to the list of local Javadoc paths."
  {:added "1.2"}
  [path]
  (dosync (commute *local-javadocs* conj path)))

(defn add-remote-javadoc
  "Adds to the list of remote Javadoc URLs. package-prefix is the
  beginning of the package name that has docs at this URL."
  {:added "1.2"}
  [package-prefix url]
  (dosync (commute *remote-javadocs* assoc package-prefix url)))

(defn- javadoc-url
  "Searches for a URL for the given class name. Tries
  *local-javadocs* first, then *remote-javadocs*. Returns a string."
  {:tag String,
   :added "1.2"}
  [^String classname]
  (let [file-path (.replace classname \. File/separatorChar)
        url-path (.replace classname \. \/)"]
    (if-let [file ^File
             (first
              (filter #(.exists ^File %)
                     (map #(File. (str %) (str file-path ".html"))
                           @*local-javadocs*)))])
        (-> file .toURI str)
        ;; If no local file, try remote URLs:
        (or (some (fn [[prefix url]]
                   (when (.startsWith classname prefix)
                     (str url url-path ".html")))
                  @*remote-javadocs*))
            ;; if *feeling-lucky* try a web search
            (when *feeling-lucky*
              (str *feeling-lucky-url* url-path ".html"))))))
))

(defn javadoc
  "Opens a browser window displaying the javadoc for the argument.
  Tries *local-javadocs* first, then *remote-javadocs*."
  {:added "1.2"}

```

```
[class-or-object]
(let [^Class c (if (instance? Class class-or-object)
                     class-or-object
                     (class class-or-object))]
  (if-let [url (javadoc-url (.getName c))]
    (browse-url url)
    (println "Could not find Javadoc for" c))))
```

11.14 shell.clj

— shell.clj —

```
\getchunk{Clojure Copyright}

(ns
  ^{:author "Chris Houser, Stuart Halloway",
    :doc "Conveniently launch a sub-process providing its stdin and
collecting its stdout"}
  clojure.java.shell
  (:use [clojure.java.io :only (as-file copy)])
  (:import
    (java.io OutputStreamWriter ByteArrayOutputStream StringWriter)
    (java.nio.charset Charset)))

(def ^:dynamic *sh-dir* nil)
(def ^:dynamic *sh-env* nil)

(defmacro with-sh-dir
  "Sets the directory for use with sh, see sh for details."
  {:added "1.2"}
  [dir & forms]
  `'(binding [*sh-dir* ~dir]
      ~@forms))

(defmacro with-sh-env
  "Sets the environment for use with sh, see sh for details."
  {:added "1.2"}
  [env & forms]
  `'(binding [*sh-env* ~env]
      ~@forms))

(defn- aconcat
  "Concatenates arrays of given type."
  [type & xs]
  (let [target (make-array type (apply + (map count xs)))]
```

```

(loop [i 0 idx 0]
  (when-let [a (nth xs i nil)]
    (System/arraycopy a 0 target idx (count a))
    (recur (inc i) (+ idx (count a)))))

(defn- parse-args
  [args]
  (let [default-encoding "UTF-8" ; see sh doc string
        default-opts {:out-enc default-encoding
                      :in-enc default-encoding
                      :dir *sh-dir*
                      :env *sh-env*}
        [cmd opts] (split-with string? args)]
    [cmd (merge default-opts (apply hash-map opts))]))

(defn- ^"[Ljava.lang.String;" as-env-strings
  "Helper so that callers can pass a Clojure map for the :env to sh."
  [arg]
  (cond
    (nil? arg) nil
    (map? arg)
      (into-array String (map (fn [[k v]] (str (name k) "=" v)) arg))
    true arg))

(defn- stream-to-bytes
  [in]
  (with-open [bout (ByteArrayOutputStream.)]
    (copy in bout)
    (.toByteArray bout)))

(defn- stream-to-string
  ([[in] (stream-to-string in (.name (Charset/defaultCharset))))]
  ([in enc]
   (with-open [bout (StringWriter.)]
     (copy in bout :encoding enc)
     (.toString bout)))))

(defn- stream-to-enc
  [stream enc]
  (if (= enc :bytes)
    (stream-to-bytes stream)
    (stream-to-string stream enc)))

(defn sh
  "Passes the given strings to Runtime.exec() to launch a sub-process.

Options are

:in      may be given followed by a String or byte array specifying"

```

```

    input to be fed to the sub-process's stdin.
:in-enc option may be given followed by a String, used as a
character encoding name (for example \"UTF-8\" or
\"ISO-8859-1\") to convert the input string specified by
the :in option to the sub-process's stdin. Defaults to
UTF-8. If the :in option provides a byte array, then the
bytes are passed unencoded, and this option is ignored.
:out-enc option may be given followed by :bytes or a String. If a
String is given, it will be used as a character encoding
name (for example \"UTF-8\" or \"ISO-8859-1\") to convert
the sub-process's stdout to a String which is returned.
If :bytes is given, the sub-process's stdout will be stored
in a byte array and returned. Defaults to UTF-8.
:env    override the process env with a map (or the underlying Java
String[] if you are a masochist).
:dir    override the process dir with a String or java.io.File.

```

You can bind :env or :dir for multiple operations using with-sh-env and with-sh-dir.

```

sh returns a map of
  :exit => sub-process's exit code
  :out  => sub-process's stdout (as byte[] or String)
  :err  =>
        sub-process's stderr (String via platform default encoding)"
{:added "1.2"}
[& args]
(let [[cmd opts] (parse-args args)
      proc (.exec (Runtime/getRuntime)
                  ^" [Ljava.lang.String;" (into-array cmd)
                  (as-env-strings (:env opts))
                  (as-file (:dir opts)))
      {:keys [in in-enc out-enc]} opts]
(if in
  (future
    (if (instance? (class (byte-array 0)) in)
        (with-open [os (.getOutputStream proc)]
          (.write os ^"[B" in))
        (with-open
          [osw (OutputStreamWriter. (.getOutputStream proc)
                                    ^String in-enc)]
          (.write osw ^String in))))
    (.close (.getOutputStream proc))
    (with-open [stdout (.getInputStream proc)
                stderr (.getErrorStream proc)]
      (let [out (future (stream-to-enc stdout out-enc))
            err (future (stream-to-string stderr))
            exit-code (.waitFor proc)]
        {:exit exit-code :out @out :err @err})))))

```

```
(comment

  (println (sh "ls" "-l"))
  (println (sh "ls" "-l" "/no-such-thing"))
  (println (sh "sed" "s/[aeiou]/oo/g" :in "hello there\n"))
  (println (sh "cat" :in "x\u25bax\n"))
  (println (sh "echo" "x\u25bax"))
  ; reads 4 single-byte chars
  (println (sh "echo" "x\u25bax" :out-enc "ISO-8859-1"))
  ; reads binary file into bytes[]
  (println (sh "cat" "myimage.png" :out-enc :bytes))
  (println (sh "cmd" "/c dir 1>&2"))

)
```

11.15 main.clj

— main.clj —

```
\getchunk{Clojure Copyright}

;; Originally contributed by Stephen C. Gilardi

(ns ^{:doc "Top-level main function for Clojure REPL and scripts."
      :author "Stephen C. Gilardi and Rich Hickey"}
  clojure.main
  (:refer-clojure :exclude [with-bindings])
  (:import (clojure.lang Compiler Compiler$CompilerException
                        LineNumberingPushbackReader RT))
  (:use [clojure.repl :only (demunge root-cause stack-element-str)]))

(declare main)

(defmacro with-bindings
  "Executes body in the context of thread-local bindings for several
  vars that often need to be set!: *ns* *warn-on-reflection*
  *math-context* *print-meta* *print-length* *print-level*
  *compile-path* *command-line-args* *1 *2 *3 *e*
  [& body]
  '(binding [*ns* *ns*
            *warn-on-reflection* *warn-on-reflection*
            *math-context* *math-context*
            *print-meta* *print-meta*
            *print-length* *print-length*
            *print-level* *print-level*
```

```

*compile-path*
  (System/getProperty "clojure.compile.path" "classes")
*command-line-args* *command-line-args*
*unchecked-math* *unchecked-math*
*assert* *assert*
*1 nil
*2 nil
*3 nil
*e nil]
~@body))

(defn repl-prompt
  "Default :prompt hook for repl"
  []
  (printf "%s=> " (ns-name *ns*)))

(defn skip-if-eol
  "If the next character on stream s is a newline, skips it, otherwise
leaves the stream untouched. Returns :line-start, :stream-end, or
:body to indicate the relative location of the next character on s.
The stream must either be an instance of LineNumberingPushbackReader
or duplicate its behavior of both supporting .unread and collapsing
all of CR, LF, and CRLF to a single \\newline."
  [s]
  (let [c (.read s)]
    (cond
      (= c (int \newline)) :line-start
      (= c -1) :stream-end
      :else (do (.unread s c) :body)))))

(defn skip-whitespace
  "Skips whitespace characters on stream s. Returns :line-start,
:stream-end, or :body to indicate the relative location of the next
character on s. Interprets comma as whitespace and semicolon as
comment to end of line. Does not interpret #! as comment to end
of line because only one character of lookahead is available.
The stream must either be an instance of LineNumberingPushbackReader
or duplicate its behavior of both supporting .unread and collapsing
all of CR, LF, and CRLF to a single \\newline."
  [s]
  (loop [c (.read s)]
    (cond
      (= c (int \newline)) :line-start
      (= c -1) :stream-end
      (= c (int \;)) (do (.readLine s) :line-start)
      (or (Character/isWhitespace (char c))
          (= c (int \,)))
      (recur (.read s))
      :else (do (.unread s c) :body))))
```

```
(defn repl-read
  "Default :read hook for repl. Reads from *in* which must either be
  an instance of LineNumberingPushbackReader or duplicate its behavior
  of both supporting .unread and collapsing all of CR, LF, and CRLF
  into a single
  \\newline. repl-read:
  - skips whitespace, then
  - returns request-prompt on start of line, or
  - returns request-exit on end of stream, or
  - reads an object from the input stream, then
    - skips the next input character if it's end of line, then
    - returns the object."
  [request-prompt request-exit]
  (or ({:line-start request-prompt :stream-end request-exit}
        (skip-whitespace *in*))
      (let [input (read)]
        (skip-if-eol *in*)
        input)))

(defn repl-exception
  "Returns the root cause of throwables"
  [throwable]
  (root-cause throwable))

(defn repl-caught
  "Default :caught hook for repl"
  [e]
  (let [ex (repl-exception e)
        tr (.getStackTrace ex)
        el (when-not (zero? (count tr)) (aget tr 0))]
    (binding [*out* *err*]
      (println
        (str (-> ex class .getSimpleName)
             " " (.getMessage ex) " "
             (when-not
               (instance? clojure.lang.Compiler$CompilerException ex)
               (str " "
                     (if el (stack-element-str el) "[trace missing]"))))))))

(defn repl
  "Generic, reusable, read-eval-print loop. By default, reads from
  *in*, writes to *out*, and prints exception summaries to *err*.
  If you use the default :read hook, *in* must either be an instance
  of LineNumberingPushbackReader or duplicate its behavior of both
  supporting .unread and collapsing CR, LF, and CRLF into a single
  \\newline. Options are sequential keyword-value pairs. Available
  options and their defaults:
  - :init, function of no arguments, initialization hook called
    with bindings for set!-able vars in place."
  [options]
```

```

default: #()

- :need-prompt, function of no arguments, called before each
  read-eval-print except the first, the user will be prompted
  if it returns true.
  default: (if (instance? LineNumberingPushbackReader *in*)
    #(atLineStart *in*)
    #(identity true))

- :prompt, function of no arguments, prompts for more input.
  default: repl-prompt

- :flush, function of no arguments, flushes output
  default: flush

- :read, function of two arguments, reads from *in*:
  - returns its first argument to request a fresh prompt
    - depending on need-prompt, this may cause the repl to
      prompt before reading again
  - returns its second argument to request an exit from the
    repl
  - else returns the next object read from the input stream
  default: repl-read

- :eval, function of one argument, returns the evaluation of its
  argument
  default: eval

- :print, function of one argument, prints its argument to the
  output
  default: prn

- :caught, function of one argument, a throwable, called when
  read, eval, or print throws an exception or error
  default: repl-caught"
[& options]
(let [cl (.getContextClassLoader (Thread/currentThread))]
  (.setContextClassLoader
    (Thread/currentThread) (clojure.lang.DynamicClassLoader. cl)))
(let [{:keys [init need-prompt prompt flush read eval print caught]
       :or {init      #()
             need-prompt
             (if (instance? LineNumberingPushbackReader *in*)
                 #(atLineStart ^LineNumberingPushbackReader *in*)
                 #(identity true))
             prompt      repl-prompt
             flush       flush
             read        repl-read
             eval        eval
             print       prn
             caught      repl-caught}}]
```

```

  (apply hash-map options)
  request-prompt (Object.)
  request-exit (Object.)
  read-eval-print
  (fn []
    (try
      (let [input (read request-prompt request-exit)]
        (or (#{request-prompt request-exit} input)
            (let [value (eval input)]
              (print value)
              (set! *3 *2)
              (set! *2 *1)
              (set! *1 value))))
      (catch Throwable e
        (caught e)
        (set! *e e))))]
  (with-bindings
    (try
      (init)
      (catch Throwable e
        (caught e)
        (set! *e e)))
    (use '[clojure.repl :only (source apropos dir pst doc find-doc)])
    (use '[clojure.java.javadoc :only (javadoc)])
    (use '[clojure.pprint :only (pp pprint)])
    (prompt)
    (flush)
    (loop []
      (when-not
        (try (= (read-eval-print) request-exit)
      (catch Throwable e
        (caught e)
        (set! *e e)
        nil))
      (when (need-prompt)
        (prompt)
        (flush)))
      (recur)))))

(defn load-script
  "Loads Clojure source from a file or resource given its path. Paths
  beginning with @ or @/ are considered relative to classpath."
  [^String path]
  (if (.startsWith path "@")
    (RT/loadResourceScript
      (.substring path (if (.startsWith path "@/") 2 1)))
    (Compiler/loadFile path)))

(defn- init-opt
  "Load a script"

```

```

[path]
(load-script path))

(defn- eval-opt
  "Evals expressions in str, prints each non-nil result using prn"
  [str]
  (let [eof (Object.)
        reader
        (LineNumberingPushbackReader. (java.io.StringReader. str))]
    (loop [input (read reader false eof)]
      (when-not (= input eof)
        (let [value (eval input)]
          (when-not (nil? value)
            (prn value))
          (recur (read reader false eof)))))))

(defn- init-dispatch
  "Returns the handler associated with an init opt"
  [opt]
  ({"-i"     init-opt
   "--init" init-opt
   "-e"     eval-opt
   "--eval" eval-opt} opt))

(defn- initialize
  "Common initialize routine for repl, script, and null opts"
  [args inits]
  (in-ns 'user)
  (set! *command-line-args* args)
  (doseq [[opt arg] inits]
    ((init-dispatch opt) arg)))

(defn- main-opt
  "Call the -main function from a namespace with string arguments from
  the command line."
  [[_ main-ns & args] inits]
  (with-bindings
    (initialize args inits)
    (apply (ns-resolve (doto (symbol main-ns) require) '-main) args)))

(defn- repl-opt
  "Start a repl with args and inits. Print greeting if no eval options
  were present"
  [[_ & args] inits]
  (when-not (some #(= eval-opt (init-dispatch (first %))) inits)
    (println "Clojure" (clojure-version)))
  (repl :init #(initialize args inits))
  (prn)
  (System/exit 0))

```

```
(defn- script-opt
  "Run a script from a file, resource, or standard in with args and
  inits"
  [[path & args] inits]
  (with-bindings
    (initialize args inits)
    (if (= path "-")
        (load-reader *in*)
        (load-script path)))))

(defn- null-opt
  "No repl or script opt present, just bind args and run inits"
  [args inits]
  (with-bindings
    (initialize args inits)))

(defn- help-opt
  "Print help text for main"
  [_ _]
  (println (:doc (meta (var main)))))

(defn- main-dispatch
  "Returns the handler associated with a main option"
  [opt]
  (or
    ({"-r"      repl-opt
      "--repl"   repl-opt
      "-m"      main-opt
      "--main"   main-opt
      nil       null-opt
      "-h"      help-opt
      "--help"   help-opt
      "-?"      help-opt} opt)
    script-opt))

(defn- legacy-repl
  "Called by the clojure.lang.Repl.main stub to run a repl with args
  specified the old way"
  [args]
  (println "WARNING: clojure.lang.Repl is deprecated.
Instead, use clojure.main like this:
java -cp clojure.jar clojure.main -i init.clj -r args...")
  (let [[inits [sep & args]] (split-with (complement #{"--"}) args)]
    (repl-opt (concat ["-r"] args) (map vector (repeat "-i") inits)))))

(defn- legacy-script
  "Called by the clojure.lang.Script.main stub to run a script with
  args specified the old way"
  [args]
  (println "WARNING: clojure.lang.Script is deprecated.
```

```

Instead, use clojure.main like this:
java -cp clojure.jar clojure.main -i init.clj script.clj args...""
(let [[inits [sep & args]] (split-with (complement #="--") args)]
  (null-opt args (map vector (repeat "-i") inits)))

(defn main
  "Usage:
   java -cp clojure.jar clojure.main [init-opt*] [main-opt] [arg*]

With no options or args, runs an interactive Read-Eval-Print Loop

init options:
  -i, --init path      Load a file or resource
  -e, --eval string    Evaluate expressions in string; print non-nil
                       values

main options:
  -m, --main ns-name   Call the -main function from a namespace
                       with args
  -r, --repl           Run a repl
  path                 Run a script from from a file or resource
  -                   Run a script from standard input
  -h, -?, --help       Print this help message and exit

operation:

  - Establishes thread-local bindings for commonly set!-able vars
  - Enters the user namespace
  - Binds *command-line-args* to a seq of strings containing command
    line args that appear after any main option
  - Runs all init options in order
  - Calls a -main function or runs a repl or script if requested

The init options may be repeated and mixed freely, but must appear
before any main option. The appearance of any eval option before
running a repl suppresses the usual repl greeting message:
\"Clojure ~(clojure-version)\".

Paths may be absolute or relative in the filesystem or relative to
classpath. Classpath-relative paths have prefix of @ or @/"
[& args]
(try
(if args
  (loop [[opt arg & more :as args] args inits []]
    (if (init-dispatch opt)
        (recur more (conj inits [opt arg]))
        ((main-dispatch opt) args inits)))
  (repl-opt nil nil))
(finally
  (flush))))
```

11.16 parallel.clj

— parallel.clj —

```
\getchunk{Clojure Copyright}

(ns ^{:doc "DEPRECATED Wrapper of the ForkJoin library (JSR-166)."
      :author "Rich Hickey"}
  clojure.parallel)
(alias 'parallel 'clojure.parallel)

(comment "
The parallel library wraps the ForkJoin library scheduled for
inclusion in JDK 7:

http://gee.cs.oswego.edu/dl/concurrency-interest/index.html

You'll need jsr166y.jar in your classpath in order to use this
library. The basic idea is that Clojure collections, and most
efficiently vectors, can be turned into parallel arrays for use by
this library with the function par, although most of the functions
take collections and will call par if needed, so normally you will
only need to call par explicitly in order to attach bound/filter/map
ops. Parallel arrays support the attachment of bounds, filters and
mapping functions prior to realization/calculation, which happens as
the result of any of several operations on the
array (pvec/psort/pfilter-nils/pfilter-dupes). Rather than perform
composite operations in steps, as would normally be done with
sequences, maps and filters are instead attached and thus composed by
providing ops to par. Note that there is an order sensitivity to the
attachments - bounds precede filters precede mappings. All operations
then happen in parallel, using multiple threads and a sophisticated
work-stealing system supported by fork-join, either when the array is
realized, or to perform aggregate operations like preduce/pmin/pmax
etc. A parallel array can be realized into a Clojure vector using
pvec.
")

(import '(jsr166y.forkjoin ParallelArray ParallelArrayWithBounds
  ParallelArrayWithFilter ParallelArrayWithMapping Ops$Op
  Ops$BinaryOp Ops$Reducer Ops$Predicate Ops$BinaryPredicate
  Ops$IntAndObjectPredicate Ops$IntAndObjectToObject))
```

```
(defn- op [f]
  (proxy [Ops$Op] []
    (op [x] (f x)))

(defn- binary-op [f]
  (proxy [Ops$BinaryOp] []
    (op [x y] (f x y)))

(defn- int-and-object-to-object [f]
  (proxy [Ops$IntAndObjectToObject] []
    (op [i x] (f x i)))

(defn- reducer [f]
  (proxy [Ops$Reducer] []
    (op [x y] (f x y)))

(defn- predicate [f]
  (proxy [Ops$Predicate] []
    (op [x] (boolean (f x)))))

(defn- binary-predicate [f]
  (proxy [Ops$BinaryPredicate] []
    (op [x y] (boolean (f x y)))))

(defn- int-and-object-predicate [f]
  (proxy [Ops$IntAndObjectPredicate] []
    (op [i x] (boolean (f x i)))))

(defn par
  "Creates a parallel array from coll. ops, if supplied, perform
  on-the-fly filtering or transformations during parallel realization
  or calculation. ops form a chain, and bounds must precede filters,
  must precede maps. ops must be a set of keyword value pairs of the
  following forms:

  :bound [start end]

  Only elements from start (inclusive) to end (exclusive) will be
  processed when the array is realized.

  :filter pred

  Filter preds remove elements from processing when the array is
  realized. pred must be a function of one argument whose return
  will be processed via boolean.

  :filter-index pred2

  pred2 must be a function of two arguments, which will be an element
  of the collection and the corresponding index, whose return will be"
```

:filter pred

Filter preds remove elements from processing when the array is realized. pred must be a function of one argument whose return will be processed via boolean.

:filter-index pred2

pred2 must be a function of two arguments, which will be an element of the collection and the corresponding index, whose return will be

processed via boolean.

```
:filter-with [pred2 coll2]
```

pred2 must be a function of two arguments, which will be corresponding elements of the 2 collections.

```
:map f
```

Map fns will be used to transform elements when the array is realized. f must be a function of one argument.

```
:map-index f2
```

f2 must be a function of two arguments, which will be an element of the collection and the corresponding index.

```
:map-with [f2 coll2]
```

f2 must be a function of two arguments, which will be corresponding elements of the 2 collections."

```
([coll]
  (if (instance? ParallelArrayWithMapping coll)
    coll
    (. ParallelArray createUsingHandoff
      (to-array coll)
      (. ParallelArray defaultExecutor))))
([coll & ops]
  (reduce
    (fn [pa [op args]]
      (cond
        (= op :bound)
        (. pa withBounds (args 0) (args 1))
        (= op :filter)
        (. pa withFilter (predicate args))
        (= op :filter-with)
        (. pa withFilter (binary-predicate (args 0)) (par (args 1)))
        (= op :filter-index)
        (. pa withIndexedFilter (int-and-object-predicate args))
        (= op :map)
        (. pa withMapping (parallel/op args))
        (= op :map-with)
        (. pa withMapping (binary-op (args 0)) (par (args 1)))
        (= op :map-index)
        (. pa withIndexedMapping (int-and-object-to-object args))
        :else (throw (Exception. (str "Unsupported par op: " op)))))))
      (par coll)
      (partition 2 ops))))
```

```

;;;;;;;;;; aggregate operations ;;;;;;;
(defn pany
  "Returns some (random) element of the coll if it satisfies the
  bound/filter/map"
  [coll]
  (. (par coll) any))

(defn pmax
  "Returns the maximum element, presuming Comparable elements, unless
  a Comparator comp is supplied"
  ([coll] (. (par coll) max))
  ([coll comp] (. (par coll) max comp)))

(defn pmin
  "Returns the minimum element, presuming Comparable elements, unless
  a Comparator comp is supplied"
  ([coll] (. (par coll) min))
  ([coll comp] (. (par coll) min comp)))

(defn- summary-map [s]
  {:min (.min s) :max (.max s) :size (.size s)
   :min-index (.indexOfMin s) :max-index (.indexOfMax s)})

(defn psummary
  "Returns a map of summary statistics (min, max, size, min-index,
  max-index, presuming Comparable elements, unless a Comparator
  comp is supplied"
  ([coll] (summary-map (. (par coll) summary)))
  ([coll comp] (summary-map (. (par coll) summary comp)))))

(defn preduce
  "Returns the reduction of the realized elements of coll
  using function f. Note f will not necessarily be called
  consecutively, and so must be commutative. Also note that
  (f base an-element) might be performed many times, i.e. base is not
  an initial value as with sequential reduce."
  [f base coll]
  (. (par coll) (reduce (reducer f) base)))

;;;;;;;;; collection-producing operations ;;;;;;;
(defn- pa-to-vec [pa]
  (vec (. pa getArray)))

(defn- pall
  "Realizes a copy of the coll as a parallel array, with any
  bounds/filters/maps applied"
  [coll]
  (if (instance? ParallelArrayWithMapping coll)
    (. coll all))

```

```

(par coll)))

(defn pvec
  "Returns the realized contents of the parallel array pa as
  a Clojure vector"
  [pa] (pa-to-vec (pall pa)))

(defn pdistinct
  "Returns a parallel array of the distinct elements of coll"
  [coll]
  (pa-to-vec (. (pall coll) allUniqueElements)))

;this doesn't work, passes null to reducer?
(defn- pcumulate [coll f init]
 (.. (pall coll) (precumulate (reducer f) init)))

(defn psort
  "Returns a new vector consisting of the realized items in coll,
  sorted, presuming Comparable elements, unless a Comparator comp
  is supplied"
  ([coll] (pa-to-vec (. (pall coll) sort)))
  ([coll comp] (pa-to-vec (. (pall coll) sort comp)))))

(defn pfilter-nils
  "Returns a vector containing the non-nil (realized) elements of coll"
  [coll]
  (pa-to-vec (. (pall coll) removeNulls)))

(defn pfilter-dupes
  "Returns a vector containing the (realized) elements of coll,
  without any consecutive duplicates"
  [coll]
  (pa-to-vec (. (pall coll) removeConsecutiveDuplicates)))

(comment
(load-file "src/parallel.clj")
(refer 'parallel)
(pdistinct [1 2 3 2 1])
;(pcumulate [1 2 3 2 1] + 0) ;broken, not exposed
(def a (make-array Object 1000000))
(dotimes i (count a)
  (aset a i (rand-int i)))
(time (reduce + 0 a))
(time (preduce + 0 a))
(time (count (distinct a)))
(time (count (pdistinct a)))

(preduce + 0 [1 2 3 2 1])
(preduce + 0 (psort a))

```

```
(pvec (par [11 2 3 2] :filter-index (fn [x i] (> i x))))
(pvec (par [11 2 3 2] :filter-with [(fn [x y] (> y x)) [110 2 33 2]]))

(pssummary ;or pvec/pmax etc
(par [11 2 3 2]
      :filter-with [(fn [x y] (> y x))
                    [110 2 33 2]]
      :map #(* % 2)))

(preduce + 0
(par [11 2 3 2]
      :filter-with [< [110 2 33 2]]))

(time (reduce + 0 (map #(* % %) (range 1000000))))
(time (preduce + 0 (par (range 1000000) :map-index *)))
(def v (range 1000000))
(time (preduce + 0 (par v :map-index *)))
(time (preduce + 0 (par v :map #(* % %))))
(time (reduce + 0 (map #(* % %) v)))
)
```

11.17 cl=format.clj

— cl=format.clj —

```
\getchunk{Clojure Copyright}

;; Author: Tom Faulhaber
;; April 3, 2009

;; This module implements the Common Lisp compatible format
;; function as documented in "Common Lisp the Language, 2nd edition",
;; Chapter 22 (available online at:
;; http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/
;; ai/html/cltl/clm/node200.html#SECTION0026330000000000000000)

(in-ns 'clojure.pprint)

;; Forward references
(declare compile-format)
(declare execute-format)
(declare init-navigator)
;; End forward references
```

```
(defn cl-format
  "An implementation of a Common Lisp compatible format function.
  cl-format formats its arguments to an output stream or string based
  on the format control string given. It supports sophisticated
  formatting of structured data."
```

Writer is an instance of `java.io.Writer`, true to output to `*out*` or nil to output to a string, format-in is the format control string and the remaining arguments are the data to be formatted.

The format control string is a string to be output with embedded 'format directives' describing how to format the various arguments passed in.

If writer is nil, cl-format returns the formatted result string. Otherwise, cl-format returns nil.

For example:

```
(let [results [46 38 22]]
  (cl-format true
    \"There ~[are~;is~:;are~]~*:~d result~:p: ~{~d~~, ~}~%\"
    (count results) results))
```

Prints to `*out*`:

```
There are 3 results: 46, 38, 22
```

Detailed documentation on format control strings is available in the "Common Lisp the Language, 2nd edition", Chapter 22 (available online at:

<http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/cltl/clm/node200.html#SECTION00263300000000000000>)
and in the Common Lisp HyperSpec at
http://www.lispworks.com/documentation/HyperSpec/Body/22_c.htm

```
"{:added "1.2",
:see-also
[[[str "http://www.cs.cmu.edu/afs/cs.cmu.edu/project/"
  "ai-repository/ai/html/cltl/clm/node200.html"
  "#SECTION00263300000000000000" )
  "Common Lisp the Language"]
 ["http://www.lispworks.com/documentation/HyperSpec/Body/22_c.htm"
  "Common Lisp HyperSpec"]]]}
[writer format-in & args]
(let [compiled-format
      (if (string? format-in) (compile-format format-in) format-in)
      navigator (init-navigator args)]
  (execute-format writer compiled-format navigator)))
```

```
(def ^:dynamic ^{:private true} *format-str* nil)
```

```
(defn- format-error [message offset]
  (let [full-message
        (str message \newline *format-str* \newline
             (apply str (repeat offset \space)) "^\\" \newline)]
    (throw (RuntimeException. full-message)))

;;;;;;;;
;; Argument navigators manage the argument list
;; as the format statement moves through the list
;; (possibly going forwards and backwards as it does so)
;;;;;;;

(defstruct ^{:private true}
  arg-navigator :seq :rest :pos )

(defn- init-navigator
  "Create a new arg-navigator from the sequence with the position
  set to 0"
  {:skip-wiki true}
  [s]
  (let [s (seq s)]
    (struct arg-navigator s s 0)))

;; TODO call format-error with offset
(defn- next-arg [navigator]
  (let [rst (:rest navigator) ]
    (if rst
        [(first rst)
         (struct arg-navigator (:seq navigator) (next rst)
                           (inc (:pos navigator)))]
        (throw (new Exception
                      "Not enough arguments for format definition"))))

(defn- next-arg-or-nil [navigator]
  (let [rst (:rest navigator)]
    (if rst
        [(first rst)
         (struct arg-navigator (:seq navigator) (next rst)
                           (inc (:pos navigator)))]
        [nil navigator])))

;; Get an argument off the arg list and compile it if it's
;; not already compiled
(defn- get-format-arg [navigator]
  (let [[raw-format navigator] (next-arg navigator)
        compiled-format (if (instance? String raw-format)
                           (compile-format raw-format)
                           raw-format)]
    [compiled-format navigator]))
```

```
(declare relative-reposition)

(defn- absolute-reposition [navigator position]
  (if (>= position (:pos navigator))
    (relative-reposition navigator (- (:pos navigator) position))
    (struct arg-navigator (:seq navigator)
      (drop position (:seq navigator) position)))

(defn- relative-reposition [navigator position]
  (let [newpos (+ (:pos navigator) position)]
    (if (neg? position)
      (absolute-reposition navigator newpos)
      (struct arg-navigator (:seq navigator)
        (drop position (:rest navigator) newpos)))))

(defdestruct ^{:private true}
  compiled-directive :func :def :params :offset)

;;;;;;;;
;; When looking at the parameter list, we may need to manipulate
;; the argument list as well (for 'V' and '#' parameter types).
;; We hide all of this behind a function, but clients need to
;; manage changing arg navigator
;;;;;;;

;; TODO: validate parameters when they come from arg list
(defn- realize-parameter [[param [raw-val offset]] navigator]
  (let [[real-param new-navigator]
        (cond
          ;pass flags through unchanged - this really isn't necessary
          (contains? #{:at :colon} param)

          [raw-val navigator]

          (= raw-val :parameter-from-args)
          (next-arg navigator)

          (= raw-val :remaining-arg-count)
          [(count (:rest navigator)) navigator]

          true
          [raw-val navigator])
        [[param [real-param offset]] new-navigator]])

(defn- realize-parameter-list [parameter-map navigator]
  (let [[pairs new-navigator]
        (map-passing-context realize-parameter navigator parameter-map)]
    [(into {} pairs) new-navigator]))

;;;;;;;
```

```

;; Functions that support individual directives
;;;;; Common handling code for ^A and ^S
;;;;;

(declare opt-base-str)

(def ^{:private true}
  special-radix-markers {2 "#b" 8 "#o", 16 "#x"})

(defn- format-simple-number [n]
  (cond
    (integer? n) (if (= *print-base* 10)
                    (str n (if *print-radix* "."))
                    (str
                     (if *print-radix*
                         (or (get special-radix-markers *print-base*)
                             (str "#" *print-base* "r")))
                     (opt-base-str *print-base* n)))
    (ratio? n) (str
                 (if *print-radix*
                     (or (get special-radix-markers *print-base*)
                         (str "#" *print-base* "r")))
                 (opt-base-str *print-base* (.numerator n))
                 "/"
                 (opt-base-str *print-base* (.denominator n)))
    :else nil))

(defn- format-ascii [print-func params arg-navigator offsets]
  (let [ [arg arg-navigator] (next-arg arg-navigator)
        ^String base-output (or (format-simple-number arg)
                                 (print-func arg))
        base-width (.length base-output)
        min-width (+ base-width (:minpad params))
        width (if (>= min-width (:mincol params))
                  min-width
                  (+ min-width
                     (* (+ (quot (- (:mincol params) min-width 1)
                                  (:colinc params) )
                            1)
                        (:colinc params))))
        chars (apply str
                     (repeat (- width base-width) (:padchar params)))]
    (if (:at params)
        (print (str chars base-output))
        (print (str base-output chars)))
    arg-navigator)))

```

```
;;;;;;;;
;;;;; Support for the integer directives ~D, ~X, ~O, ~B and some
;;;;; of ~R
;;;;;;;;
(defn- integral?
  "returns true if a number is actually an integer (that is, has no
fractional part)"
  [x]
  (cond
    (integer? x) true
    (decimal? x) ; true iff no fractional part
    (>= (.ulp (.stripTrailingZeros (bigdec 0))) 1)
    (float? x) (= x (Math/floor x))
    (ratio? x) (let [^clojure.lang.Ratio r x]
                 (= 0 (rem (.numerator r) (.denominator r))))
    :else false))

(defn- remainders
  "Return the list of remainders (essentially the 'digits') of val
in the given base"
  [base val]
  (reverse
    (first
      (consume #(if (pos? %)
                  [(rem % base) (quot % base)]
                  [nil nil])
                  val)))))

;;; TODO: xlated-val does not seem to be used here.
(defn- base-str
  "Return val as a string in the given base"
  [base val]
  (if (zero? val)
    "0"
    (let [xlated-val (cond
                        (float? val) (bigdec val)
                        (ratio? val)
                        (let [^clojure.lang.Ratio r val]
                          (/ (.numerator r) (.denominator r)))
                        :else val)]
      (apply str
        (map
          #(if (< % 10)
            (char (+ (int \0) %))
            (char (+ (int \a) (- % 10))))
          (remainders base val))))))

(def ^{:private true}
  java-base-formats {8 "%o", 10 "%d", 16 "%x"})
```

```

(defn- opt-base-str
  "Return val as a string in the given base, using
  clojure.core/format if supported
  for improved performance"
  [base val]
  (let [format-str (get java-base-formats base)]
    (if (and format-str
              (integer? val)
              (not (instance? clojure.lang.BigInt val)))
        (clojure.core/format format-str val)
        (base-str base val)))))

(defn- group-by* [unit lis]
  (reverse
   (first
    (consume (fn [x]
               [(seq (reverse (take unit x)))
                (seq (drop unit x))]) (reverse lis)))))

(defn- format-integer [base params arg-navigator offsets]
  (let [[arg arg-navigator] (next-arg arg-navigator)]
    (if (integral? arg)
        (let [neg (neg? arg)
              pos-arg (if neg (- arg) arg)
              raw-str (opt-base-str base pos-arg)
              group-str
              (if (:colon params)
                  (let [groups
                        (map #(apply str %)
                              (group-by* (:commainterval params) raw-str))]
                    commas
                    (repeat (count groups) (:commachar params)))
                  (apply str (next (interleave commas groups))))))]
          raw-str)
        ^String signed-str (cond
          neg (str "-" group-str)
          (:at params) (str "+" group-str)
          true group-str)
        padded-str
        (if (< (.length signed-str) (:mincol params))
            (str (apply str
                         (repeat (- (:mincol params) (.length signed-str))
                                (:padchar params))))
                  signed-str)
            signed-str)])
      (print padded-str))
    (format-ascii print-str {:mincol (:mincol params)
                            :colinc 1 :minpad 0
                            :padchar (:padchar params) :at true})
  
```

```

        (init-navigator [arg]) nil))
arg-navigator))

;;;;;;;;
;; Support for english formats (~R and ~:R)
;;;;;;;;
(def ^{:private true}
  english-cardinal-units
  ["zero" "one" "two" "three" "four" "five" "six" "seven" "eight"
   "nine" "ten" "eleven" "twelve" "thirteen" "fourteen"
   "fifteen" "sixteen" "seventeen" "eighteen" "nineteen"])

(def ^{:private true}
  english-ordinal-units
  ["zeroth" "first" "second" "third" "fourth" "fifth" "sixth"
   "seventh" "eighth" "ninth"
   "tenth" "eleventh" "twelfth" "thirteenth" "fourteenth"
   "fifteenth" "sixteenth" "seventeenth" "eighteenth" "nineteenth"])

(def ^{:private true}
  english-cardinal-tens
  ["" "" "twenty" "thirty" "forty" "fifty" "sixty" "seventy"
   "eighty" "ninety"])

(def ^{:private true}
  english-ordinal-tens
  ["" "" "twentieth" "thirtieth" "fortieth" "fiftieth"
   "sixtieth" "seventieth" "eightieth" "ninetieth"])

;; We use "short scale" for our units
;; (see http://en.wikipedia.org/wiki/Long_and_short_scales)
;; Number names from http://www.jimloy.com/math/billion.htm
;; We follow the rules for writing numbers from the Blue Book
;; (http://www.grammarbook.com/numbers/numbers.asp)
(def ^{:private true}
  english-scale-numbers
  ["" "thousand" "million" "billion" "trillion" "quadrillion"
   "quintillion" "sextillion" "septillion" "octillion" "nonillion"
   "decillion" "undecillion" "duodecillion" "tredecillion"
   "quattuordecillion" "quindecillion" "sexdecillion"
   "septendecillion" "octodecillion" "novemdecillion"
   "vigintillion"])
)

(defn- format-simple-cardinal
  "Convert a number less than 1000 to a cardinal english string"
  [num]
  (let [hundreds (quot num 100)
        tens (rem num 100)]
    (str
      (if (= tens 0)
        (if (= hundreds 0)
          num
          (str hundreds " hundred"))
        (str hundreds " hundred and " tens)))
      (if (= tens 0)
        ""
        (str "-" (english-cardinal-units tens)))))))

```

```

(if (pos? hundreds)
    (str (nth english-cardinal-units hundreds) " hundred"))
(if (and (pos? hundreds) (pos? tens)) " ")
(if (pos? tens)
    (if (< tens 20)
        (nth english-cardinal-units tens)
        (let [ten-digit (quot tens 10)
              unit-digit (rem tens 10)]
            (str
                (if (pos? ten-digit) (nth english-cardinal-tens ten-digit))
                (if (and (pos? ten-digit) (pos? unit-digit)) "-")
                (if (pos? unit-digit)
                    (nth english-cardinal-units unit-digit)))))))
(defn- add-english-scales
  "Take a sequence of parts, add scale numbers (e.g., million) and
  combine into a string offset is a factor of 10^3 to multiply by"
  [parts offset]
  (let [cnt (count parts)]
    (loop [acc []
           pos (dec cnt)
           this (first parts)
           remainder (next parts)]
      (if (nil? remainder)
          (str (apply str (interpose ", " acc))
               (if (and (not (empty? this)) (not (empty? acc))) ", ")
               this
               (if (and (not (empty? this)) (pos? (+ pos offset)))
                   (str " " (nth english-scale-numbers (+ pos offset))))))
          (recur
            (if (empty? this)
                acc
                (conj acc
                      (str this " " (nth english-scale-numbers (+ pos offset)))))))
      (dec pos)
      (first remainder)
      (next remainder)))))

(defn- format-cardinal-english [params navigator offsets]
  (let [[arg navigator] (next-arg navigator)]
    (if (= 0 arg)
        (print "zero")
        (let [abs-arg
              (if (neg? arg)
                  (- arg)
                  arg) ; some numbers are too big for Math/abs
              parts (remainders 1000 abs-arg)]
          (if (<= (count parts) (count english-scale-numbers))
              (let [parts-strs (map format-simple-cardinal parts)
                    full-str (add-english-scales parts-strs 0)]

```

```

    (print (str (if (neg? arg) "minus ") full-str)))
  (format-integer ;; for numbers > 10^63, we fall back on ~D
    10
    { :mincol 0, :padchar \space, :commachar \,
      :commainterval 3, :colon true}
    (init-navigator [arg])
    { :mincol 0, :padchar 0, :commachar 0 :commainterval 0}))))
  navigator))

(defn- format-simple-ordinal
  "Convert a number less than 1000 to a ordinal english string
  Note this should only be used for the last one in the sequence"
  [num]
  (let [hundreds (quot num 100)
        tens (rem num 100)]
    (str
      (if (pos? hundreds)
          (str (nth english-cardinal-units hundreds) " hundred"))
      (if (and (pos? hundreds) (pos? tens)) " ")
      (if (pos? tens)
          (if (< tens 20)
              (nth english-ordinal-units tens)
              (let [ten-digit (quot tens 10)
                    unit-digit (rem tens 10)]
                (if (and (pos? ten-digit) (not (pos? unit-digit)))
                    (nth english-ordinal-tens ten-digit)
                    (str
                      (if (pos? ten-digit) (nth english-cardinal-tens ten-digit))
                      (if (and (pos? ten-digit) (pos? unit-digit)) "-")
                      (if (pos? unit-digit)
                          (nth english-ordinal-units unit-digit)))))))
          (if (pos? hundreds) "th")))))
  (if (pos? hundreds) "th")))))

(defn- format-ordinal-english [params navigator offsets]
  (let [[arg navigator] (next-arg navigator)]
    (if (= 0 arg)
        (print "zeroth")
        (let [abs-arg
              (if (neg? arg)
                  (- arg)
                  arg) ; some numbers are too big for Math/abs
              parts (remainders 1000 abs-arg)]
          (if (<= (count parts) (count english-scale-numbers))
              (let [parts-strs (map format-simple-cardinal (drop-last parts))
                    head-str (add-english-scales parts-strs 1)
                    tail-str (format-simple-ordinal (last parts))]
                (print (str (if (neg? arg) "minus ")
                           (cond
                             (and (not (empty? head-str))
                                   (not (empty? tail-str)))))))
```

```

(str head-str ", " tail-str)

(not (empty? head-str)) (str head-str "th")
:else tail-str)))))
(do (format-integer ;; for numbers > 10^63, we fall back on ~D
10
{:mincol 0, :padchar \space, :commachar \,
:commaininterval 3, :colon true}
(init-navigator [arg])
{:mincol 0, :padchar 0, :commachar 0 :commaininterval 0})
(let [low-two-digits (rem arg 100)
not-teens (or (< 11 low-two-digits)
(> 19 low-two-digits))
low-digit (rem low-two-digits 10)]
(print (cond
(and (== low-digit 1) not-teens) "st"
(and (== low-digit 2) not-teens) "nd"
(and (== low-digit 3) not-teens) "rd"
:else "th")))))
(navigator))

;;;;;;;;
;; Support for roman numeral formats (~@R and ~@:R)
;;;;;;;;
(def ^{:private true}
old-roman-table
[[ "I" "II" "III" "IV" "V" "VI" "VII" "VIII" "VIII"
["X" "XX" "XXX" "XL" "L" "LX" "LXX" "LXXX" "LXXXX"]
["C" "CC" "CCC" "CCCC" "D" "DC" "DCC" "DCCC" "DCCCC"]
["M" "MM" "MMM"]])

(def ^{:private true}
new-roman-table
[[ "I" "II" "III" "IV" "V" "VI" "VII" "VIII" "IX"
["X" "XX" "XXX" "XL" "L" "LX" "LXX" "LXXX" "XC"]
["C" "CC" "CCC" "CD" "D" "DC" "DCC" "DCCC" "CM"]
["M" "MM" "MMM"]])

(defn- format-roman
"Format a roman numeral using the specified look-up table"
[table params navigator offsets]
(let [[arg navigator] (next-arg navigator)]
(if (and (number? arg) (> arg 0) (< arg 4000))
(let [digits (remainders 10 arg)]
(loop [acc []
pos (dec (count digits))
digits digits]
(if (empty? digits)

```

```

(print (apply str acc))
(let [digit (first digits)]
  (recur (if (= 0 digit)
            acc
            (conj acc (nth (nth table pos) (dec digit))))
            (dec pos)
            (next digits))))))
(format-integer ;; for anything <= 0 or > 3999, we fall back on ^D
 10
  {:mincol 0, :padchar \space, :commachar \,
   :commainterval 3, :colon true}
  (init-navigator [arg])
  {:mincol 0, :padchar 0, :commachar 0 :commainterval 0})
  navigator))

(defn- format-old-roman [params navigator offsets]
  (format-roman old-roman-table params navigator offsets))

(defn- format-new-roman [params navigator offsets]
  (format-roman new-roman-table params navigator offsets))

;;;;;;;;
;; Support for character formats (^C)
;;;;;;;;
(def ^{:private true}
  special-chars { 8 "Backspace", 9 "Tab", 10 "Newline",
                  13 "Return", 32 "Space"})

(defn- pretty-character [params navigator offsets]
  (let [[c navigator] (next-arg navigator)
        as-int (int c)
        base-char (bit-and as-int 127)
        meta (bit-and as-int 128)
        special (get special-chars base-char)]
    (if (> meta 0) (print "Meta-"))
    (print (cond
              special special
              (< base-char 32) (str "Control-" (char (+ base-char 64)))
              (= base-char 127) "Control-?"
              :else (char base-char)))
    navigator))

(defn- readable-character [params navigator offsets]
  (let [[c navigator] (next-arg navigator)]
    (condp = (:char-format params)
      \o (cl-format true "\o~3,'0o" (int c))
      \u (cl-format true "\u~4,'0x" (int c))
      nil (pr c))
    navigator))

```

```
(defn- plain-character [params navigator offsets]
  (let [[char navigator] (next-arg navigator)]
    (print char)
    navigator))

;; Check to see if a result is an abort (^^) construct
;; TODO: move these funcs somewhere more appropriate
(defn- abort? [context]
  (let [token (first context)]
    (or (= :up-arrow token) (= :colon-up-arrow token)))))

;; Handle the execution of "sub-clauses" in bracket constructions
(defn- execute-sub-format [format args base-args]
  (second
   (map-passing-context
    (fn [element context]
      (if (abort? context)
          [nil context] ; just keep passing it along
          (let [[params args]
                (realize-parameter-list (:params element) context)
                [params offsets] (unzip-map params)
                params (assoc params :base-args base-args)]
              [nil (apply (:func element) [params args offsets])])))
      args
      format)))

;;;;;;;;
;; Support for real number formats
;;;;;;;

;; TODO - return exponent as int to eliminate double conversion
(defn- float-parts-base
  "Produce string parts for the mantissa (normalized 1-9) and exponent"
  [^Object f]
  (let [^String s (.toLowerCase (.toString f))
        exploc (.indexOf s (int \e))]
    (if (neg? exploc)
        (let [dotloc (.indexOf s (int \.))]
          (if (neg? dotloc)
              [s (str (dec (count s)))]
              [(str (subs s 0 dotloc) (subs s (inc dotloc)))
               (str (dec dotloc))]))
        [(str (subs s 0 1) (subs s 2 exploc)) (subs s (inc exploc))])))

(defn- float-parts
  "Take care of leading and trailing zeros in decomposed floats"
  [f]
  (let [[m ^String e] (float-parts-base f)
```

```

m1 (rtrim m \0)
m2 (ltrim m1 \0)
delta (- (count m1) (count m2))
^String e (if (and (pos? (count e)) (= (nth e 0) \+))
                  (subs e 1) e)]
(if (empty? m2)
    ["0" 0]
    [m2 (- (Integer/valueOf e) delta)]))

(defn- round-str [m e d w]
  (if (or d w)
      (let [len (count m)
            round-pos (if d (+ e d 1))
            round-pos (if (and w (< (inc e) (dec w))
                                (or (nil? round-pos)
                                    (< (dec w) round-pos)))
                         (dec w)
                         round-pos)
            [m1 e1 round-pos len] (if (= round-pos 0)
                                         [(str "0" m) (inc e) 1 (inc len)]
                                         [m e round-pos len])]
            (if round-pos
                (if (neg? round-pos)
                    ["0" 0 false]
                    (if (> len round-pos)
                        (let [round-char (nth m1 round-pos)
                              ^String result (subs m1 0 round-pos)]
                          (if (>= (int round-char) (int \5))
                              (let [result-val (Integer/valueOf result)
                                    leading-zeros
                                    (subs result 0
                                          (min (prefix-count result \0)
                                                (- round-pos 1)))])
                            round-up-result
                            (str leading-zeros
                                 (String/valueOf (+ result-val
                                                    (if (neg? result-val) -1 1))))
                            expanded (> (count round-up-result)
                                       (count result)))
                            [round-up-result e1 expanded])
                          [result e1 false]))
                    [m e false])))
                [m e false]))
            [m e false])))

(defn- expand-fixed [m e d]
  (let [m1 (if (neg? e) (str (apply str (repeat (dec (- e)) \0)) m) m)
        len (count m1)
        target-len (if d (+ e d 1) (inc e))]
  (if (< len target-len)

```

```

(str m1 (apply str (repeat (- target-len len) \0)))
m1)))

(defn- insert-decimal
  "Insert the decimal point at the right spot in the number to
  match an exponent"
  [m e]
  (if (neg? e)
    (str "." m)
    (let [loc (inc e)]
      (str (subs m 0 loc) "." (subs m loc)))))

(defn- get-fixed [m e d]
  (insert-decimal (expand-fixed m e d) e))

(defn- insert-scaled-decimal
  "Insert the decimal point at the right spot in the number to
  match an exponent"
  [m k]
  (if (neg? k)
    (str "." m)
    (str (subs m 0 k) "." (subs m k)))))

;; the function to render ~F directives
;; TODO: support rationals. Back off to ~D/~A is the appropriate cases
(defn- fixed-float [params navigator offsets]
  (let [w (:w params)
        d (:d params)
        [arg navigator] (next-arg navigator)
        [sign abs] (if (neg? arg) ["-" (- arg)] [ "+" arg])
        [mantissa exp] (float-parts abs)
        scaled-exp (+ exp (:k params))
        add-sign (or (:at params) (neg? arg))
        append-zero (and (not d) (<= (dec (count mantissa)) scaled-exp))
        [rounded-mantissa scaled-exp expanded]
        (round-str mantissa scaled-exp
                    d (if w (- w (if add-sign 1 0)))))
        fixed-repr
        (get-fixed rounded-mantissa
                  (if expanded (inc scaled-exp) scaled-exp) d)
        prepend-zero (= (first fixed-repr) \.)]
    (if w
        (let [len (count fixed-repr)
              signed-len (if add-sign (inc len) len)
              prepend-zero (and prepend-zero (not (>= signed-len w)))
              append-zero (and append-zero (not (>= signed-len w)))
              full-len (if (or prepend-zero append-zero)
                         (inc signed-len)
                         signed-len)]
          (if (and (> full-len w) (:overflowchar params))

```



```

scaled-mantissa 0
(cond
  (= k 0) (dec d)
  (pos? k) d
  (neg? k) (dec d))
  (if w-mantissa (- w-mantissa (if add-sign 1 0))))
full-mantissa (insert-scaled-decimal rounded-mantissa k)
append-zero (and (= k (count rounded-mantissa)) (nil? d))]
(if (not incr-exp)
  (if w
    (let [len (+ (count full-mantissa) exp-width)
          signed-len (if add-sign (inc len) len)
          prepend-zero (and prepend-zero (not (= signed-len w)))
          full-len (if prepend-zero (inc signed-len) signed-len)
          append-zero (and append-zero (< full-len w))]
      (if (and (or (> full-len w) (and e (> (- exp-width 2) e)))
                (:overflowchar params))
              (print (apply str (repeat w (:overflowchar params)))))
        (print (str
                  (apply str
                      (repeat
                        (- w full-len (if append-zero 1 0) )
                        (:padchar params)))
                  (if add-sign (if (neg? arg) \- \+)
                      (if prepend-zero "0")
                      full-mantissa
                      (if append-zero "0"
                          scaled-exp-str)))))
        (print (str
                  (if add-sign (if (neg? arg) \- \+)
                      (if prepend-zero "0")
                      full-mantissa
                      (if append-zero "0"
                          scaled-exp-str))))
        (recur [rounded-mantissa (inc exp)]))))
      navigator))

;; the function to render ~G directives
;; This just figures out whether to pass the request off to
;; ~F or ~E based on the algorithm in CLtL.
;; TODO: support rationals. Back off to ~D/~A is the appropriate cases
;; TODO: refactor so that float-parts isn't called twice
(defn- general-float [params navigator offsets]
  (let [[arg _] (next-arg navigator)
        [mantissa exp] (float-parts (if (neg? arg) (- arg) arg))
        w (:w params)
        d (:d params)
        e (:e params)
        n (if (= arg 0.0) 0 (inc exp))
        ee (if e (+ e 2) 4))
    
```

```

ww (if w (- w ee))
d (if d d (max (count mantissa) (min n 7)))
dd (- d n)]
(if (<= 0 dd d)
  (let [navigator (fixed-float {:w ww, :d dd, :k 0,
                                :overflowchar (:overflowchar params),
                                :padchar (:padchar params),
                                :at (:at params)}
                                navigator offsets)]
    (print (apply str (repeat ee \space)))
    navigator)
  (exponential-float params navigator offsets)))

;; the function to render ~$ directives
;; TODO: support rationals. Back off to ~D/~A is the appropriate cases
(defn- dollar-float [params navigator offsets]
  (let [[^Double arg navigator] (next-arg navigator)
        [mantissa exp] (float-parts (Math/abs arg))
        d (:d params) ; digits after the decimal
        n (:n params) ; minimum digits before the decimal
        w (:w params) ; minimum field width
        add-sign (or (:at params) (neg? arg))
        [rounded-mantissa scaled-exp expanded]
        (round-str mantissa exp d nil)
        ^String fixed-repr
        (get-fixed rounded-mantissa
          (if expanded (inc scaled-exp) scaled-exp) d)
        full-repr
        (str
          (apply str
            (repeat (- n (.indexOf fixed-repr (int \.))) \0)) fixed-repr)
        full-len (+ (count full-repr) (if add-sign 1 0)))
      (print (str
        (if (and (:colon params) add-sign) (if (neg? arg) \- \+)
          (apply str (repeat (- w full-len) (:padchar params))))
        (if (and (not (:colon params)) add-sign)
          (if (neg? arg) \- \+)
          full-repr)))
      navigator))

;;;;;;
;; Support for the '~~[...~]' conditional construct in its
;; different flavors
;;;;;;

;; ~[...] without any modifiers chooses one of the clauses based
;; on the param or next argument
;; TODO check arg is positive int
(defn- choice-conditional [params arg-navigator offsets]
  (let [arg (:selector params)

```



```

[arg-list navigator] (next-arg navigator)
args (init-navigator arg-list)]
(loop [count 0
       args args
       last-pos (num -1)]
  (if (and (not max-count) (= (:pos args) last-pos) (> count 1))
      ;; TODO get the offset in here and call format exception
      (throw (RuntimeException.
              "%{ construct not consuming any arguments: Infinite loop!}"))
    (if (or (and (empty? (:rest args))
                 (or (not (:colon (:right-params params)))
                     (> count 0)))
              (and max-count (>= count max-count)))
        navigator
        (let [iter-result
              (execute-sub-format clause args (:base-args params))]
          (if (= :up-arrow (first iter-result))
              navigator
              (recur (inc count) iter-result (:pos args)))))))

;; ~:{...~} with the colon treats the next argument as a list of
;; sublists. Each of the sublists is used as the arglist for a
;; single iteration.
(defn- iterate-list-of-sublists [params navigator offsets]
  (let [max-count (:max-iterations params)
        param-clause (first (:clauses params))
        [clause navigator] (if (empty? param-clause)
                               (get-format-arg navigator)
                               [param-clause navigator])
        [arg-list navigator] (next-arg navigator)])
    (loop [count 0
           arg-list arg-list]
      (if (or (and (empty? arg-list)
                   (or (not (:colon (:right-params params)))
                       (> count 0)))
              (and max-count (>= count max-count)))
          navigator
          (let [iter-result (execute-sub-format
                            clause
                            (init-navigator (first arg-list))
                            (init-navigator (next arg-list)))]
            (if (= :colon-up-arrow (first iter-result))
                navigator
                (recur (inc count) (next arg-list))))))

;; ~@{...~} with the at sign uses the main argument list as the a
;; rguments to the iterations is consumed by all the iterations
(defn- iterate-main-list [params navigator offsets]
  (let [max-count (:max-iterations params)
        param-clause (first (:clauses params))

```

```

[clause navigator] (if (empty? param-clause)
                         (get-format-arg navigator)
                         [param-clause navigator])]

(loop [count 0
       navigator navigator
       last-pos (num -1)]
  (if (and (not max-count) (= (:pos navigator) last-pos) (> count 1))
      ;; TODO get the offset in here and call format exception
      (throw (RuntimeException.
              "%@{ construct not consuming any arguments: Infinite loop!}"))
      (if (or (and (empty? (:rest navigator))
                    (or (not (:colon (:right-params params)))
                        (> count 0)))
                    (and max-count (>= count max-count)))
              navigator
              (let [iter-result
                    (execute-sub-format clause navigator (:base-args params))]
                (if (= :up-arrow (first iter-result))
                    (second iter-result)
                    (recur
                      (inc count) iter-result (:pos navigator)))))))

;; ~@:{...~} with both colon and at sign uses the main argument list
;; as a set of sublists, one of which is consumed with each iteration
(defn- iterate-main-sublists [params navigator offsets]
  (let [max-count (:max-iterations params)
        param-clause (first (:clauses params))
        [clause navigator] (if (empty? param-clause)
                              (get-format-arg navigator)
                              [param-clause navigator])
        ]
    (loop [count 0
           navigator navigator]
      (if (or (and (empty? (:rest navigator))
                    (or (not (:colon (:right-params params)))
                        (> count 0)))
                    (and max-count (>= count max-count)))
              navigator
              (let [[sublist navigator] (next-arg-or-nil navigator)
                    iter-result
                    (execute-sub-format clause
                        (init-navigator sublist) navigator)]
                (if (= :colon-up-arrow (first iter-result))
                    navigator
                    (recur (inc count) navigator))))))

;;;;;;
;;;;; The '^<' directive has two completely different meanings
;;;;; in the '^<...~>' form it does justification, but with
;;;;; '^<...~:>' it represents the logical block operation of the

```

```

;;; pretty printer.
;;;
;;; Unfortunately, the current architecture decides what function
;;; to call at form parsing time before the sub-clauses have been
;;; folded, so it is left to run-time to make the decision.
;;;
;;; TODO: make it possible to make these decisions at compile-time.
;;;;;;;;
;;;

(declare format-logical-block)
(declare justify-clauses)

(defn- logical-block-or-justify [params navigator offsets]
  (if (:colon (:right-params params))
      (format-logical-block params navigator offsets)
      (justify-clauses params navigator offsets)))

;;;;;;
;;; Support for the '^<...^>' justification directive
;;;;;;

(defn- render-clauses [clauses navigator base-navigator]
  (loop [clauses clauses
         acc []]
    (if (empty? clauses)
        acc
        (let [clause (first clauses)
              iter-result result-str
              binding
              [*out* (java.io.StringWriter.)]
              [(execute-sub-format clause navigator base-navigator)
               (.toString *out*)]]
          (if (= :up-arrow (first iter-result))
              [acc (second iter-result)]
              (recur (next clauses) (conj acc result-str) iter-result))))))

;; TODO support for ::; constructions
(defn- justify-clauses [params navigator offsets]
  (let [[[eol-str] new-navigator]
        (when-let [else (:else params)]
          (render-clauses else navigator (:base-args params)))
        navigator (or new-navigator navigator)
        [else-params new-navigator]
        (when-let [p (:else-params params)]
          (realize-parameter-list p navigator))
        navigator (or new-navigator navigator)
        min-remaining (or (first (:min-remaining else-params)) 0)
        max-columns (or (first (:max-columns else-params))
                        (get-max-column *out*)))
        (get-max-column *out*))])

```

```

clauses (:clauses params)
[strs navigator]
  (render-clauses clauses navigator (:base-args params))
slots
(max 1
  (+ (dec (count strs))
      (if (:colon params) 1 0) (if (:at params) 1 0)))
chars (reduce + (map count strs))
mincol (:mincol params)
minpad (:minpad params)
colinc (:colinc params)
minout (+ chars (* slots minpad))
result-columns
(if (<= minout mincol)
  mincol
  (+ mincol (* colinc
    (+ 1 (quot (- minout mincol 1) colinc)))))

total-pad (- result-columns chars)
pad (max minpad (quot total-pad slots))
extra-pad (- total-pad (* pad slots))
pad-str (apply str (repeat pad (:padchar params)))]
(if (and eol-str
  (> (+ (get-column (:base @@*out*))
    min-remaining result-columns)
    max-columns)
  (print eol-str))
(loop [slots slots
  extra-pad extra-pad
  strs strs
  pad-only (or (:colon params)
    (and (= (count strs) 1) (not (:at params))))]
  (if (seq strs)
    (do
      (print
        (str (if (not pad-only) (first strs)
          (if (or pad-only (next strs) (:at params)) pad-str
            (if (pos? extra-pad) (:padchar params)))))

      (recur
        (dec slots)
        (dec extra-pad)
        (if pad-only strs (next strs)
          false))))
    navigator))

;;;;;;
;; Support for case modification with ~(...~).
;; We do this by wrapping the underlying writer with
;; a special writer to do the appropriate modification. This
;; allows us to support arbitrary-sized output and sources
;; that may block.

```

```
;;;;;;;;;;;;;;;;
(defn- downcase-writer
  "Returns a proxy that wraps writer, converting all characters to
  lower case"
  [^java.io.Writer writer]
  (proxy [java.io.Writer] []
    (close [] (.close writer))
    (flush [] (.flush writer))
    (write ([^chars cbuf ^Integer off ^Integer len]
            (.write writer cbuf off len)))
    ([x]
     (condp = (class x)
       String
       (let [s ^String x]
         (.write writer (.toLowerCase s)))

       Integer
       (let [c ^Character x]
         (.write writer
                 (int (CharactertoLowerCase (char c)))))))

     (defn- upcase-writer
       "Returns a proxy that wraps writer, converting all characters to
       upper case"
       [^java.io.Writer writer]
       (proxy [java.io.Writer] []
         (close [] (.close writer))
         (flush [] (.flush writer))
         (write ([^chars cbuf ^Integer off ^Integer len]
                 (.write writer cbuf off len)))
         ([x]
          (condp = (class x)
            String
            (let [s ^String x]
              (.write writer (.toUpperCase s)))

            Integer
            (let [c ^Character x]
              (.write writer
                      (int (CharactertoUpperCase (char c)))))))

        (defn- capitalize-string
          "Capitalizes the words in a string. If first? is false, don't
          capitalize the first character of the string even if it's a letter."
          [s first?]
          (let [^Character f (first s)
                s (if (and first? f (Character/isLetter f))
                     (str (CharactertoUpperCase f) (subs s 1))
                     s)]
            ...)
```

```

  (apply str
    (first
      (consume
        (fn [s]
          (if (empty? s)
            [nil nil]
            (let [m (re-matcher #"\W\w" s)
                  match (re-find m)
                  offset (and match (inc (.start m)))]
              (if offset
                [(str (subs s 0 offset)
                      (Character/toUpperCase
                        ^Character (nth s offset)))
                 (subs s (inc offset))]
                [s nil]))))
        s)))))

(defn- capitalize-word-writer
  "Returns a proxy that wraps writer, captializing all words"
  [^java.io.Writer writer]
  (let [last-was-whitespace? (ref true)]
    (proxy [java.io.Writer] []
      (close [] (.close writer))
      (flush [] (.flush writer))
      (write
        ([^chars cbuf ^Integer off ^Integer len]
         (.write writer cbuf off len))
        ([x]
         (condp = (class x)
           String
           (let [s ^String x]
             (.write writer
                   ^String (capitalize-string (.toLowerCase s)
                                              @last-was-whitespace?))
             (dosync
               (ref-set last-was-whitespace?
                     (Character/isWhitespace
                       ^Character (nth s (dec (count s))))))))
           Integer
           (let [c (char x)]
             (let [mod-c (if @last-was-whitespace?
                           (Character/toUpperCase (char x))
                           c)]
               (.write writer (int mod-c))
               (dosync
                 (ref-set last-was-whitespace?
                     (Character/isWhitespace (char x)))))))))))
      (defn- init-cap-writer

```

```

"Returns a proxy that wraps writer, capitalizing the first word"
[^java.io.Writer writer]
(let [capped (ref false)]
  (proxy [java.io.Writer] []
    (close [] (.close writer))
    (flush [] (.flush writer))
    (write ([^chars cbuf ^Integer off ^Integer len]
            (.write writer cbuf off len)))
    ([x]
     (condp = (class x)
       String
       (let [s (.toLowerCase ^String x)]
         (if (not @capped)
             (let [m (re-matcher #"\S" s)
                   match (re-find m)
                   offset (and match (.start m))]
               (if offset
                   (do (.write writer
                           (str (subs s 0 offset)
                                 (Character/toUpperCase
                                   ^Character (nth s offset)))
                           (.toLowerCase
                             ^String (subs s (inc offset))))))
                   (dosync (ref-set capped true)))
               (.write writer s)))
             (.write writer (.toLowerCase s)))))

     Integer
     (let [c ^Character (char x)]
       (if (and (not @capped) (Character/isLetter c))
           (do
             (dosync (ref-set capped true))
             (.write writer (int (Character/toUpperCase c))))
           (.write writer
                 (int (Character/toLowerCase c))))))))))

(defn- modify-case [make-writer params navigator offsets]
  (let [clause (first (:clauses params))]
    (binding [*out* (make-writer *out*)]
      (execute-sub-format clause navigator (:base-args params)))))

;;;;;;;;
;; If necessary, wrap the writer in a PrettyWriter object
;;;;;;;

(defn get-pretty-writer
  "Returns the java.io.Writer passed in wrapped in a pretty writer
  proxy, unless it's already a pretty writer. Generally, it is
  unneccesary to call this function, since pprint, write, and
  cl-format all call it if they need to. However if you want the

```

state to be preserved across calls, you will want to wrap them with this.

For example, when you want to generate column-aware output with multiple calls to cl-format, do it like in this example:

```
(defn print-table [aseq column-width]
  (binding [*out* (get-pretty-writer *out*)]
    (doseq [row aseq]
      (doseq [col row]
        (cl-format true \"~4D~7,vT\" col column-width))
      (prn))))
```

Now when you run:

```
user> (print-table (map #(vector % (* % %) (* % % %)) (range 1 11)) 8)
```

It prints a table of squares and cubes for the numbers from 1 to 10:

```
1      1      1
2      4      8
3      9      27
4      16     64
5      25     125
6      36     216
7      49     343
8      64     512
9      81     729
10     100    1000"
{:added "1.2"}
[writer]
(if (pretty-writer? writer)
  writer
  (pretty-writer writer *print-right-margin* *print-miser-width*))

;;;;;;;;
;; Support for column-aware operations ~&, ~T
;;;;;;;

(defn fresh-line
  "Make a newline if *out* is not already at the beginning of the line.
  If *out* is not a pretty writer (which keeps track of columns), this
  function always outputs a newline."
  {:added "1.2"}
  []
  (if (instance? clojure.lang.IDeref *out*)
    (if (not (= 0 (get-column (:base @*out*))))
      (prn)
      (prn)))
```

```
(defn- absolute-tabulation [params navigator offsets]
  (let [colnum (:colnum params)
        colinc (:colinc params)
        current (get-column (:base @@*out*))
        space-count (cond
                      (< current colnum) (- colnum current)
                      (= colinc 0) 0
                      :else (- colinc (rem (- current colnum) colinc)))]
    (print (apply str (repeat space-count \space))))
  navigator)

(defn- relative-tabulation [params navigator offsets]
  (let [colrel (:colnum params)
        colinc (:colinc params)
        start-col (+ colrel (get-column (:base @@*out*)))
        offset (if (pos? colinc) (rem start-col colinc) 0)
        space-count (+ colrel (if (= 0 offset) 0 (- colinc offset)))]
    (print (apply str (repeat space-count \space))))
  navigator)

;;;;;;;;
;; Support for accessing the pretty printer from a format
;;;;;;;;
;; TODO: support ~@; per-line-prefix separator
;; TODO: get the whole format wrapped so we can start the lb at any
;; column
(defn- format-logical-block [params navigator offsets]
  (let [clauses (:clauses params)
        clause-count (count clauses)
        prefix (cond
                  (> clause-count 1)
                  (:string (:params (first (first clauses))))
                  (:colon params) "(")
        body (nth clauses (if (> clause-count 1) 1 0))
        suffix (cond
                  (> clause-count 2)
                  (:string (:params (first (nth clauses 2))))
                  (:colon params) ")")
        [arg navigator] (next-arg navigator)])
  (pprint-logical-block :prefix prefix :suffix suffix
    (execute-sub-format
      body
      (init-navigator arg)
      (:base-args params)))
  navigator))

(defn- set-indent [params navigator offsets]
  (let [relative-to (if (:colon params) :current :block)]
```

```

(pprint-indent relative-to (:n params))
  navigator))

;;; TODO: support ~:T section options for ~T

(defn- conditional-newline [params navigator offsets]
  (let [kind (if (:colon params)
                (if (:at params) :mandatory :fill)
                (if (:at params) :miser :linear))]
    (pprint-newline kind)
    navigator))

;;;;;;;;
;;;;;; The table of directives we support, each with its params,
;;;;; properties, and the compilation function
;;;;;;;;
;;;;;; We start with a couple of helpers
(defn- process-directive-table-element
  [ [ char params flags bracket-info & generator-fn ] ]
  [char,
   {:directive char,
    :params `(array-map ~@params),
    :flags flags,
    :bracket-info bracket-info,
    :generator-fn (concat '(fn [ params offset]) generator-fn)})]

(defmacro ^{:private true}
  defdirectives
  [& directives]
  `(def ^{:private true}
     directive-table
     (hash-map
      ~@(mapcat process-directive-table-element directives)))))

(defdirectives
 (\A
  [ :mincol [0 Integer] :colinc [1 Integer] :minpad [0 Integer]
   :padchar [\space Character] ]
  #{:at :colon :both} {}
  #(format-ascii print-str %1 %2 %3))

 (\S
  [ :mincol [0 Integer] :colinc [1 Integer] :minpad [0 Integer]
   :padchar [\space Character] ]
  #{:at :colon :both} {}
  #(format-ascii pr-str %1 %2 %3))

 (\D
  [ :mincol [0 Integer] :padchar [\space Character]

```

```

:commachar [\, Character] :commainterval [ 3 Integer]]
#{ :at :colon :both } {}
#(format-integer 10 %1 %2 %3))

(\B
[ :mincol [0 Integer] :padchar [\space Character]
:commachar [\, Character] :commainterval [ 3 Integer]]
#{ :at :colon :both } {}
#(format-integer 2 %1 %2 %3))

(\O
[ :mincol [0 Integer] :padchar [\space Character]
:commachar [\, Character] :commainterval [ 3 Integer]]
#{ :at :colon :both } {}
#(format-integer 8 %1 %2 %3))

(\X
[ :mincol [0 Integer] :padchar [\space Character]
:commachar [\, Character] :commainterval [ 3 Integer]]
#{ :at :colon :both } {}
#(format-integer 16 %1 %2 %3))

(\R
[:base [nil Integer] :mincol [0 Integer]
:padchar [\space Character] :commachar [\, Character]
:commainterval [ 3 Integer]]
#{ :at :colon :both } {}
(do
  (cond
    ; ~R is overloaded with bizarreness
    (first (:base params)) #(format-integer (:base %1) %1 %2 %3)
    (and (:at params) (:colon params)) #(format-old-roman %1 %2 %3)
    (:at params) #(format-new-roman %1 %2 %3)
    (:colon params) #(format-ordinal-english %1 %2 %3)
    true #(format-cardinal-english %1 %2 %3)))))

(\P
[]
#{ :at :colon :both } {}
(fn [params navigator offsets]
(let [navigator (if (:colon params)
  (relative-reposition navigator -1)
  navigator)
strs (if (:at params) ["y" "ies"] ["" "s"])
[arg navigator] (next-arg navigator)]
  (print (if (= arg 1) (first strs) (second strs)))
  navigator)))

(\C
[:char-format [nil Character]]
#{ :at :colon :both } {})

```

```

(cond
  (:colon params) pretty-character
  (:at params) readable-character
  :else plain-character))

(\F
 [ :w [nil Integer] :d [nil Integer] :k [0 Integer]
   :overflowchar [nil Character] :padchar [\space Character] ]
 #{:at} {}
 fixed-float)

(\E
 [ :w [nil Integer] :d [nil Integer] :e [nil Integer] :k [1 Integer]
   :overflowchar [nil Character] :padchar [\space Character]
   :exponentchar [nil Character] ]
 #{:at} {}
 exponential-float)

(\G
 [ :w [nil Integer] :d [nil Integer] :e [nil Integer] :k [1 Integer]
   :overflowchar [nil Character] :padchar [\space Character]
   :exponentchar [nil Character] ]
 #{:at} {}
 general-float)

(\$
 [ :d [2 Integer] :n [1 Integer] :w [0 Integer]
   :padchar [\space Character]]
 #{:at :colon :both} {}
 dollar-float)

(\%
 [ :count [1 Integer] ]
 #{} {}
 (fn [params arg-navigator offsets]
  (dotimes [i (:count params)]
   (prn))
  arg-navigator))

(\&
 [ :count [1 Integer] ]
 #{:pretty} {}
 (fn [params arg-navigator offsets]
  (let [cnt (:count params)]
   (if (pos? cnt) (fresh-line))
   (dotimes [i (dec cnt)]
    (prn))))
  arg-navigator))

(\|

```

```

[ :count [1 Integer] ]
#{ } {}
(fn [params arg-navigator offsets]
  (dotimes [i (:count params)]
    (print \formfeed))
  arg-navigator))

(\~
[ :n [1 Integer] ]
#{ } {}
(fn [params arg-navigator offsets]
  (let [n (:n params)]
    (print (apply str (repeat n \~)))
  arg-navigator))

(\newline ; Whitespace suppression is handled in the compilation loop
[ ]
#:colon :at {})
(fn [params arg-navigator offsets]
  (if (:at params)
    (prn))
  arg-navigator))

(\T
[ :colnum [1 Integer] :colinc [1 Integer] ]
#:at :pretty {})
(if (:at params)
  #(relative-tabulation %1 %2 %3)
  #(absolute-tabulation %1 %2 %3)))

(\*
[ :n [1 Integer] ]
#:colon :at {} {})
(fn [params navigator offsets]
  (let [n (:n params)]
    (if (:at params)
      (absolute-reposition navigator n)
      (relative-reposition navigator (if (:colon params) (- n) n)))
    )))
(\?
[ ]
#:at {} {})
(if (:at params)
  (fn [params navigator offsets] ; args from main arg list
    (let [[subformat navigator] (get-format-arg navigator)]
      (execute-sub-format subformat navigator
        (:base-args params))))
  (fn [params navigator offsets] ; args from sub-list
    (let [[subformat navigator] (get-format-arg navigator)

```

```

[subargs navigator] (next-arg navigator)
  sub-navigator (init-navigator subargs)]
  (execute-sub-format subformat sub-navigator
    (:base-args params))
  navigator)))))

(\(
 [ ]
#{ :colon :at :both} { :right \}, :allows-separator nil, :else nil }
(let [mod-case-writer (cond
  (and (:at params) (:colon params))
  uppercase-writer

  (:colon params)
  capitalize-word-writer

  (:at params)
  init-cap-writer

  :else
  downcase-writer)])
  #(modify-case mod-case-writer %1 %2 %3)))

(\) [] #{} {} nil)

(\[
 [ :selector [nil Integer] ]
#{ :colon :at } { :right \}, :allows-separator true, :else :last }
(cond
  (:colon params)
  boolean-conditional

  (:at params)
  check-arg-conditional

  true
  choice-conditional))

(\; [:min-remaining [nil Integer] :max-columns [nil Integer]]
#{ :colon } { :separator true } nil)

([] [] #{} {} nil)

(\{
 [ :max-iterations [nil Integer] ]
#{ :colon :at :both} { :right \}, :allows-separator false }
(cond
  (and (:at params) (:colon params))
  iterate-main-sublists

```

```

(:colon params)
iterate-list-of-sublists

(:at params)
iterate-main-list

true
iterate-sublist))

(\} [] #{:colon} {} nil)

(\<
  [::mincol [0 Integer] :colinc [1 Integer] :minpad [0 Integer]
   :padchar [\space Character]]
  #{:colon :at :both :pretty} { :right \>,
    :allows-separator true, :else :first }
  logical-block-or-justify)

(\> [] #{:colon} {} nil)

;; TODO: detect errors in cases where colon not allowed
(\^ [:arg1 [nil Integer] :arg2 [nil Integer] :arg3 [nil Integer]]
#{:colon} {}
(fn [params navigator offsets]
  (let [arg1 (:arg1 params)
        arg2 (:arg2 params)
        arg3 (:arg3 params)
        exit (if (:colon params) :colon-up-arrow :up-arrow)]
    (cond
      (and arg1 arg2 arg3)
      (if (<= arg1 arg2 arg3) [exit navigator] navigator)

      (and arg1 arg2)
      (if (= arg1 arg2) [exit navigator] navigator)

      arg1
      (if (= arg1 0) [exit navigator] navigator)

      true      ; TODO: handle looking up the arglist stack for info
      (if (if (:colon params)
               (empty? (:rest (:base-args params)))
               (empty? (:rest navigator)))
          [exit navigator] navigator)))))

(\W
[])
#{:at :colon :both} {}
(if (or (:at params) (:colon params))

```

```

(let [bindings (concat
                (if (:at params) [:level nil :length nil] [])
                (if (:colon params) [:pretty true] []))]
  (fn [params navigator offsets]
    (let [[arg navigator] (next-arg navigator)]
      (if (apply write arg bindings)
          [:up-arrow navigator]
          navigator))))
  (fn [params navigator offsets]
    (let [[arg navigator] (next-arg navigator)]
      (if (write-out arg)
          [:up-arrow navigator]
          navigator)))))

(\_
 []
 #{:at :colon :both} {}
 conditional-newline)

(\I
 [:n [0 Integer]]
 #{:colon} {}
 set-indent)
)

;;;;;;;;
;; Code to manage the parameters and flags associated with each
;; directive in the format string.
;;;;;;;;

(def ^{:private true}
  param-pattern #"(?<-[vV]|#|(.)|([+-]?\d+)|(=?,)")")
(def ^{:private true}
  special-params #{ :parameter-from-args :remaining-arg-count })

(defn- extract-param [[s offset saw-comma]]
  (let [m (re-matcher param-pattern s)
        param (re-find m)]
    (if param
        (let [token-str (first (re-groups m))
              remainder (subs s (.end m))
              new-offset (+ offset (.end m))]
          (if (not (= \, (nth remainder 0)))
              [ [token-str offset] [remainder new-offset false]]
              [ [token-str offset]
                [(subs remainder 1) (inc new-offset) true]]))
        (if saw-comma
            (format-error
              "Badly formed parameters in format directive" offset)
            [ nil [s offset]]))))
```

```

(defn- extract-params [s offset]
  (consume extract-param [s offset false]))

(defn- translate-param
  "Translate the string representation of a param to the internalized
   representation"
  [[^String p offset]]
  [(cond
      (= (.length p) 0) nil
      (and (= (.length p) 1)
            (contains? #{\v \V} (nth p 0))) :parameter-from-args
      (and (= (.length p) 1) (= \# (nth p 0))) :remaining-arg-count
      (and (= (.length p) 2) (= \' (nth p 0))) (nth p 1)
      true (new Integer p))
   offset])

(def ^{:private true}
  flag-defs { \: :colon, \@ :at })

(defn- extract-flags [s offset]
  (consume
   (fn [[s offset flags]]
     (if (empty? s)
         [nil [s offset flags]]
         (let [flag (get flag-defs (first s))]
           (if flag
               (if (contains? flags flag)
                   (format-error
                    (str
                     "Flag \""
                     (first s)
                     "\" appears more than once in a directive")
                     offset)
                   [true [(subs s 1) (inc offset)
                           (assoc flags flag [true offset])]])
               [nil [s offset flags]]))))
   [s offset {}]))

(defn- check-flags [def flags]
  (let [allowed (:flags def)]
    (if (and (not (:at allowed)) (:at flags))
        (format-error
         (str
          "\"@"
          " is an illegal flag for format directive \""
          (:directive def)
          "\""
          (nth (:at flags) 1)))
        (if (and (not (:colon allowed)) (:colon flags))
            (format-error (str
              "\"@"
              " is an illegal flag for format directive \""
              (:directive def)
              "\""
              (nth (:colon flags) 1)))
            (if (and (not (:both allowed)) (:at flags) (:colon flags))
                (format-error
                 (str
                  "\"@"
                  " is an illegal flag for format directive \""
                  (:directive def)
                  "\""
                  (nth (:at flags) 1)))))))))))

```

```

(format-error (str
  "Cannot combine \"@\" and \"\:\" flags for format directive \""
  (:directive def) "\"")
  (min (nth (:colon flags) 1) (nth (:at flags) 1)))))

(defn- map-params
  "Takes a directive definition and the list of actual parameters
  and a map of flags and returns a map of the parameters and flags
  with defaults filled in. We check to make sure that there are the
  right types and number of parameters as well."
  [def params flags offset]
  (check-flags def flags)
  (if (> (count params) (count (:params def)))
    (format-error
      (cl-format
        nil
        (str
          "Too many parameters for directive \"~C\": ~D~:* "
          "~[were~;was~:;were~] specified but only ~D~:* "
          "~[are~;is~:;are~] allowed")
        (:directive def) (count params) (count (:params def)))
        (second (first params))))
    (doall
      (map #(let [val (first %1)]
        (if (not (or (nil? val) (contains? special-params val)
                     (instance? (second (second %2)) val)))
            (format-error (str "Parameter " (name (first %2))
              " has bad type in directive \""
              (:directive def) "\": "
              (class val))
              (second %1)))
        params (:params def)))
      (merge
        ; create the result map
        (into (array-map) ; start with the default values,
              ; make sure the order is right
              (reverse (for [[name [default]] (:params def)]
                        [name [default offset]])))
        (reduce #(apply assoc %1 %2) {})
        (filter #(first (nth % 1)) ; add the specified parameters,
                ; filtering out nils
                (zipmap (keys (:params def)) params)))
        flags)) ; and finally add the flags

(defn- compile-directive [s offset]
  (let [[raw-params [rest offset]] (extract-params s offset)
        [_ [rest offset flags]] (extract-flags rest offset)
        directive (first rest)
        def (get directive-table
                  (Character/toUpperCase ^Character directive))
        params (if def

```

```

(map-params def (map translate-param raw-params)
            flags offset))]

(if (not directive)
    (format-error
     "Format string ended in the middle of a directive" offset))
(if (not def)
    (format-error (str "Directive \""
                       directive "\" is undefined")
                 offset))
[(struct compiled-directive ((:generator-fn def) params offset)
                           def params offset)
 (let [remainder (subs rest 1)
       offset (inc offset)
       trim? (and (= \newline (:directive def))
                  (not (:colon params)))
       trim-count
       (if trim? (prefix-count remainder [\space \tab]) 0)
       remainder (subs remainder trim-count)
       offset (+ offset trim-count)]
   [remainder offset])))

(defn- compile-raw-string [s offset]
  (struct compiled-directive
    (fn [_ a _] (print s) a) nil { :string s } offset))

(defn- right-bracket [this] (:right (:bracket-info (:def this))))
(defn- separator? [this] (:separator (:bracket-info (:def this))))
(defn- else-separator? [this]
  (and (:separator (:bracket-info (:def this)))
       (:colon (:params this)))))

(declare collect-clauses)

(defn- process-bracket [this remainder]
  (let [[subex remainder] (collect-clauses (:bracket-info (:def this))
                                            (:offset this) remainder)]
    [(struct compiled-directive
        (:func this) (:def this)
        (merge (:params this) (tuple-map subex (:offset this)))
        (:offset this))
     remainder)]))

(defn- process-clause [bracket-info offset remainder]
  (consume
   (fn [remainder]
     (if (empty? remainder)
         (format-error "No closing bracket found." offset)
         (let [this (first remainder)
               remainder (next remainder)]
           (cond
             (and (= (:offset this) offset)
                  (= (:offset (last remainder)) offset))
               (process-clause bracket-info offset remainder)))))))

(defn- consume [this offset]
  (let [remainder (rest this)]
    (if (= (:offset this) offset)
        (process-clause (:bracket-info this) offset remainder)
        (consume this offset)))
  remainder)

```

```

(right-bracket this)
(process-bracket this remainder)

(= (:right bracket-info) (:directive (:def this)))
[ nil [:right-bracket (:params this) nil remainder]]

(else-separator? this)
[nil [:else nil (:params this) remainder]]

(separator? this)
[nil [:separator nil nil remainder]] ;; TODO: check to make
;; sure that there are
;; no params on ~;

true
[this remainder)))))

remainder))

(defn- collect-clauses [bracket-info offset remainder]
(second
(consume
(fn [[clause-map saw-else remainder]]
(let [[clause [type right-params else-params remainder]]
(process-clause bracket-info offset remainder)])
(cond
(= type :right-bracket)
[nil [(merge-with concat clause-map
{(if saw-else :else :clauses) [clause]
:right-params right-params})
remainder]]]

(= type :else)
(cond
(:else clause-map)
(format-error
"Two else clauses (\\"~:;\\") inside bracket construction."
offset)

(not (:else bracket-info))
(format-error (str
"An else clause (\\"~:;\\") is in a bracket type that"
" doesn't support it." )
offset)

(and (= :first (:else bracket-info))
(seq (:clauses clause-map)))
(format-error (str
"The else clause (\\"~:;\\") is only allowed in the first "
"position for this directive." )
offset)

```

```

true      ; if the `;;` is in the last position, the else clause
          ; is next, this was a regular clause
(if (= :first (:else bracket-info))
  [true [(merge-with concat clause-map
    { :else [clause] :else-params else-params})
    false remainder]]
  [true [(merge-with concat clause-map { :clauses [clause] })
    true remainder]]))

(= type :separator)
(cond
  saw-else
  (format-error (str
    "A plain clause (with `~;`) follows an else "
    "clause (`~;`) inside bracket construction.") offset)

  (not (:allows-separator bracket-info))
  (format-error
    (str "A separator (`~;`) is in a bracket type that "
      "doesn't support it." )
    offset))

  true
  [true [(merge-with concat clause-map { :clauses [clause] })
    false remainder]])))
[{:clauses []} false remainder])))

(defn- process-nesting
  "Take a linearly compiled format and process the bracket directives
  to give it the appropriate tree structure"
  [format]
  (first
    (consume
      (fn [remainder]
        (let [this (first remainder)
              remainder (next remainder)
              bracket (:bracket-info (:def this))]

          (if (:right bracket)
            (process-bracket this remainder)
            [this remainder])))

      format)))

(defn- compile-format
  "Compiles format-str into a compiled format which can be used as
  an argument to cl-format just like a plain format string. Use this
  function for improved performance when you're using the same format
  string repeatedly"
  [format-str]
  ; (prlabel compiling format-str)

```

```

(binding [*format-str* format-str]
  (process-nesting
    (first
      (consume
        (fn [[^String s offset]]
          (if (empty? s)
              [nil s]
              (let [tilde (.indexOf s (int \~))]
                (cond
                  (neg? tilde)
                  [(compile-raw-string s offset)
                   ["" (+ offset (.length s))]]
                  (zero? tilde) (compile-directive (subs s 1) (inc offset))
                  true
                  [(compile-raw-string (subs s 0 tilde) offset)
                   [(subs s tilde) (+ tilde offset)]]))))
            [format-str 0]))))

(defn- needs-pretty
  "determine whether a given compiled format has any directives
  that depend on the column number or pretty printing"
  [format]
  (loop [format format]
    (if (empty? format)
        false
        (if (or (:pretty (:flags (:def (first format))))
                (some needs-pretty
                      (first (:clauses (:params (first format)))))))
            (some needs-pretty
                  (first (:else (:params (first format)))))))
        true
        (recur (next format)))))

(defn- execute-format
  "Executes the format with the arguments."
  {:skip-wiki true}
  ([stream format args]
   (let [^java.io.Writer real-stream
         (cond
           (not stream) (java.io.StringWriter.)
           (true? stream) *out*
           :else stream)
           ^java.io.Writer wrapped-stream
           (if (and (needs-pretty format)
                     (not (pretty-writer? real-stream)))
               (get-pretty-writer real-stream)
               real-stream)])
     (binding [*out* wrapped-stream]
       (try
         (execute-format format args)
         )))))

```

```

        (finally
          (if-not (identical? real-stream wrapped-stream)
            (.flush wrapped-stream)))
          (if (not stream) (.toString real-stream)))))

([format args]
  (map-passing-context
    (fn [element context]
      (if (abort? context)
        [nil context]
        (let [[params args] (realize-parameter-list
                             (:params element) context)
              [params offsets] (unzip-map params)
              params (assoc params :base-args args)]
          [nil (apply (:func element) [params args offsets])])))

      args
      format)
    nil))

;;; This is a bad idea, but it prevents us from leaking private symbols
;;; This should all be replaced by really compiled formats anyway.
(def ^{:private true} cached-compile (memoize compile-format))

(defmacro formatter
  "Makes a function which can directly run format-in. The function is
fn [stream & args] ... and returns nil unless the stream is nil (meaning
output to a string) in which case it returns the resulting string.

format-in can be either a control string or a previously compiled
format."
  {:added "1.2"}
  [format-in]
  '(let [format-in# ~format-in
         my-c-c# (var-get (get (ns-interns (the-ns 'clojure pprint))
                                'cached-compile))
         my-e-f# (var-get (get (ns-interns (the-ns 'clojure pprint))
                                'execute-format))
         my-i-n# (var-get (get (ns-interns (the-ns 'clojure pprint))
                                'init-navigator))
         cf# (if (string? format-in#) (my-c-c# format-in#) format-in#)]
    (fn [stream# & args#]
      (let [navigator# (my-i-n# args#)
            (my-e-f# stream# cf# navigator#)))))

(defmacro formatter-out
  "Makes a function which can directly run format-in. The function is
fn [& args] ... and returns nil. This version of the formatter macro is
designed to be used with *out* set to an appropriate Writer. In
particular, this is meant to be used as part of a pretty printer
dispatch method.

```

```

format-in can be either a control string or a previously compiled
format."
{:added "1.2"}
[format-in]
'(let [format-in# ~format-in
      cf# (if (string? format-in#)
             #'clojure.pprint/cached-compile format-in#) format-in#])
  (fn [& args#]
    (let [navigator# (#'clojure.pprint/init-navigator args#)]
      (#'clojure.pprint/execute-format cf# navigator#))))
```

11.18 column-writer.clj

— column-writer.clj —

```

\getchunk{Clojure Copyright}
;; column_writer.clj -- part of the pretty printer for Clojure

;; Author: Tom Faulhaber
;; April 3, 2009
;; Revised to use proxy instead of gen-class April 2010

;; This module implements a column-aware wrapper around an
;; instance of java.io.Writer

(in-ns 'clojure.pprint)

(import [clojure.lang IDeref]
        [java.io Writer])

(def ^:dynamic ^{:private true} *default-page-width* 72)

(defn- get-field [^Writer this sym]
  (sym @@this))

(defn- set-field [^Writer this sym new-val]
  (alter @this assoc sym new-val))

(defn- get-column [this]
  (get-field this :cur))

(defn- get-line [this]
  (get-field this :line))

(defn- get-max-column [this]
```

```

(get-field this :max))

(defn- set-max-column [this new-max]
  (dosync (set-field this :max new-max))
  nil)

(defn- get-writer [this]
  (get-field this :base))

(defn- c-write-char [^Writer this ^Integer c]
  (dosync (if (= c (int \newline))
    (do
      (set-field this :cur 0)
      (set-field this :line (inc (get-field this :line))))
    (set-field this :cur (inc (get-field this :cur))))))
  (.write ^Writer (get-field this :base) c))

(defn- column-writer
  ([writer] (column-writer writer *default-page-width*))
  ([writer max-columns]
   (let [fields (ref {:max max-columns, :cur 0,
                      :line 0 :base writer})]
     (proxy [Writer IDeref] []
       (deref [] fields)
       (write
         ([^chars cbuf ^Integer off ^Integer len]
          (let [^Writer writer (get-field this :base)]
            (.write writer cbuf off len)))
         ([x]
          (condp = (class x)
            String
            (let [^String s x
                  nl (.lastIndexOf s (int \newline))]
              (dosync (if (neg? nl)
                (set-field this :cur
                  (+ (get-field this :cur) (count s))))
                (do
                  (set-field this :cur (- (count s) nl 1))
                  (set-field this :line
                    (+ (get-field this :line)
                      (count (filter #(= % \newline) s)))))))
              (.write ^Writer (get-field this :base) s))

            Integer
            (c-write-char this x)
            Long
            (c-write-char this x))))))))

```

11.19 dispatch.clj

— dispatch.clj —

```
\getchunk{Clojure Copyright}
;; dispatch.clj -- part of the pretty printer for Clojure

;; Author: Tom Faulhaber
;; April 3, 2009

;; This module implements the default dispatch tables for pretty
;; printing code and data.

(in-ns 'clojure.pprint)

(defn- use-method
  "Installs a function as a new method of multimethod associated
  with dispatch-value. "
  [multifn dispatch-val func]
  (. multifn addMethod dispatch-val func))

;;;;;;
;; Implementations of specific dispatch table entries
;;;;;;

;;;;; Handle forms that can be "back-translated" to reader macros
;;;;; Not all reader macros can be dealt with this way or at all.
;;;;; Macros that we can't deal with at all are:
;;;; ; - The comment character is absorbed by the reader and never is
;;;;     part of the form
;;;; ' - Is fully processed at read time into a lisp expression
;;;;     (which will contain concats and regular quotes).
;;;; ~@ - Also fully eaten by the processing of ' and can't be used
;;;;     outside.
;;;; , - is whitespace and is lost (like all other whitespace).
;;;;     Formats can generate commas where they deem them useful
;;;;     to help readability.
;;;; ^ - Adding metadata completely disappears at read time and the
;;;;     data appears to be completely lost.
;;;;;
;;;; Most other syntax stuff is dealt with directly by the formats
;;;; (like (), [], {}, and #{}) or directly by printing the objects
;;;; using Clojure's built-in print functions (like :keyword, \char,
;;;; or ""). The notable exception is #() which is special-cased.

(def {:private true} reader-macros
  {'quote "'", 'clojure.core/deref "@",
   'var "#'", 'clojure.core/unquote "~"})
```

```
(defn- pprint-reader-macro [alis]
  (let [^String macro-char (reader-macros (first alis))]
    (when (and macro-char (= 2 (count alis)))
      (.write ^java.io.Writer *out* macro-char)
      (write-out (second alis))
      true)))

;;;;;;;;
;; Dispatch for the basic data types when interpreted
;; as data (as opposed to code).
;;;;;;;

;;; TODO: inline these formatter statements into funcs so that we
;;; are a little easier on the stack. (Or, do "real" compilation, a
;;; la Common Lisp)

;;; (def pprint-simple-list (formatter-out "~~<~@{~w~~ ~_~}~:>"))
(defn- pprint-simple-list [alis]
  (pprint-logical-block :prefix "(" :suffix ")"
    (loop [alis (seq alis)]
      (when alis
        (write-out (first alis))
        (when (next alis)
          (.write ^java.io.Writer *out* " ")
          (pprint-newline :linear)
          (recur (next alis)))))))

(defn- pprint-list [alis]
  (if-not (pprint-reader-macro alis)
    (pprint-simple-list alis)))

;;; (def pprint-vector (formatter-out "~~<[~;~@{~w~~ ~_~}~;]~:>"))
(defn- pprint-vector [avec]
  (pprint-logical-block :prefix "[" :suffix "]"
    (loop [aseq (seq avec)]
      (when aseq
        (write-out (first aseq))
        (when (next aseq)
          (.write ^java.io.Writer *out* " ")
          (pprint-newline :linear)
          (recur (next aseq)))))))

(def ^{:private true} pprint-array
  (formatter-out "~~<[~;~@{~w~~, ~:_~}~;]~:>"))

;;; (def pprint-map
;;;   (formatter-out "~~<{~;~@{~<~w~~ ~_~w~:~>~~, ~_~}~;~>"))
(defn- pprint-map [amap]
  (pprint-logical-block :prefix "{" :suffix "}"
```

```

(loop [aseq (seq amap)]
  (when aseq
    (pprint-logical-block
      (.write-out (ffirst aseq))
      (.write ^java.io.Writer *out* " ")
      (pprint-newline :linear)
      (.write-out (fnnext (first aseq))))
    (when (next aseq)
      (.write ^java.io.Writer *out* ", ")
      (pprint-newline :linear)
      (recur (next aseq)))))

(def ^{:private true} pprint-set
  (formatter-out "^\^<#\{^\^;^\^@{^\^w^\^ ~^\^:_^\^}^\^;^\^>^\^"))

(def ^{:private true}
  type-map {"core$future_call" "Future",
            "core$promise" "Promise"})

(defn- map-ref-type
  "Map ugly type names to something simpler"
  [name]
  (or (when-let [match (re-find #"\^\[^$]+\$\[^$]+\^\ name")]
        (type-map match))
       name))

(defn- pprint-ideref [o]
  (let [prefix (format "#\^<%s@\%x%\^s: "
                        (map-ref-type (.getSimpleName (class o)))
                        (System/identityHashCode o)
                        (if (and (instance? clojure.lang.Agent o)
                                 (agent-error o))
                            " FAILED"
                            ""))]
    (pprint-logical-block :prefix prefix :suffix ">"
      (pprint-indent :block (-> (count prefix) (- 2) -))
      (pprint-newline :linear)
      (write-out
        (cond
          (and (future? o) (not (future-done? o))) :pending
          (and (instance? clojure.lang.IPromiseImpl o)
               (not (.hasValue o))) :not-delivered
          :else @o)))))

(def ^{:private true} pprint-pqueue
  (formatter-out "^\^<<-(^\^;^\^@{^\^w^\^ ~^\^:_^\^}^\^;^\^)-<^\^>^\^"))

(defn- pprint-simple-default [obj]
  (cond
    (.isArray (class obj)) (pprint-array obj)
    
```

```

(and *print-suppress-namespaces* (symbol? obj)) (print (name obj))
:else (pr obj)))

(defmulti
  simple-dispatch
  "The pretty print dispatch function for simple data structure format."
  {:added "1.2" :arglists '[[object]]}
  class)

(use-method simple-dispatch clojure.lang.ISeq pprint-list)
(use-method simple-dispatch clojure.lang.IPersistentVector pprint-vector)
(use-method simple-dispatch clojure.lang.IPersistentMap pprint-map)
(use-method simple-dispatch clojure.lang.IPersistentSet pprint-set)
(use-method simple-dispatch clojure.lang.PersistentQueue pprint-pqueue)
(use-method simple-dispatch clojure.lang.IDeref pprint-ideref)
(use-method simple-dispatch nil pr)
(use-method simple-dispatch :default pprint-simple-default)

;;;;;;
;; Dispatch for the code table
;;;;;;

(declare pprint-simple-code-list)

;;;;;;
;; Format something that looks like a simple def (sans metadata,
;; since the reader won't give it to us now).
;;;;;;

(def ^{:private true} pprint-hold-first
  (formatter-out "``{:~w`^`@`_`w`^`_`~@`{`~w`^`_`~}`~`>`"))
  ;;; Format something that looks like a defn or defmacro
  ;;; Format the params and body of a defn with a single arity
(defn- single-defn [alis has-doc-str?]
  (if (seq alis)
    (do
      (if has-doc-str?
        ((formatter-out " ``_`"))
        ((formatter-out " ``@`")))
      ((formatter-out " ``{`~w`^`_`~}` ``)`") alis)))
  ;;; Format the param and body sublists of a defn with multiple arities
(defn- multi-defn [alis has-doc-str?]
  (if (seq alis)
    ((formatter-out " ``_`{`~w`^`_`~}` ``)`") alis)))

```

```

;;; TODO: figure out how to support capturing metadata
;;; in defns (we might need a special reader)
(defn- pprint-defn [alis]
  (if (next alis)
      (let [[defn-sym defn-name & stuff] alis
            [doc-str stuff] (if (string? (first stuff))
                               [(first stuff) (next stuff)]
                               [nil stuff])
            [attr-map stuff] (if (map? (first stuff))
                               [(first stuff) (next stuff)]
                               [nil stuff])]
        (pprint-logical-block :prefix "(" :suffix ")"
          ((formatter-out "^\w~1I~@\w") defn-sym defn-name)
          (if doc-str
              ((formatter-out "^\w~@\w") doc-str))
          (if attr-map
              ((formatter-out "^\w~@\w") attr-map)))
        ;; Note: the multi-defn case will work OK for malformed defns too
        (cond
          (vector? (first stuff))
          (single-defn stuff (or doc-str attr-map))
          :else (multi-defn stuff (or doc-str attr-map))))))
  (pprint-simple-code-list alis)))

;;;;;;
;;;;; Format something with a binding form
;;;;;

(defn- pprint-binding-form [binding-vec]
  (pprint-logical-block :prefix "[" :suffix "]"
    (loop [binding binding-vec]
      (when (seq binding)
        (pprint-logical-block binding
          (write-out (first binding)))
        (when (next binding)
          (.write ^java.io.Writer *out* " ")
          (pprint-newline :miser)
          (write-out (second binding))))
        (when (next (rest binding))
          (.write ^java.io.Writer *out* " ")
          (pprint-newline :linear)
          (recur (next (rest binding))))))))
  (defn- pprint-let [alis]
    (let [base-sym (first alis)]
      (pprint-logical-block :prefix "(" :suffix ")"
        (if (and (next alis) (vector? (second alis)))
            (do
              ((formatter-out "^\w~1I~@\w") base-sym)

```

```

(pprint-binding-form (second alis))
((formatter-out " ~_~{~w~^ ~_~}") (next (rest alis))))
(pprint-simple-code-list alis)))))

;; Format something that looks like "if"
;;;

(def ^{:private true} pprint-if
  (formatter-out "~~<~1I~w~^ ~@_~w~@{ ~_~w~}~:>"))

(defn- pprint-cond [alis]
  (pprint-logical-block :prefix "(" :suffix ")"
    (pprint-indent :block 1)
    (write-out (first alis))
    (when (next alis)
      (.write ^java.io.Writer *out* " ")
      (pprint-newline :linear)
      (loop [alis (next alis)]
        (when alis
          (pprint-logical-block alis
            (write-out (first alis))
            (when (next alis)
              (.write ^java.io.Writer *out* " ")
              (pprint-newline :miser)
              (write-out (second alis))))
            (when (next (rest alis))
              (.write ^java.io.Writer *out* " ")
              (pprint-newline :linear)
              (recur (next (rest alis)))))))))))
    (when (next (rest alis))
      (.write ^java.io.Writer *out* " ")
      (pprint-newline :linear)
      (recur (next (rest alis))))))))))

(defn- pprint-condp [alis]
  (if (> (count alis) 3)
    (pprint-logical-block :prefix "(" :suffix ")"
      (pprint-indent :block 1)
      (apply (formatter-out " ~w ~@_~w ~@_~w ~_") alis)
      (loop [alis (seq (drop 3 alis))]
        (when alis
          (pprint-logical-block alis
            (write-out (first alis))
            (when (next alis)
              (.write ^java.io.Writer *out* " ")
              (pprint-newline :miser)
              (write-out (second alis))))
            (when (next (rest alis))
              (.write ^java.io.Writer *out* " ")
              (pprint-newline :linear)
              (recur (next (rest alis)))))))
        (when (next (rest alis))
          (.write ^java.io.Writer *out* " ")
          (pprint-newline :linear)
          (recur (next (rest alis)))))))
      (pprint-simple-code-list alis))))
```

```

;; The map of symbols that are defined in an enclosing #()
;; anonymous function
(def ^{:dynamic {:private true}} *symbol-map* {})

(defn- pprint-anon-func [alis]
  (let [args (second alis)
        nlis (first (rest (rest alis)))]
    (if (vector? args)
        (binding [*symbol-map* (if (= 1 (count args))
                                    {(first args) "%"}
                                    (into {})}
                           (map
                             #(vector %1 (str \% \%2))
                             args
                             (range 1 (inc (count args))))))]
        ((formatter-out "^\^<#(~;^\@{^\w^\^\^\_^\}^\^;)^>" nlis))
        (pprint-simple-code-list alis)))))

;;;;;;
;;;;; The master definitions for formatting lists in code (that
;;;; is, (fn args...) or special forms).
;;;;;

;;;;; This is the equivalent of (formatter-out "^\^<^\I^\@{^\w^\^\^\_^\}^\^>"),
;;;; but is easier on the stack.

(defn- pprint-simple-code-list [alis]
  (pprint-logical-block :prefix "(" :suffix ")"
    (pprint-indent :block 1)
    (loop [alis (seq alis)]
      (when alis
        (write-out (first alis))
        (when (next alis)
          (.write ^java.io.Writer *out* " ")
          (pprint-newline :linear)
          (recur (next alis))))))

;; Take a map with symbols as keys and add versions with no namespace.
;; That is, if ns/sym->val is in the map, add sym->val to the result.
(defn- two-forms [amap]
  (into {}
    (mapcat
      identity
      (for [x amap]
        [x [(symbol (name (first x))) (second x)]]))))

(defn- add-core-ns [amap]
  (let [core "clojure.core"]
    (into {}
```

```

(map #(let [[s f] %]
         (if (not (or (namespace s) (special-symbol? s)))
             [(symbol core (name s)) f]
             %))
      amap)))

(def ^{:dynamic {:private true} *code-table*
       (two-forms
        (add-core-ns
         {'def           pprint-hold-first,
          'defonce       pprint-hold-first,
          'defn          pprint-defn,
          'defn-         pprint-defn,
          'defmacro     pprint-defn,
          'fn            pprint-defn,
          'let           pprint-let,
          'loop          pprint-let,
          'binding       pprint-let,
          'with-local-vars pprint-let,
          'with-open     pprint-let,
          'when-let      pprint-let,
          'if-let         pprint-let,
          'doseq          pprint-let,
          'dotimes        pprint-let,
          'when-first    pprint-let,
          'if            pprint-if,
          'if-not        pprint-if,
          'when          pprint-if,
          'when-not      pprint-if,
          'cond          pprint-cond,
          'condp         pprint-condp,
          'fn*           pprint-anon-func,
          '.
          '..           pprint-hold-first,
          '->          pprint-hold-first,
          'locking      pprint-hold-first,
          'struct        pprint-hold-first,
          'struct-map   pprint-hold-first,
         })))
       }

(defn- pprint-code-list [alis]
  (if-not (pprint-reader-macro alis)
    (if-let [special-form (*code-table* (first alis))]
        (special-form alis)
        (pprint-simple-code-list alis)))

(defn- pprint-code-symbol [sym]
  (if-let [arg-num (sym *symbol-map*)]
    (print arg-num)
    (if *print-suppress-namespaces*

```

```

  (print (name sym))
  (pr sym)))))

(defmulti
  code-dispatch
  "The pretty print dispatch function for pretty printing Clojure code."
  {:added "1.2" :arglists '[[object]]}
  class)

(use-method code-dispatch clojure.lang.ISeq pprint-code-list)
(use-method code-dispatch clojure.lang.Symbol pprint-code-symbol)

;; The following are all exact copies of simple-dispatch
(use-method code-dispatch clojure.lang.IPersistentVector pprint-vector)
(use-method code-dispatch clojure.lang.IPersistentMap pprint-map)
(use-method code-dispatch clojure.lang.IPersistentSet pprint-set)
(use-method code-dispatch clojure.lang.PersistentQueue pprint-pqueue)
(use-method code-dispatch clojure.lang.IDeref pprint-ideref)
(use-method code-dispatch nil pr)
(use-method code-dispatch :default pprint-simple-default)

(set-pprint-dispatch simple-dispatch)

;;; For testing
(comment

(with-pprint-dispatch code-dispatch
  (pprint
   '(defn cl-format
      "An implementation of a Common Lisp compatible format function"
      [stream format-in & args]
      (let [compiled-format
            (if (string? format-in)
                (compile-format format-in)
                format-in)
            navigator (init-navigator args)]
        (execute-format stream compiled-format navigator)))))

(with-pprint-dispatch code-dispatch
  (pprint
   '(defn cl-format
      [stream format-in & args]
      (let [compiled-format
            (if (string? format-in)
                (compile-format format-in)
                format-in)
            navigator (init-navigator args)]
        (execute-format stream compiled-format navigator))))
```

```
(with-pprint-dispatch code-dispatch
  ( pprint
    '(defn- -write
      ([this x]
        (condp = (class x)
          String
            (let [s0 (write-initial-lines this x)
                  s (.replaceFirst s0 "\\s+$" "")]
              white-space (.substring s0 (count s))
              mode (getf :mode)])
          (if (= mode :writing)
              (dosync
                (write-white-space this)
                (.col-write this s)
                (setf :trailing-white-space white-space))
              (add-to-buffer this (make-buffer-blob s white-space)))))

        Integer
        (let [c ^Character x]
          (if (= (getf :mode) :writing)
              (do
                (write-white-space this)
                (.col-write this x))
              (if (= c (int \newline))
                  (write-initial-lines this "\n")
                  (add-to-buffer this
                    (make-buffer-blob (str (char c)) nil)))))))))

(with-pprint-dispatch code-dispatch
  ( pprint
    '(defn pprint-defn [writer alis]
      (if (next alis)
          (let [[defn-sym defn-name & stuff] alis
                [doc-str stuff] (if (string? (first stuff))
                                  [(first stuff) (next stuff)]
                                  [nil stuff])
                [attr-map stuff] (if (map? (first stuff))
                                  [(first stuff) (next stuff)]
                                  [nil stuff])]
            (pprint-logical-block writer :prefix "(" :suffix ")"
              (cl-format true "~w ~1I~@_~w" defn-sym defn-name)
              (if doc-str (cl-format true " ~_~w" doc-str))
              (if attr-map (cl-format true " ~_~w" attr-map))
              ;; Note: the multi-defn case will work OK for
              ;; malformed defns too
              (cond
                (vector? (first stuff))
                (single-defn stuff (or doc-str attr-map))
                :else (multi-defn stuff (or doc-str attr-map))))))
      ( pprint-simple-code-list writer alis))))
```

```
)  
nil
```

11.20 pprint'base.clj

— pprint'base.clj —

```
\getchunk{Clojure Copyright}  
;; pprint_base.clj -- part of the pretty printer for Clojure  
  
;; Author: Tom Faulhaber  
;; April 3, 2009  
  
;; This module implements the generic pretty print functions and  
;; special variables  
  
(in-ns 'clojure pprint)  
  
;;;;;;;  
;; Variables that control the pretty printer  
;;;;;;;  
  
;;;  
;; *print-length*, *print-level* and *print-dup* are defined in  
;; clojure.core  
;; TODO: use *print-dup* here (or is it supplanted by other  
;; variables?)  
;; TODO: make dispatch items like "(let..." get counted in  
;; *print-length* constructs  
  
(def ^:dynamic  
  ^{:doc "Bind to true if you want write to use pretty printing",  
    :added "1.2"}  
  *print-pretty* true)  
  
(defonce ^:dynamic ; If folks have added stuff here, don't overwrite  
  ^{:doc "The pretty print dispatch function. Use with-pprint-dispatch  
          or set-pprint-dispatch to modify.",  
    :added "1.2"}  
  *print-pprint-dispatch* nil)  
  
(def ^:dynamic
```

```

^{:doc "Pretty printing will try to avoid anything going beyond this
       column. Set it to nil to have pprint let the line be arbitrarily
       long. This will ignore all non-mandatory newlines.",
  :added "1.2"}
*print-right-margin* 72)

(def ^:dynamic
  ^{:doc "The column at which to enter miser style. Depending on the
          dispatch table, miser style add newlines in more places to
          try to keep lines short allowing for further levels of
          nesting.",
  :added "1.2"}
*print-miser-width* 40)

;; TODO implement output limiting
(def ^:dynamic
  ^{:private true,
  :doc "Maximum number of lines to print in a pretty print instance
        (N.B. This is not yet used)"}
*print-lines* nil)

;; TODO: implement circle and shared
(def ^:dynamic
  ^{:private true,
  :doc "Mark circular structures (N.B. This is not yet used)"}
*print-circle* nil)

;; TODO: should we just use *print-dup* here?
(def ^:dynamic
  ^{:private true,
  :doc "Mark repeated structures rather than repeat them
        (N.B. This is not yet used)"}
*print-shared* nil)

(def ^:dynamic
  ^{:doc "Don't print namespaces with symbols. This is particularly
          useful when pretty printing the results of macro expansions"
  :added "1.2"}
*print-suppress-namespaces* nil)

;; TODO: support print-base and print-radix in cl-format
;; TODO: support print-base and print-radix in rationals
(def ^:dynamic
  ^{:doc "Print a radix specifier in front of integers and rationals.
          If *print-base* is 2, 8, or 16, then the radix specifier used
          is #b, #o, or #x, respectively. Otherwise the radix specifier
          is in the form #XXr where XX is the decimal value of
          *print-base* "
  :added "1.2"}
*print-radix* nil)

```

```
(def ^{:dynamic
  ^{:doc "The base to use for printing integers and rationals."
  :added "1.2"}
  *print-base* 10)

;;;;;;;;
;; Internal variables that keep track of where we are in the
;; structure
;;;;;;;;
(def ^{:dynamic ^{:private true} } *current-level* 0)

(def ^{:dynamic ^{:private true} } *current-length* nil)

;; TODO: add variables for length, lines.

;;;;;;;;
;; Support for the write function
;;;;;;;;
(declare format-simple-number)

(def ^{:private true} orig-pr pr)

(defn- pr-with-base [x]
  (if-let [s (format-simple-number x)]
    (print s)
    (orig-pr x)))

(def ^{:private true} write-option-table
  {;;array          *print-array*
   :base           'clojure.pprint/*print-base*, 
   ;;case           *print-case*, 
   :circle          'clojure.pprint/*print-circle*, 
   ;;escape          *print-escape*, 
   ;;gensym          *print-gensym*, 
   :length          'clojure.core/*print-length*, 
   :level           'clojure.core/*print-level*, 
   :lines            'clojure.pprint/*print-lines*, 
   :miser-width     'clojure.pprint/*print-miser-width*, 
   :dispatch         'clojure.pprint/*print-pprint-dispatch*, 
   :pretty           'clojure.pprint/*print-pretty*, 
   :radix            'clojure.pprint/*print-radix*, 
   :readably         'clojure.core/*print-readably*, 
   :right-margin     'clojure.pprint/*print-right-margin*, 
   :suppress-namespaces 'clojure.pprint/*print-suppress-namespaces*)}
```

```
(defmacro ^{:private true} binding-map [amap & body]
  (let []
    `(do
      (. clojure.lang.Var (pushThreadBindings ~amap))
      (try
        `@body
        (finally
          (. clojure.lang.Var (popThreadBindings)))))))

(defn- table-size [t m]
  (apply hash-map
    (mapcat
      #'(when-let [v (get t (key %))] [(find-var v) (val %)]) m)))

(defn- pretty-writer?
  "Return true iff x is a PrettyWriter"
  [x] (and (instance? clojure.lang.IDeref x) (:pretty-writer @x)))

(defn- make-pretty-writer
  "Wrap base-writer in a PrettyWriter with the specified right-margin
  and miser-width"
  [base-writer right-margin miser-width]
  (pretty-writer base-writer right-margin miser-width))

(defmacro ^{:private true} with-pretty-writer [base-writer & body]
  `(~(let [base-writer# ~base-writer
           new-writer# (not (pretty-writer? base-writer#))]
        (binding [*out* (if new-writer#
                           (make-pretty-writer base-writer#
                           *print-right-margin* *print-miser-width*)
                           base-writer#)]
          `@body
          (.flush *out*)))))

;;;TODO: if pretty print is not set, don't use pr but rather something
;;;;that respects *print-base*, etc.
(defn write-out
  "Write an object to *out* subject to the current bindings of the
  printer control variables. Use the kw-args argument to override
  individual variables for this call (and any recursive calls).

*out* must be a PrettyWriter if pretty printing is enabled.
This is the responsibility of the caller.

This method is primarily intended for use by pretty print dispatch
functions that already know that the pretty printer will have set up
their environment appropriately. Normal library clients should use
the standard \"write\" interface. "

```

```

{:added "1.2"}
[object]
(let [length-reached (and
                      *current-length*
                      *print-length*
                      (>= *current-length* *print-length*))]
  (if-not *print-pretty*
    (pr object)
    (if length-reached
      (print "...")
      (do
        (if *current-length*
          (set! *current-length* (inc *current-length*)))
        (*print-pprint-dispatch* object))))
  length-reached))

(defn write
  "Write an object subject to the current bindings of the printer
  control variables. Use the kw-args argument to override individual
  variables for this call (and any recursive calls). Returns the
  string result if :stream is nil or nil otherwise."

```

The following keyword arguments can be passed with values:

Keyword	Meaning and Default value
:stream	Writer for output or nil true (indicates *out*)
:base	Base to use for writing rationals Current value of *print-base*
:circle*	If true, mark circular structures Current value of *print-circle*
:length	Maximum elements to show in sublists Current value of *print-length*
:level	Maximum depth Current value of *print-level*
:lines*	Maximum lines of output Current value of *print-lines*
:miser-width	Width to enter miser mode Current value of *print-miser-width*
:dispatch	The pretty print dispatch function Current value of *print-pprint-dispatch*
:pretty	If true, do pretty printing Current value of *print-pretty*
:radix	If true, prepend a radix specifier Current value of *print-radix*
:readably*	If true, print readably Current value of *print-readably*
:right-margin	The column for the right margin Current value of *print-right-margin*
:suppress-namespaces	If true, no namespaces in symbols Current value of *print-suppress-namespaces*

```

* = not yet supported
"
{:added "1.2"}
[object & kw-args]
(let [options (merge {:stream true} (apply hash-map kw-args))]
  (binding-map (table-ize write-option-table options)
    (binding-map (if (or (not (= *print-base* 10))
                         *print-radix*)
                     #'pr pr-with-base)
                  {}))
  (let [optval (if (contains? options :stream)
                   (:stream options)
                   true)
        base-writer (condp = optval
                      nil (java.io.StringWriter.)
                      true *out*
                      optval)]
    (if *print-pretty*
        (with-pretty-writer base-writer
          (write-out object))
        (binding [*out* base-writer]
          (pr object)))
    (if (nil? optval)
        (.toString ^java.io.StringWriter base-writer))))))

(defn pprint
  "Pretty print object to the optional output writer. If the writer
  is not provided, print the object to the currently bound value
  of *out*."
  {:added "1.2"}
  ([object] (pprint object *out*))
  ([object writer]
   (with-pretty-writer writer
     (binding [*print-pretty* true]
       (binding-map
         (if (or (not (= *print-base* 10))
                 *print-radix*)
             #'pr pr-with-base)
             {}))
       (write-out object)))
   (if (not (= 0 (get-column *out*)))
       (prn)))))

(defmacro pp
  "A convenience macro that pretty prints the last thing output.
  This is exactly equivalent to (pprint *1)."
  {:added "1.2"}
  [] '(pprint *1))

```

```
(defn set- pprint-dispatch
  "Set the pretty print dispatch function to a function matching
  (fn [obj] ...) where obj is the object to pretty print. That
  function will be called with *out* set to a pretty printing
  writer to which it should do its printing.

For example functions, see simple-dispatch and code-dispatch in
clojure.pprint.dispatch.clj."
  {:added "1.2"}
  [function]
  (let [old-meta (meta #'*print- pprint-dispatch*)]
    (alter-var-root #'*print- pprint-dispatch* (constantly function))
    (alter-meta! #'*print- pprint-dispatch* (constantly old-meta)))
  nil)

(defmacro with- pprint-dispatch
  "Execute body with the pretty print dispatch function bound to
  function."
  {:added "1.2"}
  [function & body]
  `'(binding [*print- pprint-dispatch* ~function]
     ~@body))

;;;;;;;;
;; Support for the functional interface to the pretty printer
;;;;;;;;
(defn- parse-lb-options [opts body]
  (loop [body body
         acc []]
    (if (opts (first body))
        (recur (drop 2 body) (concat acc (take 2 body)))
        [(apply hash-map acc) body])))

(defn- check-enumerated-arg [arg choices]
  (if-not (choices arg)
    (throw
      (IllegalArgumentException.
        ;; TODO clean up choices string
        (str "Bad argument: " arg ". It must be one of " choices)))))

(defn- level-exceeded []
  (and *print-level* (>= *current-level* *print-level*)))

(defmacro pprint-logical-block
  "Execute the body as a pretty printing logical block with output to
  *out* which must be a pretty printing writer. When used from pprint
  or cl-format, this can be assumed.
```

This function is intended for use when writing custom dispatch functions.

Before the body, the caller can optionally specify options:

```
:prefix, :per-line-prefix, and :suffix."
{:added "1.2", :arglists '[[options* body]]}
[& args]
(let [[options body]
      (parse-lb-options #{:prefix :per-line-prefix :suffix} args)]
  `(do (if (#'clojure pprint/level-exceeded)
          (.write ^java.io.Writer *out* "#")
          (do
            (push-thread-bindings
             #'clojure pprint/*current-level*
             (inc (var-get #'clojure pprint/*current-level*))
             #'clojure pprint/*current-length* 0})
            (try
              (#'clojure pprint/start-block *out*
               `(~(#{:prefix options}
                     ~(#{:per-line-prefix options}
                         ~(#{:suffix options})
                         `~@body
                         (#'clojure pprint/end-block *out*))
               (finally
                 (pop-thread-bindings)))))))
        nil)))
(defn pprint-newline
  "Print a conditional newline to a pretty printing stream. kind
  specifies if the newline is :linear, :miser, :fill, or :mandatory."
```

This function is intended for use when writing custom dispatch functions.

Output is sent to *out* which must be a pretty printing writer."

```
{:added "1.2"}
[kind]
(check-enumerated-arg kind #{:linear :miser :fill :mandatory})
(nl *out* kind))

(defn pprint-indent
  "Create an indent at this point in the pretty printing stream. This
  defines how following lines are indented. relative-to can be either
  :block or :current depending whether the indent should be computed
  relative to the start of the logical block or the current column
  position. n is an offset."
```

This function is intended for use when writing custom dispatch functions.

```

Output is sent to *out* which must be a pretty printing writer."
{:added "1.2"}
[relative-to n]
(check-enumerated-arg relative-to #{:block :current})
(indent *out* relative-to n))

;; TODO a real implementation for pprint-tab
(defn pprint-tab
  "Tab at this point in the pretty printing stream. kind specifies
whether the tab is :line, :section, :line-relative, or
:section-relative.

Colnum and colinc specify the target column and the increment to
move the target forward if the output is already past the original
target.

This function is intended for use when writing custom dispatch
functions.

Output is sent to *out* which must be a pretty printing writer.

THIS FUNCTION IS NOT YET IMPLEMENTED."
{:added "1.2"}
[kind colnum colinc]
(check-enumerated-arg kind
  #{:line :section :line-relative :section-relative})
(throw (UnsupportedOperationException.
  "pprint-tab is not yet implemented")))

```

nil

11.21 pretty-writer.clj

— pretty-writer.clj —

```

\getchunk{Clojure Copyright}
;; pretty_writer.clj -- part of the pretty printer for Clojure

;; Author: Tom Faulhaber
;; April 3, 2009
;; Revised to use proxy instead of gen-class April 2010

;; This module implements a wrapper around a java.io.Writer which
;; implements the core of the XP algorithm.

```

```
(in-ns 'clojure pprint)

(import [clojure.lang IDeref]
        [java.io Writer])

;; TODO: Support for tab directives

;;;;;;
;;;;; Forward declarations
;;;;;;

(declare get-miser-width)

;;;;;;
;;;;; Macros to simplify dealing with types and classes. These are
;;;;; really utilities, but I'm experimenting with them here.
;;;;;;

(defmacro ^{:private true}
  getf
  "Get the value of the field a named by the argument
  (which should be a keyword)."
  [sym]
  `(~sym @@`'this))

(defmacro ^{:private true}
  setf [sym new-val]
  "Set the value of the field SYM to NEW-VAL"
  `(~alter @@`'this assoc ~sym ~new-val))

(defmacro ^{:private true}
  deftype [type-name & fields]
  (let [name-str (name type-name)]
    `(do
       (defstruct ~type-name :type-tag ~@fields)
       (alter-meta! #'~type-name assoc :private true)
       (defn- ~symbol (str "make-" name-str))
       [& vals#] (apply struct ~type-name ~(keyword name-str) vals#))
       (defn- ~symbol (str name-str "?"))
       [x#] (= (:type-tag x#) ~(keyword name-str))))))

;;;;;;
;;;;; The data structures used by pretty-writer
;;;;;;

(defstruct ^{:private true} logical-block
  :parent :section :start-col :indent
  :done-nl :intra-block-nl
```

```

          :prefix :per-line-prefix :suffix
          :logical-block-callback)

(defn- ancestor? [parent child]
  (loop [child (:parent child)]
    (cond
      (nil? child) false
      (identical? parent child) true
      :else (recur (:parent child)))))

(defstruct ^{:private true} section :parent)

(defn- buffer-length [l]
  (let [l (seq l)]
    (if l
        (- (:end-pos (last l)) (:start-pos (first l)))
        0)))

; A blob of characters (aka a string)
(deftype buffer-blob :data :trailing-white-space :start-pos :end-pos)

; A newline
(deftype nl-t :type :logical-block :start-pos :end-pos)

(deftype start-block-t :logical-block :start-pos :end-pos)

(deftype end-block-t :logical-block :start-pos :end-pos)

(deftype indent-t :logical-block :relative-to :offset :start-pos
           :end-pos)

;;;;;;;;;;;;;;;;
;; Functions to write tokens in the output buffer
;;;;;;;;;;;;;;;

(def ^{:private pp-newline
       (memoize #(System/getProperty "line.separator")))

(declare emit-nl)

(defmulti ^{:private true} write-token #(:type-tag %2))
(defmethod write-token :start-block-t [^Writer this token]
  (when-let [cb (getf :logical-block-callback)] (cb :start))
  (let [lb (:logical-block token)]
    (dosync
      (when-let [^String prefix (:prefix lb)]
        (.write (getf :base) prefix))
      (let [col (get-column (getf :base))]
        (ref-set (:start-col lb) col)
        (ref-set (:indent lb) col)))))
```

```

(defmethod write-token :end-block-t [^Writer this token]
  (when-let [cb (getf :logical-block-callback)] (cb :end))
  (when-let [^String suffix (:suffix (:logical-block token))]
    (.write (getf :base) suffix)))

(defmethod write-token :indent-t [^Writer this token]
  (let [lb (:logical-block token)]
    (ref-set (:indent lb)
      (+ (:offset token)
          (condp = (:relative-to token)
            :block @(:start-col lb)
            :current (get-column (getf :base))))))

(defmethod write-token :buffer-blob [^Writer this token]
  (.write (getf :base) ^String (:data token)))

(defmethod write-token :nl-t [^Writer this token]
;  (prlabel wt @(:done-nl (:logical-block token)))
;  (prlabel wt (:type token) (= (:type token) :mandatory))
  (if (or (= (:type token) :mandatory)
          (and (not (= (:type token) :fill))
               @(:done-nl (:logical-block token))))
      (emit-nl this token)
      (if-let [^String tws (getf :trailing-white-space)]
          (.write (getf :base) tws)))
  (dosync (setf :trailing-white-space nil)))

(defn- write-tokens [^Writer this tokens force-trailing-whitespace]
  (doseq [token tokens]
    (if-not (= (:type-tag token) :nl-t)
        (if-let [^String tws (getf :trailing-white-space)]
            (.write (getf :base) tws))
        (write-token this token)
        (setf :trailing-white-space (:trailing-white-space token)))
  (let [^String tws (getf :trailing-white-space)]
    (when (and force-trailing-whitespace tws)
      (.write (getf :base) tws)
      (setf :trailing-white-space nil)))))

;;;;;;;;
;;;;
;;;;;; emit-nl? method defs for each type of new line. This makes
;;;;; the decision about whether to print this type of new line.
;;;;;;;;
;;;;;;;;
(defn- tokens-fit? [^Writer this tokens]
;;  (prlabel tf? (get-column (getf :base) (buffer-length tokens))
  (let [maxcol (get-max-column (getf :base))]
    (or

```

```

  (nil? maxcol)
  (< (+ (get-column (getf :base)) (buffer-length tokens)) maxcol)))

(defn- linear-nl? [this lb section]
;  (prlabel lnl? @(:done-nl lb) (tokens-fit? this section))
  (or @(:done-nl lb)
      (not (tokens-fit? this section)))))

(defn- miser-nl? [^Writer this lb section]
  (let [miser-width (get-miser-width this)
        maxcol (get-max-column (getf :base))]
    (and miser-width maxcol
         (>= @(:start-col lb) (- maxcol miser-width))
         (linear-nl? this lb section)))))

(defmulti ^{:private true} emit-nl? (fn [t _ _ _] (:type t)))

(defmethod emit-nl? :linear [newl this section _]
  (let [lb (:logical-block newl)]
    (linear-nl? this lb section)))

(defmethod emit-nl? :miser [newl this section _]
  (let [lb (:logical-block newl)]
    (miser-nl? this lb section)))

(defmethod emit-nl? :fill [newl this section subsection]
  (let [lb (:logical-block newl)]
    (or @(:intra-block-nl lb)
        (not (tokens-fit? this subsection))
        (miser-nl? this lb section)))))

(defmethod emit-nl? :mandatory [_ _ _ _]
  true)

;;;;;;
;; Various support functions
;;;;;;

(defn- get-section [buffer]
  (let [nl (first buffer)
        lb (:logical-block nl)
        section (seq
                  (take-while #(not (and (nl-t? %)
                                         (ancestor? (:logical-block %) lb)))
                               (next buffer)))]
    [section (seq (drop (inc (count section)) buffer))]))

(defn- get-sub-section [buffer]
  (let [nl (first buffer)
        ...]
    ...))

```

```

lb (:logical-block nl)
section
  (seq
    (take-while
      #(let [nl-lb (:logical-block %)]
        (not (and (nl-t? %)
                   (or (= nl-lb lb) (ancestor? nl-lb lb))))))
      (next buffer)))
  section))

(defn- update-nl-state [lb]
  (dosync
    (ref-set (:intra-block-nl lb) false)
    (ref-set (:done-nl lb) true)
    (loop [lb (:parent lb)]
      (if lb
          (do (ref-set (:done-nl lb) true)
              (ref-set (:intra-block-nl lb) true)
              (recur (:parent lb)))))))

(defn- emit-nl [^Writer this nl]
  (.write (getf :base) (pp-newline))
  (dosync (setf :trailing-white-space nil))
  (let [lb (:logical-block nl)
        ^String prefix (:per-line-prefix lb)]
    (if prefix
        (.write (getf :base) prefix))
    (let [^String istr
          (apply str (repeat (- @(:indent lb) (count prefix))
                           \space))]
      (.write (getf :base) istr)
      (update-nl-state lb)))))

(defn- split-at-newline [tokens]
  (let [pre (seq (take-while #(not (nl-t? %)) tokens))]
    [pre (seq (drop (count pre) tokens))]))

;; Methods for showing token strings for debugging

(defmulti {:private true} tok :type-tag)
(defmethod tok :nl-t [token]
  (:type token))
(defmethod tok :buffer-blob [token]
  (str "\"" (:data token) (:trailing-white-space token) "\""))
(defmethod tok :default [token]
  (:type-tag token))
(defn- toks [toks] (map tok toks))

;; write-token-string is called when the set of tokens in the buffer
;; is longer than the available space on the line

```

```

(defn- write-token-string [this tokens]
  (let [[a b] (split-at-newline tokens)]
    (prlabel wts (toks a) (toks b))
    (if a (write-tokens this a false))
    (if b
        (let [[section remainder] (get-section b)
              newl (first b)]
          (prlabel wts (toks section))
          (prlabel wts (:type newl))
          (prlabel wts (toks remainder)))
        (let [do-nl (emit-nl? newl this section (get-sub-section b))
              result (if do-nl
                        (do
                          (prlabel emit-nl (:type newl))
                          (emit-nl this newl)
                          (next b))
                        b)
              long-section (not (tokens-fit? this result))
              result (if long-section
                        (let [rem2 (write-token-string this section)]
                          (prlabel recurse (toks rem2))
                          (if (= rem2 section)
                            (do ; If that didn't produce any output, it
                                ; has no nls so we'll force it
                                (write-tokens this section false)
                                remainder)
                            (into [] (concat rem2 remainder))))
                        result)
              ff (prlabel wts (toks result))
            ]
          result)))))

(defn- write-line [^Writer this]
  (dosync
    (loop [buffer (getf :buffer)]
      (prlabel wl1 (toks buffer))
      (setf :buffer (into [] buffer))
      (if (not (tokens-fit? this buffer))
        (let [new-buffer (write-token-string this buffer)]
          (prlabel wl new-buffer)
          (if-not (identical? buffer new-buffer)
            (recur new-buffer)))))

    ;; Add a buffer token to the buffer and see if it's time to start
    ;; writing
  (defn- add-to-buffer [^Writer this token]
    (prlabel a2b token)
    (dosync
      (setf :buffer (conj (getf :buffer) token)))
  
```

```

(if (not (tokens-fit? this (getf :buffer)))
  (write-line this)))

;; Write all the tokens that have been buffered
(defn- write-buffered-output [^Writer this]
  (write-line this)
  (if-let [buf (getf :buffer)]
    (do
      (write-tokens this buf true)
      (setf :buffer [])))

(defn- write-white-space [^Writer this]
  (when-let [^String tws (getf :trailing-white-space)]
    ; (prlabel wws (str "*" tws "*"))
    (.write (getf :base) tws)
    (dosync
      (setf :trailing-white-space nil)))

;; If there are newlines in the string, print the lines up until
;; the last newline, making the appropriate adjustments. Return
;; the remainder of the string
(defn- write-initial-lines
  [^Writer this ^String s]
  (let [lines (.split s "\n" -1)]
    (if (= (count lines) 1)
        s
        (dosync
          (let [^String prefix (:per-line-prefix
                                (first (getf :logical-blocks)))
                ^String l (first lines)]
            (if (= :buffering (getf :mode))
                (let [oldpos (getf :pos)
                      newpos (+ oldpos (count l))]
                  (setf :pos newpos)
                  (add-to-buffer this
                    (make-buffer-blob l nil oldpos newpos))
                  (write-buffered-output this)))
                (do
                  (write-white-space this)
                  (.write (getf :base) l)))
            (.write (getf :base) (int \newline))
            (doseq [^String l (next (butlast lines))]
              (.write (getf :base) l)
              (.write (getf :base) (pp-newline))
              (if prefix
                  (.write (getf :base) prefix)))
            (setf :buffering :writing)
            (last lines)))))))

```

```
(defn- p-write-char [^Writer this ^Integer c]
  (if (= (getf :mode) :writing)
      (do
        (write-white-space this)
        (.write (getf :base) c))
      (if (= c \newline)
          (write-initial-lines this "\n")
          (let [oldpos (getf :pos)
                newpos (inc oldpos)]
            (dosync
              (setf :pos newpos)
              (add-to-buffer this
                (make-buffer-blob (str (char c)) nil oldpos newpos)))))))

;;;;;;;;
;; Initialize the pretty-writer instance
;;;;;;;

(defn- pretty-writer [writer max-columns miser-width]
  (let [lb (struct logical-block nil nil (ref 0) (ref 0)
                     (ref false) (ref false))
        fields (ref {:pretty-writer true
                     :base (column-writer writer max-columns)
                     :logical-blocks lb
                     :sections nil
                     :mode :writing
                     :buffer []
                     :buffer-block lb
                     :buffer-level 1
                     :miser-width miser-width
                     :trailing-white-space nil
                     :pos 0})]
    (proxy [Writer IDeref] []
      (deref [] fields)

      (write
        ([x]
         ;(prlabel write x (getf :mode))
         (condp = (class x)
           String
           (let [^String s0 (write-initial-lines this x)
                 ^String s (.replaceFirst s0 "\\s+$ \" ")
                 white-space (.substring s0 (count s))
                 mode (getf :mode)]
             (dosync
               (if (= mode :writing)
                   (do
                     (write-white-space this)
```

```

        (.write (getf :base) s)
        (setf :trailing-white-space white-space))
      (let [oldpos (getf :pos)
            newpos (+ oldpos (count s0))]
        (setf :pos newpos)
        (add-to-buffer this
                      (make-buffer-blob s white-space oldpos newpos)))))

  Integer
  (p-write-char this x)
  Long
  (p-write-char this x)))))

(flush []
  (if (= (getf :mode) :buffering)
    (dosync
      (write-tokens this (getf :buffer) true)
      (setf :buffer []))
    (write-white-space this)))

(close []
  (.flush this)))))

;;;;;;;;
;; Methods for pretty-writer
;;;;;;;

(defn- start-block
  [^Writer this
   ^String prefix ^String per-line-prefix ^String suffix]
  (dosync
    (let [lb (struct logical-block (getf :logical-blocks) nil
                           (ref 0) (ref 0) (ref false) (ref false)
                           prefix per-line-prefix suffix)]
      (setf :logical-blocks lb)
      (if (= (getf :mode) :writing)
        (do
          (write-white-space this)
          (when-let [cb (getf :logical-block-callback)] (cb :start))
          (if prefix
            (.write (getf :base) prefix))
          (let [col (get-column (getf :base))]
            (ref-set (:start-col lb) col)
            (ref-set (:indent lb) col)))
        (let [oldpos (getf :pos)
              newpos (+ oldpos (if prefix (count prefix) 0))]
          (setf :pos newpos)
          (add-to-buffer this (make-start-block-t lb oldpos newpos)))))))

```

```

(defn- end-block [^Writer this]
  (dosync
    (let [lb (getf :logical-blocks)
          ^String suffix (:suffix lb)]
      (if (= (getf :mode) :writing)
          (do
            (write-white-space this)
            (if suffix
                (.write (getf :base) suffix)
                (when-let [cb (getf :logical-block-callback)] (cb :end)))
            (let [oldpos (getf :pos)
                  newpos (+ oldpos (if suffix (count suffix) 0))]
              (setf :pos newpos)
              (add-to-buffer this (make-end-block-t lb oldpos newpos))))
          (setf :logical-blocks (:parent lb)))))

(defn- nl [^Writer this type]
  (dosync
    (setf :mode :buffering)
    (let [pos (getf :pos)]
      (add-to-buffer this
        (make-nl-t type (getf :logical-blocks) pos pos)))))

(defn- indent [^Writer this relative-to offset]
  (dosync
    (let [lb (getf :logical-blocks)]
      (if (= (getf :mode) :writing)
          (do
            (write-white-space this)
            (ref-set (:indent lb)
                     (+ offset (condp = relative-to
                      :block @(:start-col lb)
                      :current (get-column (getf :base))))))
          (let [pos (getf :pos)]
            (add-to-buffer this
              (make-indent-t lb relative-to offset pos pos))))))

(defn- get-miser-width [^Writer this]
  (getf :miser-width))

(defn- set-miser-width [^Writer this new-miser-width]
  (dosync (setf :miser-width new-miser-width)))

(defn- set-logical-block-callback [^Writer this f]
  (dosync (setf :logical-block-callback f)))

```

11.22 print-table.clj

— print-table.clj —

```
\getchunk{Clojure Copyright}

(in-ns 'clojure pprint)

(defn print-table
  "Alpha - subject to change.
  Prints a collection of maps in a textual table. Prints table
  headings ks, and then a line of output for each row, corresponding
  to the keys in ks. If ks are not specified, use the keys of the
  first item in rows."
  {:added "1.3"}
  ([ks rows]
   (when (seq rows)
     (let [widths
           (map
             (fn [k]
               (apply max (count (str k))
                     (map #(count (str (get % k))) rows)))
             ks)
           fmts (map #(str "%" % "s") widths)
           fmt-row (fn [row]
                     (apply str
                           (interpose " | "
                           (for [[col fmt]
                                 (map vector (map #(get row %) ks) fmcts)]
                             (format fmt (str col))))))
                     header (fmt-row (zipmap ks ks))
                     bar (apply str (repeat (count header) "=")))]
       (println bar)
       (println header)
       (println bar)
       (doseq [row rows]
         (println (fmt-row row)))
       (println bar)))
     ([rows] (print-table (keys (first rows)) rows))))
```

—

11.23 utilities.clj

— utilities.clj —

```
\getchunk{Clojure Copyright}
;; utilities.clj -- part of the pretty printer for Clojure

;; Author: Tom Faulhaber
;; April 3, 2009

;; This module implements some utility function used in formatting
;; and pretty printing. The functions here could go in a more general
;; purpose library, perhaps.

(in-ns 'clojure pprint)

;;;;;;
;;;; Helper functions for digesting formats in the various
;;;; phases of their lives.
;;;; These functions are actually pretty general.
;;;;;;

(defn- map-passing-context [func initial-context lis]
  (loop [context initial-context
         lis lis
         acc []]
    (if (empty? lis)
        [acc context]
        (let [this (first lis)
              remainder (next lis)
              [result new-context] (apply func [this context])]
          (recur new-context remainder (conj acc result)))))

(defn- consume [func initial-context]
  (loop [context initial-context
         acc []]
    (let [[result new-context] (apply func [context])]
      (if (not result)
          [acc new-context]
          (recur new-context (conj acc result)))))

(defn- consume-while [func initial-context]
  (loop [context initial-context
         acc []]
    (let [[result continue new-context] (apply func [context])]
      (if (not continue)
          [acc context]
          (recur new-context (conj acc result)))))

(defn- unzip-map [m]
  "Take a map that has pairs in the value slots and produce a pair
  of maps, the first having all the first elements of the pairs
  and the second all the second elements of the pairs"
  [(into {} (for [[k [v1 v2]] m] [k v1]))]
```

```

(into {} (for [[k [v1 v2]] m] [k v2]))))

(defn- tuple-map [m v1]
  "For all the values, v, in the map, replace them with [v v1]"
  (into {} (for [[k v] m] [k [v v1]])))

(defn- rtrim [s c]
  "Trim all instances of c from the end of sequence s"
  (let [len (count s)]
    (if (and (pos? len) (= (nth s (dec (count s))) c))
      (loop [n (dec len)]
        (cond
          (neg? n) ""
          (not (= (nth s n) c)) (subs s 0 (inc n))
          true (recur (dec n))))
      s)))

(defn- ltrim [s c]
  "Trim all instances of c from the beginning of sequence s"
  (let [len (count s)]
    (if (and (pos? len) (= (nth s 0) c))
      (loop [n 0]
        (if (or (= n len) (not (= (nth s n) c)))
          (subs s n)
          (recur (inc n))))
      s)))

(defn- prefix-count [aseq val]
  "Return the number of times that val occurs at the start of
  sequence aseq, if val is a seq itself, count the number of
  times any element of val occurs at the beginning of aseq"
  (let [test (if (coll? val) (set val) #{val})]
    (loop [pos 0]
      (if (or (= pos (count aseq)) (not (test (nth aseq pos))))
        pos
        (recur (inc pos))))))

(defn- prerr [& args]
  "Println to *err*"
  (binding [*out* *err*]
    (apply println args)))

(defmacro ^{:private true} prlabel [prefix arg & more-args]
  "Print args to *err* in name = value format"
  `'(prerr ~@(~(cons
    (list 'quote prefix)
    (mapcat #'(list (list 'quote %) "=" %)
      (cons arg (seq more-args)))))))

```

11.24 pprint.clj

— pprint.clj —

```
\getchunk{Clojure Copyright}
;; pprint.clj -- Pretty printer and Common Lisp compatible format
;; function (cl-format) for Clojure

;; Author: Tom Faulhaber
;; April 3, 2009

(ns
  ^{:author "Tom Faulhaber",
    :doc "A Pretty Printer for Clojure"

clojure.pprint implements a flexible system for printing structured
data in a pleasing, easy-to-understand format. Basic use of the
pretty printer is simple, just call pprint instead of println.
More advanced users can use the building blocks provided to create
custom output formats.

Out of the box, pprint supports a simple structured format for basic
data and a specialized format for Clojure source code. More advanced
formats, including formats that don't look like Clojure data at all
like XML and JSON, can be rendered by creating custom dispatch
functions.

In addition to the pprint function, this module contains cl-format,
a text formatting function which is fully compatible with the format
function in Common Lisp. Because pretty printing directives are
directly integrated with cl-format, it supports very concise custom
dispatch. It also provides a more powerful alternative to Clojure's
standard format function.

See documentation for pprint and cl-format for more information or
complete documentation on the the clojure web site on github.",
  :added "1.2"}
  clojure.pprint
  (:refer-clojure :exclude (deftype)))

(load "pprint/utilities")
(load "pprint/column_writer")
(load "pprint/pretty_writer")
(load "pprint/pprint_base")
```

```
(load "pprint/cl_format")
(load "pprint/dispatch")
(load "pprint/print_table")

nil
```

11.24.1 java.clj

— java.clj —

```
\getchunk{Clojure Copyright}
(in-ns 'clojure.reflect)

(require '[clojure.set :as set]
          '[clojure.string :as str])
(import '[clojure.asm ClassReader ClassVisitor Type]
        '[java.lang.reflect Modifier]
        java.io.InputStream)

(extend-protocol TypeReference
  clojure.lang.Symbol
  (typename [s] (str/replace (str s) "<>" "[]"))

  Class
  ;; neither .getName nor .get SimpleName returns the right thing,
  ;; so best to delegate to Type
  (typename
    [c]
    (typename (Type/getType c)))

  Type
  (typename
    [t]
    (-> (.getClassName t)))))

(defn- typesym
  "Given a typeref, create a legal Clojure symbol version of the
   type's name."
  [t]
  (-> (typename t)
       (str/replace "[]" "<>")
       (symbol)))

(defn- resource-name
  "Given a typeref, return implied resource name. Used by Reflectors
   such as ASM that need to find and read classbytes from files."
```

```
[typeref]
(-> (typename typeref)
      (str/replace "." "/")
      (str ".class")))

(defn- access-flag
  [[name flag & contexts]]
  {:name name :flag flag :contexts (set (map keyword contexts)))}

(defn- field-descriptor->class-symbol
  "Convert a Java field descriptor to a Clojure class symbol. Field
  descriptors are described in section 4.3.2 of the JVM spec, 2nd ed.:
  http://java.sun.com/docs/books/jvms/second_edition/html/
  ClassFile.doc.html#14152"
  [^String d]
  {:pre [(string? d)]}
  (typesym (Type/getType d)))

(defn- internal-name->class-symbol
  "Convert a Java internal name to a Clojure class symbol. Internal
  names uses slashes instead of dots, e.g. java/lang/String. See
  Section 4.2 of the JVM spec, 2nd ed.:
  http://java.sun.com/docs/books/jvms/second_edition/html/
  ClassFile.doc.html#14757"
  [d]
  {:pre [(string? d)]}
  (typesym (Type/getObjectType d)))

(def ^{:doc "The Java access bitflags, along with their friendly
           names and the kinds of objects to which they can apply."}
  flag-descriptors
  (vec
    (map access-flag
         [[:public 0x0001 :class :field :method]
          [:private 0x0002 :class :field :method]
          [:protected 0x0004 :class :field :method]
          [:static 0x0008 :field :method]
          [:final 0x0010 :class :field :method]
          ;; :super is ancient history and is unfindable (?) by
          ;; reflection. skip it
          #_[:super 0x0020 :class]
          [:synchronized 0x0020 :method]
          [:volatile 0x0040 :field]
          [:bridge 0x0040 :method]
          [:varargs 0x0080 :method]
          [:transient 0x0080 :field]
          [:native 0x0100 :method]
          [:interface 0x0200 :class]
          [:abstract 0x0400 :class :method]]))
```

```

[:strict 0x0800 :method]
[:synthetic 0x1000 :class :field :method]
[:annotation 0x2000 :class]
[:enum 0x4000 :class :field :inner]])))

(defn- parse-flags
  "Convert reflection bitflags into a set of keywords."
  [flags context]
  (reduce
   (fn [result fd]
     (if (and (get (:contexts fd) context)
              (not (zero? (bit-and flags (:flag fd))))))
         (conj result (:name fd))
         result)
   #{}
   flag-descriptors))

(defrecord Constructor
  [name declaring-class parameter-types exception-types flags])

(defn- constructor->map
  [^java.lang.reflect.Constructor constructor]
  (Constructor.
   (symbol (.getName constructor))
   (typesym (.getDeclaringClass constructor))
   (vec (map typesym (.getParameterTypes constructor)))
   (vec (map typesym (.getExceptionTypes constructor)))
   (parse-flags (.getModifiers constructor) :method)))

(defn- declared-constructors
  "Return a set of the declared constructors of class as a Clojure map."
  [^Class cls]
  (set (map
        constructor->map
        (.getDeclaredConstructors cls)))))

(defrecord Method
  [name return-type declaring-class parameter-types
   exception-types flags])

(defn- method->map
  [^java.lang.reflect.Method method]
  (Method.
   (symbol (.getName method))
   (typesym (.getReturnType method))
   (typesym (.getDeclaringClass method))
   (vec (map typesym (.getParameterTypes method)))
   (vec (map typesym (.getExceptionTypes method)))
   (parse-flags (.getModifiers method) :method)))

```

```
(defn- declared-methods
  "Return a set of the declared constructors of class as a Clojure map."
  [^Class cls]
  (set (map
    method->map
    (.getDeclaredMethods cls)))))

(defrecord Field
  [name type declaring-class flags])

(defn- field->map
  [^java.lang.reflect.Field field]
  (Field.
    (symbol (.getName field))
    (typesym (.getType field))
    (typesym (.getDeclaringClass field))
    (parse-flags (.getModifiers field) :field)))

(defn- declared-fields
  "Return a set of the declared fields of class as a Clojure map."
  [^Class cls]
  (set (map
    field->map
    (.getDeclaredFields cls)))))

(deftype JavaReflector [classloader]
  Reflector
  (do-reflect [_ typeref]
    (let [cls (Class.forName (typename typeref) false classloader)]
      {:bases (not-empty (set (map typesym (bases cls))))
       :flags (parse-flags (.getModifiers cls) :class)
       :members (set/union (declared-fields cls)
                           (declared-methods cls)
                           (declared-constructors cls))))))

(def ^:private default-reflector
  (JavaReflector. (.getContextClassLoader (Thread/currentThread)))))

(defn- parse-method-descriptor
  [^String md]
  {:parameter-types (vec (map typesym (Type/getArgumentTypes md)))
   :return-type (typesym (Type/getReturnType md))})

(defprotocol ClassResolver
  (^InputStream resolve-class [this name]
   "Given a class name, return that typeref's class bytes
   as an InputStream."))

(extend-protocol ClassResolver
  clojure.lang.Fn
```

```
(resolve-class [this typeref] (this typeref))

ClassLoader
(resolve-class [this typeref]
  (.getResourceAsStream this (resource-name typeref)))

(deftype AsmReflector [class-resolver]
  Reflector
  (do-reflect [_ typeref]
    (with-open [is (resolve-class class-resolver typeref)]
      (let [class-symbol (typesym typeref)
            r (ClassReader. is)
            result (atom {:bases #{} :flags #{} :members #{}})]
        (.accept
          r
          (reify
            ClassVisitor
            (visit [_ version access name signature superName interfaces]
              (let [flags (parse-flags access :class)
                    ; ignore java.lang.Object on interfaces to match reflection
                    superName (if (and (flags :interface)
                        (= superName "java/lang/Object"))
                      nil
                      superName)
                    bases (->> (cons superName interfaces)
                      (remove nil?))
                      (map internal-name->class-symbol)
                      (map symbol)
                      (set)
                      (not-empty))]
                (swap! result merge {:bases bases
                  :flags flags})))
            (visitSource [_ name debug])
            (visitInnerClass [_ name outerName innerName access])
            (visitField [_ access name desc signature value]
              (swap! result update-in [:members] (fn[conj] #{}))
              (Field. (symbol name)
                (field-descriptor->class-symbol desc)
                class-symbol
                (parse-flags access :field)))
              nil)
            (visitMethod [_ access name desc signature exceptions]
              (when-not (= name "<clinit>")
                (let [constructor? (= name "<init>")]
                  (swap! result update-in
                    [:members] (fn[conj] #{}))
                  (let [{:keys
                    [parameter-types return-type]}
                    (parse-method-descriptor desc)
                    flags
                    ]
                    (fn[conj] #{})))))))))))
```

```

          (parse-flags access :method)])
(if constructor?
  (Constructor. class-symbol
                class-symbol
                parameter-types
                (vec
                  (map
                    internal-name->class-symbol
                    exceptions)))
                flags)
  (Method. (symbol name)
    return-type
    class-symbol
    parameter-types
    (vec
      (map
        internal-name->class-symbol
        exceptions)))
    flags))))))
nil)
(visitEnd [])
) 0)
@result))))
```

11.25 reflect.clj

— reflect.clj —

```
\getchunk{Clojure Copyright}
(ns ^{:author "Stuart Halloway"
      :added "1.3"
      :doc "Reflection on Host Types"
      Alpha - subject to change.
```

Two main entry points:

- * type-reflect reflects on something that implements TypeReference.
- * reflect (for REPL use) reflects on the class of an instance, or on a class if passed a class

Key features:

- * Exposes the read side of reflection as pure data. Reflecting on a type returns a map with keys :bases, :flags, and :members.

- * Canonicalizes class names as Clojure symbols. Types can extend to the TypeReference protocol to indicate that they can be unambiguously resolved as a type name. The canonical format requires one non-Java-ish convention: array brackets are <> instead of [] so they can be part of a Clojure symbol.
- * Pluggable Reflectors for different implementations. The default JavaReflector is good when you have a class in hand, or use the AsmReflector for \"hands off\" reflection without forcing classes to load.

Platform implementers must:

```
* Create an implementation of Reflector.
* Create one or more implementations of TypeReference.
* def default-reflector to be an instance that satisfies Reflector."}
  clojure.reflect
  (:require [clojure.set :as set])))

(defprotocol Reflector
  "Protocol for reflection implementers."
  (do-reflect [reflector typeref]))

(defprotocol TypeReference
  "A TypeReference can be unambiguously converted to a type name on
  the host platform.

  All typerefs are normalized into symbols. If you need to
  normalize a typeref yourself, call typesym."
  (typename [o] "Returns Java name as returned by ASM getClassName,
  e.g. byte[], java.lang.String[]"))

(declare default-reflector)

(defn type-reflect
  "Alpha - subject to change.
  Reflect on a typeref, returning a map with :bases, :flags, and
  :members. In the discussion below, names are always Clojure symbols.

  :bases          a set of names of the type's bases
  :flags          a set of keywords naming the boolean attributes
                  of the type.
  :members        a set of the type's members. Each member is a map
                  and can be a constructor, method, or field.

  Keys common to all members:
  :name           name of the type
  :declaring-class name of the declarer
  :flags          keyword naming boolean attributes of the member"
  
```

```

Keys specific to constructors:
:parameter-types vector of parameter type names
:exception-types vector of exception type names

Key specific to methods:
:parameter-types vector of parameter type names
:exception-types vector of exception type names
:return-type      return type name

Keys specific to fields:
:type            type name

Options:

:ancestors      in addition to the keys described above, also
                 include an :ancestors key with the entire set of
                 ancestors, and add all ancestor members to
                 :members.
:reflector       implementation to use. Defaults to JavaReflector,
                 AsmReflector is also an option."
{:added "1.3"}
[typeref & options]
(let [[:keys [ancestors reflector]]
      (merge {:reflector default-reflector}
             (apply hash-map options))
      refl (partial do-reflect reflector)
      result (refl typeref)]
  ;; could make simpler loop of two args: names an
  (if ancestors
      (let [make-ancestor-map (fn [names]
                               (zipmap names (map refl names)))]
        (loop [reflections (make-ancestor-map (:bases result))]
              (let [ancestors-visited (set (keys reflections))
                    ancestors-to-visit
                    (set/difference (set (mapcat :bases (vals reflections)))
                                   ancestors-visited)]
                (if (seq ancestors-to-visit)
                    (recur
                     (merge reflections
                            (make-ancestor-map ancestors-to-visit)))
                    (apply merge-with into result
                           {:_ancestors ancestors-visited}
                           (map #(select-keys % [:members]) (vals reflections)))))))
        result)))
  (defn reflect
    "Alpha - subject to change.
     Reflect on the type of obj (or obj itself if obj is a class).
     Return value and options are the same as for type-reflect. "
  )

```

```
{:added "1.3"}
[obj & options]
(apply type-reflect (if (class? obj) obj (class obj)) options))

(load "reflect/java")
```

11.26 repl.clj

— repl.clj —

```
\getchunk{Houser Copyright}

; Utilities meant to be used interactively at the REPL

(ns
#^{:author
  "Chris Houser, Christophe Grand, Stephen Gilardi, Michel Salim"
:doc "Utilities meant to be used interactively at the REPL"}
clojure.repl
(:import (java.io LineNumberReader InputStreamReader PushbackReader)
(clojure.lang RT Reflector)))

(def ^{:private special-doc-map
'/. {:url "java_interop#dot"
:forms [(.instanceMember instance args*)
(.instanceMember Classname args*)
(Classname/staticMethod args*)
Classname/staticField]
:doc "The instance member form works for both fields and methods.
They all expand into calls to the dot operator at macroexpansion
time."}
def {:forms [(def symbol init?)]

:doc "Creates and interns a global var with the name
of symbol in the current namespace (*ns*) or locates such a var if
it already exists. If init is supplied, it is evaluated, and the
root binding of the var is set to the resulting value. If init is
not supplied, the root binding of the var is unaffected."
do {:forms [(do exprs*)]

:doc "Evaluates the expressions in order and returns the value
of the last. If no expressions are supplied, returns nil."
if {:forms [(if test then else?)]

:doc "Evaluates test. If not the singular values nil or false,
evaluates and yields then, otherwise, evaluates and yields else. If
else is not supplied it defaults to nil."
monitor-enter {:forms [(monitor-enter x)]}
```

```

        :doc "Synchronization primitive that should be
avoided in user code. Use the 'locking' macro."
monitor-exit {:forms [(monitor-exit x)]}
        :doc "Synchronization primitive that should be
avoided in user code. Use the 'locking' macro."
new {:forms [(Classname. args*) (new Classname args*)]}
        :url "java_interop#new"
        :doc "The args, if any, are evaluated from left to right, and
passed to the constructor of the class named by Classname. The
constructed object is returned."
quote {:forms [(quote form)]}
        :doc "Yields the unevaluated form."
recur {:forms [(recur exprs*)]}
        :doc "Evaluates the exprs in order, then, in parallel,
rebinds the bindings of the recursion point to the values of the
exprs. Execution then jumps back to the recursion point, a loop or
fn method."
set! {:forms[(set! var-symbol expr)
            (set! (. instance Expr instanceFieldname-symbol) expr)
            (set! (. Classname-symbol staticFieldname-symbol) expr)]}
        :url "vars#set"
        :doc "Used to set thread-local-bound vars, Java object instance
fields, and Java class static fields."
throw {:forms [(throw expr)]}
        :doc "The expr is evaluated and thrown, therefore it should
yield an instance of some derivee of Throwable."
try {:forms [(try expr* catch-clause* finally-clause?)])
        :doc "catch-clause => (catch classname name expr*)
finally-clause => (finally expr*)"

Catches and handles Java exceptions."
var {:forms [(var symbol)]}
        :doc "The symbol must resolve to a var, and the Var object
itself (not its value) is returned. The reader macro #'x expands
to (var x)."}}

(defn- special-doc [name-symbol]
  (assoc (or (special-doc-map name-symbol)
             (meta (resolve name-symbol)))
         :name name-symbol
         :special-form true))

(defn- namespace-doc [nspace]
  (assoc (meta nspace) :name (ns-name nspace)))

(defn- print-doc [m]
  (println "-----")
  (println (str (when-let [ns (:ns m)] (str (ns-name ns) "/"))
                (:name m)))
  (cond

```

```

(:forms m) (doseq [f (:forms m)]
  (print " ")
  (prn f))
(:arglists m) (prn (:arglists m)))
(if (:special-form m)
  (do
    (println "Special Form")
    (println " " (:doc m))
    (if (contains? m :url)
        (when (:url m)
          (println (str "\n Please see http://clojure.org/" (:url m))))
        (println (str "\n Please see http://clojure.org/special_forms#"
                     (:name m)))))

  (do
    (when (:macro m)
      (println "Macro"))
    (println " " (:doc m)))))

(defn find-doc
  "Prints documentation for any var whose documentation or name
  contains a match for re-string-or-pattern"
  {:added "1.0"}
  [re-string-or-pattern]
  (let [re (re-pattern re-string-or-pattern)
        ms (concat (mapcat
                    #(sort-by :name (map meta (vals (ns-interns %)))))
                    (all-ns))
              (map namespace-doc (all-ns))
              (map special-doc (keys special-doc-map)))]
    (doseq [m ms
            :when (and (:doc m)
                       (or (re-find (re-matcher re (:doc m)))
                           (re-find (re-matcher re (str (:name m))))))]
           (print-doc m)))))

(defmacro doc
  "Prints documentation for a var or special form given its name"
  {:added "1.0"}
  [name]
  (if-let [special-name ('{& fn catch try finally try} name)]
    (#'print-doc #'special-doc special-name)
    (cond
      (special-doc-map name) ('(#'print-doc #'special-doc 'name))
      (resolve name) ('(#'print-doc (meta (var name)))
      (find-ns name) ('(#'print-doc (namespace-doc (find-ns 'name))))))

;; -----
;; Examine Clojure functions (Vars, really)

(defn source-fn

```

"Returns a string of the source code for the given symbol, if it can find it. This requires that the symbol resolve to a Var defined in a namespace for which the .clj is in the classpath. Returns nil if it can't find the source. For most REPL usage, 'source' is more convenient.

```
Example: (source-fn 'filter)
[x]
(when-let [v (resolve x)]
  (when-let [filepath (:file (meta v))]
    (when-let [strm (.getResourceAsStream (RT/baseLoader) filepath)]
      (with-open [rdr (LineNumberReader. (InputStreamReader. strm))]
        (dotimes [_ (dec (:line (meta v)))] (.readLine rdr))
        (let [text (StringBuilder.)]
          (pbr (proxy [PushbackReader] [rdr]
            (read [] (let [i (proxy-super read)]
              (.append text (char i))
              i))))
          (read (PushbackReader. pbr))
          (str text))))))

(defmacro source
  "Prints the source code for the given symbol, if it can find it.
  This requires that the symbol resolve to a Var defined in a
  namespace for which the .clj is in the classpath.

  Example: (source filter)"
  [n]
  '(println (or (source-fn '^n) (str "Source not found"))))

(defn apropos
  "Given a regular expression or stringable thing, return a seq of
  all definitions in all currently-loaded namespaces that match the
  str-or-pattern."
  [str-or-pattern]
  (let [matches? (if (instance? java.util.regex.Pattern str-or-pattern)
                  #(re-find str-or-pattern (str %))
                  #(.contains (str %) (str str-or-pattern)))]
    (mapcat (fn [ns]
              (filter matches? (keys (ns-publics ns))))
            (all-ns)))

(defn dir-fn
  "Returns a sorted seq of symbols naming public vars in
  a namespace"
  [ns]
  (sort (map first (ns-publics (the-ns ns)))))

(defmacro dir
  "Prints a sorted directory of public vars in a namespace"
```

```

[nsname]
'(doseq [v# (dir-fn 'nsname)]
  (println v#))

(def ^:private demunge-map
  (into {"$" "/"}
    (map (fn [[k v]] [v k]) clojure.lang.Compiler/CHAR_MAP)))

(def ^:private demunge-pattern
  (re-pattern (apply str (interpose "|" (map #(str "\\Q" % "\\E")
    (keys demunge-map))))))

(defn- re-replace [re s f]
  (let [m (re-matcher re s)
    mseq (take-while identity
      (repeatedly #(when (re-find m)
        [(re-groups m) (.start m) (.end m)])))
    (apply str
      (concat
        (mapcat (fn [[_ _ start] [groups end]]
          (if end
            [(subs s start end) (f groups)]
            [(subs s start)])
          (cons [0 0 0] mseq)
          (concat mseq [nil]))))))
  (defn demunge
    "Given a string representation of a fn class,
    as in a stack trace element, returns a readable version."
    {:added "1.3"}
    [fn-name]
    (re-replace demunge-pattern fn-name demunge-map))

  (defn root-cause
    "Returns the initial cause of an exception or error by peeling off
    all of its wrappers"
    {:added "1.3"}
    [^Throwable t]
    (loop [cause t]
      (if (and (instance? clojure.lang.Compiler$CompilerException cause)
        (not= (.source
          ^clojure.lang.Compiler$CompilerException cause)
        "NO_SOURCE_FILE"))
        cause
        (if-let [cause (.getCause cause)]
          (recur cause)
          cause)))))

  (defn stack-element-str
    "Returns a (possibly unmunged) string representation of a
    "
  )

```

```

StackTraceElement"
{:added "1.3"}
[~StackTraceElement el]
(let [file (.getFileName el)
      closure-fn? (and file (or (.endsWith file ".clj")
                                  (= file "NO_SOURCE_FILE")))]
  (str (if closure-fn?
           (demunge (.getClassName el))
           (str (.getClassName el) "." (.getMethodName el)))
        " (" (.getFileName el) ":" (.getLineNumber el) ")"))

(defn pst
  "Prints a stack trace of the exception, to the depth requested. If
  none supplied, uses the root cause of the most recent repl
  exception (*e), and a depth of 12."
{:added "1.3"}
([] (pst 12))
([e-or-depth]
(if (instance? Throwable e-or-depth)
    (pst e-or-depth 12)
    (when-let [e *e]
      (pst (root-cause e) e-or-depth))))
([~Throwable e depth]
(binding [*out* *err*]
  (println (str (-> e class .getSimpleName) " " (.getMessage e)))
  (let [st (.getStackTrace e)
        cause (.getCause e)]
    (doseq [el (take depth
                      (remove
                        #(#{clojure.lang.RestFn" clojure.lang.AFn"}
                           (.getClassName %))
                        st))])
      (println (str \tab (stack-element-str el))))
    (when cause
      (println "Caused by:")
      (pst cause (min depth
                        (+ 2 (- (count (.getStackTrace cause))
                                  (count st))))))))
  ;; -----
  ;; Handle Ctrl-C keystrokes

(defn thread-stopper
  "Returns a function that takes one arg and uses that as an
  exception message to stop the given thread. Defaults to the
  current thread"
([] (thread-stopper (Thread/currentThread)))
([thread] (fn [msg] (.stop thread (Error. msg)))))

(defn set-break-handler!

```

```
"Register INT signal handler. After calling this, Ctrl-C will cause
the given function f to be called with a single argument, the signal.
Uses thread-stopper if no function given."
([] (set-break-handler! (thread-stopper)))
([f]
  (sun.misc.Signal/handle
    (sun.misc.Signal. "INT")
    (proxy [sun.misc.SignalHandler] []
      (handle [signal]
        (f (str "-- caught signal " signal))))))
```

11.27 set.clj

— set.clj —

```
\getchunk{Clojure Copyright}

(ns ^{:doc "Set operations such as union/intersection."
      :author "Rich Hickey"}
  clojure.set)

(defn- bubble-max-key [k coll]
  "Move a maximal element of coll according to fn k (which returns a
  number) to the front of coll."
  (let [max (apply max-key k coll)]
    (cons max (remove #(= max %) coll)))))

(defn union
  "Return a set that is the union of the input sets"
  {:added "1.0"}
  ([] #{})
  ([s1] s1)
  ([s1 s2]
     (if (< (count s1) (count s2))
         (reduce conj s2 s1)
         (reduce conj s1 s2)))
  ([s1 s2 & sets]
     (let [bubbled-sets (bubble-max-key count (conj sets s2 s1))]
       (reduce into (first bubbled-sets) (rest bubbled-sets)))))

(defn intersection
  "Return a set that is the intersection of the input sets"
  {:added "1.0"}
  ([s1] s1)
  ([s1 s2]
```

```

(if (< (count s2) (count s1))
    (recur s2 s1)
    (reduce (fn [result item]
              (if (contains? s2 item)
                  result
                  (disj result item)))
            s1 s1))
  ([s1 s2 & sets]
   (let [bubbled-sets
         (bubble-max-key #(- (count %)) (conj sets s2 s1))]
     (reduce intersection
            (first bubbled-sets) (rest bubbled-sets)))))

(defn difference
  "Return a set that is the first set without elements of the
  remaining sets"
  {:added "1.0"}
  ([s1] s1)
  ([s1 s2]
   (if (< (count s1) (count s2))
       (reduce (fn [result item]
                 (if (contains? s2 item)
                     (disj result item)
                     result))
               s1 s1)
       (reduce disj s1 s2)))
  ([s1 s2 & sets]
   (reduce difference s1 (conj sets s2)))))

(defn select
  "Returns a set of the elements for which pred is true"
  {:added "1.0"}
  [pred xset]
  (reduce (fn [s k] (if (pred k) s (disj s k)))
         xset xset))

(defn project
  "Returns a rel of the elements of xrel with only the keys in ks"
  {:added "1.0"}
  [xrel ks]
  (set (map #(select-keys % ks) xrel)))

(defn rename-keys
  "Returns the map with the keys in kmap renamed to the vals in kmap"
  {:added "1.0"}
  [map kmap]
  (reduce
   (fn [m [old new]]
     (if (and (not= old new)

```

```

        (contains? m old))
    (-> m (assoc new (get m old)) (dissoc old))
    m))
map kmap))

(defn rename
  "Returns a rel of the maps in xrel with the keys in kmap renamed
  to the vals in kmap"
  {:added "1.0"}
  [xrel kmap]
  (set (map #(rename-keys % kmap) xrel)))

(defn index
  "Returns a map of the distinct values of ks in the xrel mapped to a
  set of the maps in xrel with the corresponding values of ks."
  {:added "1.0"}
  [xrel ks]
  (reduce
    (fn [m x]
      (let [ik (select-keys x ks)]
        (assoc m ik (conj (get m ik #{}) x))))
    {} xrel))

(defn map-invert
  "Returns the map with the vals mapped to the keys."
  {:added "1.0"}
  [m] (reduce (fn [m [k v]] (assoc m v k)) {} m))

(defn join
  "When passed 2 rels, returns the rel corresponding to the natural
  join. When passed an additional keymap, joins on the corresponding
  keys."
  {:added "1.0"}
  ([xrel yrel] ;natural join
   (if (and (seq xrel) (seq yrel))
       (let [ks (intersection (set (keys (first xrel)))
                             (set (keys (first yrel)))))
             [r s] (if (<= (count xrel) (count yrel))
                       [xrel yrel]
                       [yrel xrel])
             idx (index r ks)]
         (reduce (fn [ret x]
                   (let [found (idx (select-keys x ks))]
                     (if found
                         (reduce #(conj %1 (merge %2 x)) ret found)
                         ret)))
                 #{} s))
       #{}))
  ([xrel yrel km] ;arbitrary key mapping
   (let [[r s k] (if (<= (count xrel) (count yrel))
                    (let [ks (intersection (set (keys (first xrel)))
                                          (set (keys (first yrel))))]
                      (if (= (count xrel) (count yrel))
                          (zipmap ks (map (fn [k] (assoc km k)) ks))
                          (zipmap ks (map (fn [k] (assoc km k)) ks)))
                    #{}))
  
```

```

[xrel yrel (map-invert km)]
[yrel xrel km])
(idx (index r (vals k)))
(reduce (fn [ret x]
(let [found
      (idx (rename-keys (select-keys x (keys k)) k))]
(if found
    (reduce #(conj %1 (merge %2 x)) ret found)
    ret)))
#{}) s)))

(defn subset?
  "Is set1 a subset of set2?"
  {:added "1.2",
   :tag Boolean}
[set1 set2]
(and (<= (count set1) (count set2))
     (every? #(contains? set2 %) set1)))

(defn superset?
  "Is set1 a superset of set2?"
  {:added "1.2",
   :tag Boolean}
[set1 set2]
(and (>= (count set1) (count set2))
     (every? #(contains? set1 %) set2)))

(comment
(refer 'set)
(def xs #{{:a 11 :b 1 :c 1 :d 4}
          {:a 2 :b 12 :c 2 :d 6}
          {:a 3 :b 3 :c 3 :d 8 :f 42}})

(def ys #{{:a 11 :b 11 :c 11 :e 5}
          {:a 12 :b 11 :c 12 :e 3}
          {:a 3 :b 3 :c 3 :e 7 }})

(join xs ys)
(join xs (rename ys {:b :yb :c :yc})) {:a :a})

(union #{:a :b :c} #{:c :d :e })
(difference #{:a :b :c} #{:c :d :e})
(intersection #{:a :b :c} #{:c :d :e})

(index ys [:b])
)

```

11.28 stacktrace.clj

— stacktrace.clj —

```
\getchunk{Clojure Copyright}

;;; stacktrace.clj: print Clojure-centric stack traces

;; by Stuart Sierra
;; January 6, 2009

(ns ^{:doc "Print stack traces oriented towards Clojure, not Java."
       :author "Stuart Sierra"}
  clojure.stacktrace)

(defn root-cause
  "Returns the last 'cause' Throwable in a chain of Throwables."
  {:added "1.1"}
  [tr]
  (if-let [cause (.getCause tr)]
    (recur cause)
    tr))

(defn print-trace-element
  "Prints a Clojure-oriented view of one element in a stack trace."
  {:added "1.1"}
  [e]
  (let [class (.getClassName e)
        method (.getMethodName e)]
    (let [match (re-matches #"[A-Za-z0-9_.-]+\$\(\w+\)\_\d+""
                           (str class))]
      (if (and match (= "invoke" method))
          (apply printf "%s/%s" (rest match))
          (printf "%s.%s" class method)))
    (printf " (%s:%d)" (or (.getFileName e) "") (.getLineNumber e)))))

(defn print-throwable
  "Prints the class and message of a Throwable."
  {:added "1.1"}
  [tr]
  (printf "%s: %s" (.getName (class tr)) (.getMessage tr)))

(defn print-stack-trace
  "Prints a Clojure-oriented stack trace of tr, a Throwable.
  Prints a maximum of n stack frames (default: unlimited).
  Does not print chained exceptions (causes)."
  {:added "1.1"}
  ([tr] (print-stack-trace tr nil)))
```

```

([tr n]
  (let [st (.getStackTrace tr)]
    (print-throwable tr)
    (newline)
    (print " at ")
    (print-trace-element (first st))
    (newline)
    (doseq [e (if (nil? n)
                   (rest st)
                   (take (dec n) (rest st)))]
      (print "    ")
      (print-trace-element e)
      (newline)))))

(defn print-cause-trace
  "Like print-stack-trace but prints chained exceptions (causes)."
  {:added "1.1"}
  ([tr] (print-cause-trace tr nil))
  ([tr n]
   (print-stack-trace tr n)
   (when-let [cause (.getCause tr)]
     (print "Caused by: " )
     (recur cause n)))))

(defn e
  "REPL utility. Prints a brief stack trace for the root cause of the
  most recent exception."
  {:added "1.1"}
  []
  (print-stack-trace (root-cause *e) 8))

—————

```

11.29 string.clj

— string.clj —

```

\getchunk{Clojure Copyright}

(ns ^{:doc "Clojure String utilities

It is poor form to (:use clojure.string). Instead, use require
with :as to specify a prefix, e.g.

(ns your.namespace.here
  (:require [clojure.string :as str]))}

```

Design notes for clojure.string:

1. Strings are objects (as opposed to sequences). As such, the string being manipulated is the first argument to a function; passing nil will result in a NullPointerException unless documented otherwise. If you want sequence-y behavior instead, use a sequence.
 2. Functions are generally not lazy, and call straight to host methods where those are available and efficient.
 3. Functions take advantage of String implementation details to write high-performing loop/recurs instead of using higher-order functions. (This is not idiomatic in general-purpose application code.)
 4. When a function is documented to accept a string argument, it will take any implementation of the correct *interface* on the host platform. In Java, this is CharSequence, which is more general than String. In ordinary usage you will almost always pass concrete strings. If you are doing something unusual, e.g. passing a mutable implementation of CharSequence, then thread-safety is your responsibility."
- ```
:author "Stuart Sierra, Stuart Halloway, David Liebke"
clojure.string
(:refer-clojure :exclude (replace reverse))
(:import (java.util.regex Pattern)
 clojure.lang.LazilyPersistentVector))
```

```
(defn ^String reverse
 "Returns s with its characters reversed."
 {:added "1.2"}
 [^CharSequence s]
 (.toString (.reverse (StringBuilder. s)))))

(defn- replace-by
 [^CharSequence s re f]
 (let [m (re-matcher re s)]
 (let [buffer (StringBuffer. (.length s))]
 (loop []
 (if (.find m)
 (do (.appendReplacement m buffer (f (re-groups m)))
 (recur))
 (do (.appendTail m buffer)
 (.toString buffer)))))))

(defn ^String replace
 "Replaces all instance of match with replacement in s.

 match/replacement can be:
```

```

string / string
char / char
pattern / (string or function of match).

See also replace-first."
{:added "1.2"}
[^CharSequence s match replacement]
(let [s (.toString s)]
 (cond
 (instance? Character match)
 (.replace s ^Character match ^Character replacement)
 (instance? CharSequence match)
 (.replace s ^CharSequence match ^CharSequence replacement)
 (instance? Pattern match)
 (if (instance? CharSequence replacement)
 (.replaceAll (re-matcher ^Pattern match s)
 (.toString ^CharSequence replacement)))
 (replace-by s match replacement))
 :else (throw (IllegalArgumentException.
 (str "Invalid match arg: " match))))))

(defn- replace-first-by
 [^CharSequence s ^Pattern re f]
 (let [m (re-matcher re s)]
 (let [buffer (StringBuffer. (.length s))]
 (if (.find m)
 (let [rep (f (re-groups m))]
 (.appendReplacement m buffer rep)
 (.appendTail m buffer)
 (str buffer)))))

(defn- replace-first-char
 [^CharSequence s ^Character match replace]
 (let [s (.toString s)
 i (.indexOf s (int match))]
 (if (= -1 i)
 s
 (str (subs s 0 i) replace (subs s (inc i))))))

(defn ^String replace-first
 "Replaces the first instance of match with replacement in s.

match/replacement can be:

char / char
string / string
pattern / (string or function of match).

See also replace-all."

```

```

{:added "1.2"}
[~CharSequence s match replacement]
(let [s (.toString s)]
 (cond
 (instance? Character match)
 (replace-first-char s match replacement)
 (instance? CharSequence match)
 (.replaceFirst s (Pattern/quote (.toString ^CharSequence match))
 (.toString ^CharSequence replacement)))
 (instance? Pattern match)
 (if (instance? CharSequence replacement)
 (.replaceFirst (re-matcher ^Pattern match s)
 (.toString ^CharSequence replacement)))
 (replace-first-by s match replacement)))
 :else (throw (IllegalArgumentException.
 (str "Invalid match arg: " match))))))

(defn ^String join
 "Returns a string of all elements in coll, as returned by (seq coll),
 separated by an optional separator."
{:added "1.2"}
([coll]
 (apply str coll))
([separator coll]
 (loop [sb (StringBuilder. (str (first coll)))
 more (next coll)
 sep (str separator)]
 (if more
 (recur (-> sb (.append sep) (.append (str (first more))))
 (next more)
 sep)
 (str sb))))
 (str sb)))))

(defn ^String capitalize
 "Converts first character of the string to upper-case, all other
 characters to lower-case."
{:added "1.2"}
[~CharSequence s]
(let [s (.toString s)]
 (if (< (count s) 2)
 (.toUpperCase s)
 (str (.toUpperCase (subs s 0 1))
 (.toLowerCase (subs s 1))))))

(defn ^String upper-case
 "Converts string to all upper-case."
{:added "1.2"}
[~CharSequence s]
(.. s toString toUpperCase))

```

```
(defn ^String lower-case
 "Converts string to all lower-case."
 {:added "1.2"}
 [^CharSequence s]
 (.. s toString toLowerCase))

(defn split
 "Splits string on a regular expression. Optional argument limit is
 the maximum number of splits. Not lazy. Returns vector of the splits."
 {:added "1.2"}
 ([^CharSequence s ^Pattern re]
 (LazilyPersistentVector/createOwning (.split re s)))
 ([^CharSequence s ^Pattern re limit]
 (LazilyPersistentVector/createOwning (.split re s limit)))))

(defn split-lines
 "Splits s on \\n or \\r\\n."
 {:added "1.2"}
 [^CharSequence s]
 (split s #"\r?\n"))

(defn ^String trim
 "Removes whitespace from both ends of string."
 {:added "1.2"}
 [^CharSequence s]
 (.. s toString trim))

(defn ^String triml
 "Removes whitespace from the left side of string."
 {:added "1.2"}
 [^CharSequence s]
 (loop [index (int 0)]
 (if (= (.length s) index)
 ""
 (if (Character/isWhitespace (.charAt s index))
 (recur (inc index))
 (.. s (subSequence index (.length s)) toString)))))

(defn ^String trimr
 "Removes whitespace from the right side of string."
 {:added "1.2"}
 [^CharSequence s]
 (loop [index (.length s)]
 (if (zero? index)
 ""
 (if (Character/isWhitespace (.charAt s (dec index)))
 (recur (dec index))
 (.. s (subSequence 0 index) toString)))))
```

```
(defn ^String trim-newline
 "Removes all trailing newline \\n or return \\r characters from
 string. Similar to Perl's chomp."
 {:added "1.2"}
 [^CharSequence s]
 (loop [index (.length s)]
 (if (zero? index)
 ""
 (let [ch (.charAt s (dec index))]
 (if (or (= ch \newline) (= ch \return))
 (recur (dec index))
 (.. s (subSequence 0 index) toString))))))

(defn blank?
 "True if s is nil, empty, or contains only whitespace."
 {:added "1.2"}
 [^CharSequence s]
 (if s
 (loop [index (int 0)]
 (if (= (.length s) index)
 true
 (if (Character/isWhitespace (.charAt s index))
 (recur (inc index))
 false)))
 true))

(defn ^String escape
 "Return a new string, using cmap to escape each character ch
 from s as follows:

 If (cmap ch) is nil, append ch to the new string.
 If (cmap ch) is non-nil, append (str (cmap ch)) instead."
 {:added "1.2"}
 [^CharSequence s cmap]
 (loop [index (int 0)
 buffer (StringBuilder. (.length s))]
 (if (= (.length s) index)
 (.toString buffer)
 (let [ch (.charAt s index)]
 (if-let [replacement (cmap ch)]
 (.append buffer replacement)
 (.append buffer ch))
 (recur (inc index) buffer)))))
```

---

## 11.30 template.clj

— template.clj —

```
\getchunk{Clojure Copyright}

;; template.clj - anonymous functions that pre-evaluate sub-expressions

;; By Stuart Sierra
;; June 23, 2009

;; CHANGE LOG
;;
;; June 23, 2009: complete rewrite, eliminated _1,_2,... argument
;; syntax
;;
;; January 20, 2009: added "template?" and checks for valid template
;; expressions.
;;
;; December 15, 2008: first version

(ns ^{:doc "Macros that expand to repeated copies of a
 template expression."
 :author "Stuart Sierra"}
 clojure.template
 (:require [clojure.walk :as walk]))

(defn apply-template
 "For use in macros. argv is an argument list, as in defn. expr is
 a quoted expression using the symbols in argv. values is a sequence
 of values to be used for the arguments.

 apply-template will recursively replace argument symbols in expr
 with their corresponding values, returning a modified expr.

 Example: (apply-template '[x] '(+ x x) '[2])
 ;;=> (+ 2 2)"
 [argv expr values]
 (assert (vector? argv))
 (assert (every? symbol? argv))
 (walk/prewalk-replace (zipmap argv values) expr))

(defmacro do-template
 "Repeatedly copies expr (in a do block) for each group of arguments
 in values. values are automatically partitioned by the number of
 arguments in argv, an argument vector as in defn.
```

```
Example: (macroexpand '(do-template [x y] (+ y x) 2 4 3 5))
 ;=> (do (+ 4 2) (+ 5 3))"
[argv expr & values]
(let [c (count argv)]
 '(do ~@ (map (fn [a] (apply-template argv expr a))
 (partition c values))))
```

—————

## 11.31 junit.clj

— junit.clj —

```
\getchunk{Clojure Copyright}

;; test/junit.clj: Extension to clojure.test for JUnit-compatible
;; XML output

;; by Jason Sankey
;; June 2009

;; DOCUMENTATION
;;

(ns ^{:doc "clojure.test extension for JUnit-compatible XML output.

JUnit (http://junit.org/) is the most popular unit-testing library
for Java. As such, tool support for JUnit output formats is
common. By producing compatible output from tests, this tool
support can be exploited."}

To use, wrap any calls to clojure.test/run-tests in the
with-junit-output macro, like this:
```

```
(use 'clojure.test)
(use 'clojure.test.junit)

(with-junit-output
 (run-tests 'my.cool.library))
```

```
To write the output to a file, rebind clojure.test/*test-out* to
your own PrintWriter (perhaps opened using
clojure.java.io/writer)."
:author "Jason Sankey"}
clojure.test.junit
(:require [clojure.stacktrace :as stack]
 [clojure.test :as t]))
```

```

;; copied from clojure.contrib.lazy-xml
(def ^{:private true}
 escape-xml-map
 (zipmap "'<>\"&" (map #(str \& % \;) '[apos lt gt quot amp])))
(defn- escape-xml [text]
 (apply str (map #(escape-xml-map % %) text)))

(def ^{:dynamic *var-context*})
(def ^{:dynamic *depth*})

(defn indent
 []
 (dotimes [n (* *depth* 4)] (print " "))

(defn start-element
 [tag pretty & [attrs]]
 (if pretty (indent))
 (print (str "<" tag))
 (if (seq attrs)
 (doseq [[key value] attrs]
 (print (str " " (name key) "=\""
 (escape-xml value) "\"")))
 (print ">"))
 (if pretty (println))
 (set! *depth* (inc *depth*)))

(defn element-content
 [content]
 (print (escape-xml content)))

(defn finish-element
 [tag pretty]
 (set! *depth* (dec *depth*))
 (if pretty (indent))
 (print (str "</>" tag)))
 (if pretty (println)))

(defn test-name
 [vars]
 (apply str (interpose "."
 (reverse (map #(:name (meta %)) vars)))))

(defn package-class
 [name]
 (let [i (.lastIndexOf name ".")]
 (if (< i 0)
 [nil name]
 [(.substring name 0 i) (.substring name (+ i 1))])))

(defn start-case

```

```

[name classname]
(start-element 'testcase true {:name name :classname classname}))

(defn finish-case
[]
(finish-element 'testcase true))

(defn suite-attrs
[package classname]
(let [attrs {:name classname}]
(if package
(assoc attrs :package package)
attrs)))

(defn start-suite
[name]
(let [[package classname] (package-class name)]
(start-element 'testsuite true (suite-attrs package classname)))))

(defn finish-suite
[]
(finish-element 'testsuite true))

(defn message-el
[tag message expected-str actual-str]
(indent)
(start-element tag false (if message {:message message} {}))
(element-content
(let [[file line] (t/file-position 5)
detail (apply str (interpose
"\n"
[(str "expected: " expected-str)
(str " actual: " actual-str)
(str " at: " file ":" line)]]])
(if message (str message "\n" detail) detail)))
(finish-element tag false)
(prn))

(defn failure-el
[message expected actual]
(message-el 'failure message (pr-str expected) (pr-str actual)))

(defn error-el
[message expected actual]
(message-el 'error
message
(pr-str expected)
(if (instance? Throwable actual)
(with-out-str
(stack/print-cause-trace actual
)))))
```

```

 t/*stack-trace-depth*)))
 (prn actual)))))

;; This multimethod will override test-is/report
(defmulti junit-report :type)

(defmethod junit-report :begin-test-ns [m]
 (t/with-test-out
 (start-suite (name (ns-name (:ns m))))))

(defmethod junit-report :end-test-ns []
 (t/with-test-out
 (finish-suite)))

(defmethod junit-report :begin-test-var [m]
 (t/with-test-out
 (let [var (:var m)]
 (binding [*var-context* (conj *var-context* var)]
 (start-case
 (test-name *var-context*)
 (name (ns-name (:ns (meta var)))))))))

(defmethod junit-report :end-test-var [m]
 (t/with-test-out
 (finish-case)))

(defmethod junit-report :pass [m]
 (t/with-test-out
 (t/inc-report-counter :pass)))

(defmethod junit-report :fail [m]
 (t/with-test-out
 (t/inc-report-counter :fail)
 (failure-el (:message m)
 (:expected m)
 (:actual m)))))

(defmethod junit-report :error [m]
 (t/with-test-out
 (t/inc-report-counter :error)
 (error-el (:message m)
 (:expected m)
 (:actual m)))))

(defmethod junit-report :default [])

(defmacro with-junit-output
 "Execute body with modified test-is reporting functions that write
JUnit-compatible XML output."
 {:added "1.1"}

```

```
[& body]
'(binding [t/report junit-report
 var-context (list)
 depth 1]
 (t/with-test-out
 (println "<?xml version=\"1.0\" encoding=\"UTF-8\"?>")
 (println "<testsuites>"))
 (let [result# `@body]
 (t/with-test-out (println "</testsuites>"))
 result#)))
```

---

## 11.32 tap.clj

— tap.clj —

```
\getchunk{Clojure Copyright}

;;; test_is/tap.clj: Extension to test for TAP output

;; by Stuart Sierra
;; March 31, 2009

;; Inspired by ClojureCheck by Meikel Brandmeyer:
;; http://kotka.de/projects/clojure/clojurecheck.html

;; DOCUMENTATION
;;

(ns ^{:doc "clojure.test extensions for the Test Anything Protocol (TAP)

TAP is a simple text-based syntax for reporting test results. TAP
was originally developed for Perl, and now has implementations in
several languages. For more information on TAP, see
http://testanything.org/ and
http://search.cpan.org/~petdance/TAP-1.0.0/TAP.pm

To use this library, wrap any calls to
clojure.test/run-tests in the with-tap-output macro,
like this:

(use 'clojure.test)
(use 'clojure.test.tap)"}))
```

```

(with-tap-output
 (run-tests 'my.cool.library))"
 :author "Stuart Sierra"}
clojure.test.tap
(:require [clojure.test :as t]
 [clojure.stacktrace :as stack])))

(defn print-tap-plan
 "Prints a TAP plan line like '1..n'. n is the number of tests"
 {:added "1.1"}
 [n]
 (println (str "1.." n)))

(defn print-tap-diagnostic
 "Prints a TAP diagnostic line. data is a (possibly multi-line)
string."
 {:added "1.1"}
 [data]
 (doseq [line (.split ^String data "\n")]
 (println "#" line)))

(defn print-tap-pass
 "Prints a TAP 'ok' line. msg is a string, with no line breaks"
 {:added "1.1"}
 [msg]
 (println "ok" msg))

(defn print-tap-fail
 "Prints a TAP 'not ok' line. msg is a string, with no line breaks"
 {:added "1.1"}
 [msg]
 (println "not ok" msg))

;; This multimethod will override test/report
(defmulti tap-report (fn [data] (:type data)))

(defmethod tap-report :default [data]
 (t/with-test-out
 (print-tap-diagnostic (pr-str data)))))

(defmethod tap-report :pass [data]
 (t/with-test-out
 (t/inc-report-counter :pass)
 (print-tap-pass (t/testing-vars-str))
 (when (seq t/*testing-contexts*)
 (print-tap-diagnostic (t/testing-contexts-str)))
 (when (:message data)
 (print-tap-diagnostic (:message data))))
 (print-tap-diagnostic (str "expected:" (pr-str (:expected data))))))

```

```
(print-tap-diagnostic (str " actual:" (pr-str (:actual data)))))

(defmethod tap-report :error [data]
 (t/with-test-out
 (t/inc-report-counter :error)
 (print-tap-fail (t/testing-vars-str))
 (when (seq t/*testing-contexts*)
 (print-tap-diagnostic (t/testing-contexts-str)))
 (when (:message data)
 (print-tap-diagnostic (:message data)))
 (print-tap-diagnostic "expected:" (pr-str (:expected data)))
 (print-tap-diagnostic " actual: ")
 (print-tap-diagnostic
 (with-out-str
 (if (instance? Throwable (:actual data))
 (stack/print-cause-trace (:actual data) t/*stack-trace-depth*)
 (prn (:actual data))))))

(defmethod tap-report :summary [data]
 (t/with-test-out
 (print-tap-plan (+ (:pass data) (:fail data) (:error data)))))

(defmacro with-tap-output
 "Execute body with modified test reporting functions that produce
 TAP output"
 {:added "1.1"}
 [& body]
 `(~(binding [t/report tap-report]
 ~@body))
```

—————

## 11.33 test.clj

— test.clj —

```
\getchunk{Clojure Copyright}

;;; test.clj: test framework for Clojure

;; by Stuart Sierra
;; March 28, 2009

;; Thanks to Chas Emerick, Allen Rohner, and Stuart Halloway for
;; contributions and suggestions.
```

```
(ns
 ^{:author "Stuart Sierra, with contributions and suggestions by
 Chas Emerick, Allen Rohner, and Stuart Halloway",
 :doc "A unit testing framework."}
```

#### ASSERTIONS

The core of the library is the `\"is\"` macro, which lets you make assertions of any arbitrary expression:

```
(is (= 4 (+ 2 2)))
(is (instance? Integer 256))
(is (.startsWith \"abcde\" \"ab\"))
```

You can type an `\"is\"` expression directly at the REPL, which will print a message if it fails.

```
user> (is (= 5 (+ 2 2)))

FAIL in (:1)
expected: (= 5 (+ 2 2))
actual: (not (= 5 4))
false
```

The `\"expected:\"` line shows you the original expression, and the `\"actual:\"` shows you what actually happened. In this case, it shows that `(+ 2 2)` returned 4, which is not = to 5. Finally, the `\"false\"` on the last line is the value returned from the expression. The `\"is\"` macro always returns the result of the inner expression.

There are two special assertions for testing exceptions. The `\"(is (thrown? c ...))\"` form tests if an exception of class `c` is thrown:

```
(is (thrown? ArithmeticException (/ 1 0)))

\"(is (thrown-with-msg? c re ...))\" does the same thing and also
tests that the message on the exception matches the regular
expression re:

(is (thrown-with-msg? ArithmeticException #\"Divide by zero\""
(/ 1 0)))
```

#### DOCUMENTING TESTS

`\"is\"` takes an optional second argument, a string describing the assertion. This message will be included in the error report.

```
(is (= 5 (+ 2 2)) \"Crazy arithmetic\")
```

In addition, you can document groups of assertions with the `\"testing\"` macro, which takes a string followed by any number of assertions. The string will be included in failure reports. Calls to `\"testing\"` may be nested, and all of the strings will be joined together with spaces in the final report, in a style similar to RSpec <<http://rspec.info/>>

```
(testing \"Arithmetic\"
 (testing \"with positive integers\"
 (is (= 4 (+ 2 2)))
 (is (= 7 (+ 3 4))))
 (testing \"with negative integers\"
 (is (= -4 (+ -2 -2)))
 (is (= -1 (+ 3 -4)))))
```

Note that, unlike RSpec, the `\"testing\"` macro may only be used INSIDE a `\"deftest\"` or `\"with-test\"` form (see below).

#### DEFINING TESTS

There are two ways to define tests. The `\"with-test\"` macro takes a defn or def form as its first argument, followed by any number of assertions. The tests will be stored as metadata on the definition.

```
(with-test
 (defn my-function [x y]
 (+ x y))
 (is (= 4 (my-function 2 2)))
 (is (= 7 (my-function 3 4))))
```

As of Clojure SVN rev. 1221, this does not work with defmacro. See <http://code.google.com/p/clojure/issues/detail?id=51>

The other way lets you define tests separately from the rest of your code, even in a different namespace:

```
(deftest addition
 (is (= 4 (+ 2 2)))
 (is (= 7 (+ 3 4)))

(deftest subtraction
 (is (= 1 (- 4 3)))
 (is (= 3 (- 7 4))))
```

This creates functions named `\"addition\"` and `\"subtraction\"`, which can be called like any other function. Therefore, tests can be grouped and composed, in a style similar to the test framework in

```
Peter Seibel's \"Practical Common Lisp\"

<http://www.gigamonkeys.com/book/practical-building-a-unit-test-framework.html>

(deftest arithmetic
 (addition)
 (subtraction))
```

The names of the nested tests will be joined in a list, like `\"(arithmetic addition)\"`, in failure reports. You can use nested tests to set up a context shared by several tests.

#### RUNNING TESTS

Run tests with the function `\"(run-tests namespaces...)\"`:

```
(run-tests 'your.namespace 'some.other.namespace)
```

If you don't specify any namespaces, the current namespace is used. To run all tests in all namespaces, use `\"(run-all-tests)\"`.

By default, these functions will search for all tests defined in a namespace and run them in an undefined order. However, if you are composing tests, as in the `\"arithmetic\"` example above, you probably do not want the `\"addition\"` and `\"subtraction\"` tests run separately. In that case, you must define a special function named `\"test-ns-hook\"` that runs your tests in the correct order:

```
(defn test-ns-hook []
 (arithmetic))
```

Note: `test-ns-hook` prevents execution of fixtures (see below).

#### OMITTING TESTS FROM PRODUCTION CODE

You can bind the variable `\"*load-tests*\"` to false when loading or compiling code in production. This will prevent any tests from being created by `\"with-test\"` or `\"deftest\"`.

#### FIXTURES

Fixtures allow you to run code before and after tests, to set up the context in which tests should be run.

A fixture is just a function that calls another function passed as an argument. It looks like this:

```
(defn my-fixture [f]
 Perform setup, establish bindings, whatever.
 (f) Then call the function we were passed.
 Tear-down / clean-up code here.
)
```

Fixtures are attached to namespaces in one of two ways. `\"each\"` fixtures are run repeatedly, once for each test function created with `\"deftest\"` or `\"with-test\"`. `\"each\"` fixtures are useful for establishing a consistent before/after state for each test, like clearing out database tables.

`\"each\"` fixtures can be attached to the current namespace like this:  
`(use-fixtures :each fixture1 fixture2 ...)`  
 The fixture1, fixture2 are just functions like the example above.  
 They can also be anonymous functions, like this:  
`(use-fixtures :each (fn [f] setup... (f) cleanup...))`

The other kind of fixture, a `\"once\"` fixture, is only run once, around ALL the tests in the namespace. `\"once\"` fixtures are useful for tasks that only need to be performed once, like establishing database connections, or for time-consuming tasks.

Attach `\"once\"` fixtures to the current namespace like this:  
`(use-fixtures :once fixture1 fixture2 ...)`

Note: Fixtures and test-ns-hook are mutually incompatible. If you are using test-ns-hook, fixture functions will **\*never\*** be run.

#### SAVING TEST OUTPUT TO A FILE

All the test reporting functions write to the var `*test-out*`. By default, this is the same as `*out*`, but you can rebind it to any PrintWriter. For example, it could be a file opened with `clojure.java.io/writer`.

#### EXTENDING TEST-IS (ADVANCED)

You can extend the behavior of the `\"is\"` macro by defining new methods for the `\"assert-expr\"` multimethod. These methods are called during expansion of the `\"is\"` macro, so they should return quoted forms to be evaluated.

You can plug in your own test-reporting framework by rebinding the `\"report\"` function: (report event)

The 'event' argument is a map. It will always have a `:type` key, whose value will be a keyword signaling the type of event being

reported. Standard events with :type value of :pass, :fail, and :error are called when an assertion passes, fails, and throws an exception, respectively. In that case, the event will also have the following keys:

```
:expected The form that was expected to be true
:actual A form representing what actually occurred
:message The string message given as an argument to 'is'
```

The \"testing\" strings will be a list in \"\*testing-contexts\*\", and the vars being tested will be a list in \"\*testing-vars\*\".

Your \"report\" function should wrap any printing calls in the \"with-test-out\" macro, which rebinds \*out\* to the current value of \*test-out\*.

```
For additional event types, see the examples in the code.
"}
clojure.test
(:require [clojure.template :as temp]
 [clojure.stacktrace :as stack]))

;; Nothing is marked "private" here, so you can rebind things to plug
;; in your own testing or reporting frameworks.

;;; USER-MODIFIABLE GLOBALS

(defonce ^:dynamic
 `{:doc "True by default. If set to false, no test functions will
 be created by deftest, set-test, or with-test. Use this to omit
 tests when compiling or loading production code."
 :added "1.1"}
 load-tests true)

(def ^:dynamic
 `{:doc "The maximum depth of stack traces to print when an Exception
 is thrown during a test. Defaults to nil, which means print the
 complete stack trace."
 :added "1.1"}
 stack-trace-depth nil)

;;; GLOBALS USED BY THE REPORTING FUNCTIONS

; bound to a ref of a map in test-ns
(def ^:dynamic *report-counters* nil)

; used to initialize *report-counters*
(def ^:dynamic *initial-report-counters*
```

```

{:test 0, :pass 0, :fail 0, :error 0})

; bound to hierarchy of vars being tested
(def ^:dynamic *testing-vars* (list))

; bound to hierarchy of "testing" strings
(def ^:dynamic *testing-contexts* (list))

; PrintWriter for test reporting output
(def ^:dynamic *test-out* *out*)

(defmacro with-test-out
 "Runs body with *out* bound to the value of *test-out*."
 {:added "1.1"}
 [& body]
 '(binding [*out* *test-out*]
 ~@body))

;;; UTILITIES FOR REPORTING FUNCTIONS

(defn file-position
 "Returns a vector [filename line-number] for the nth call up the
 stack.

Deprecated in 1.2: The information needed for test reporting is
now on :file and :line keys in the result map."
 {:added "1.1"
 :deprecated "1.2"}
 [n]
 (let [^StackTraceElement s
 (nth (.getStackTrace (new java.lang.Throwable)) n)]
 [(.getFileName s) (.getLineNumber s)]))

(defn testing-vars-str
 "Returns a string representation of the current test. Renders names
 in *testing-vars* as a list, then the source file and line of
 current assertion."
 {:added "1.1"}
 [m]
 (let [{:keys [file line]} m]
 (str
 ;; Uncomment to include namespace in failure report:
 ;;(ns-name (:ns (meta (first *testing-vars*)))) "/ "
 (reverse (map #(:name (meta %)) *testing-vars*))
 " (" file ":" line ")"))

(defn testing-contexts-str
 "Returns a string representation of the current test context. Joins
 strings in *testing-contexts* with spaces."
 {:added "1.1"})

```

```

[]

(apply str (interpose " " (reverse *testing-contexts*)))

(defn inc-report-counter
 "Increments the named counter in *report-counters*, a ref to a map.
 Does nothing if *report-counters* is nil."
 {:added "1.1"}
 [name]
 (when *report-counters*
 (dosync (commute *report-counters* assoc name
 (inc (or (*report-counters* name) 0))))))

;; TEST RESULT REPORTING

(defmulti
 {:doc "Generic reporting function, may be overridden to plug in
 different report formats (e.g., TAP, JUnit). Assertions such as
 'is' call 'report' to indicate results. The argument given to
 'report' will be a map with a :type key. See the documentation at
 the top of test-is.clj for more information on the types of
 arguments for 'report'."}
 :dynamic true
 {:added "1.1"}
 report :type)

(defn- file-and-line
 [exception depth]
 (let [^StackTraceElement s (nth (.getStackTrace exception) depth)]
 {:file (.getFileName s) :line (.getLineNumber s)}))

(defn do-report
 "Add file and line information to a test result and call report.
 If you are writing a custom assert-expr method, call this function
 to pass test results to report."
 {:added "1.2"}
 [m]
 (report
 (case
 (:type m)
 :fail (merge (file-and-line (new java.lang.Throwable) 1) m)
 :error (merge (file-and-line (:actual m) 0) m)
 m)))

(defmethod report :default [m]
 (with-test-out (prn m)))

(defmethod report :pass [m]
 (with-test-out (inc-report-counter :pass)))

(defmethod report :fail [m]

```

```

(with-test-out
 (inc-report-counter :fail)
 (println "\nFAIL in" (testing-vars-str m))
 (when (seq *testing-contexts*) (println (testing-contexts-str)))
 (when-let [message (:message m)] (println message))
 (println "expected:" (pr-str (:expected m)))
 (println " actual:" (pr-str (:actual m)))))

(defmethod report :error [m]
 (with-test-out
 (inc-report-counter :error)
 (println "\nERROR in" (testing-vars-str m))
 (when (seq *testing-contexts*) (println (testing-contexts-str)))
 (when-let [message (:message m)] (println message))
 (println "expected:" (pr-str (:expected m)))
 (print " actual: ")
 (let [actual (:actual m)]
 (if (instance? Throwable actual)
 (stack/print-cause-trace actual *stack-trace-depth*)
 (prn actual)))))

(defmethod report :summary [m]
 (with-test-out
 (println "\nRan" (:test m) "tests containing"
 (+ (:pass m) (:fail m) (:error m)) "assertions.")
 (println (:fail m) "failures," (:error m) "errors.")))

(defmethod report :begin-test-ns [m]
 (with-test-out
 (println "\nTesting" (ns-name (:ns m)))))

;; Ignore these message types:
(defmethod report :end-test-ns [m])
(defmethod report :begin-test-var [m])
(defmethod report :end-test-var [m])

;; UTILITIES FOR ASSERTIONS

(defn get-possibly-unbound-var
 "Like var-get but returns nil if the var is unbound."
 {:added "1.1"}
 [v]
 (try (var-get v)
 (catch IllegalStateException e
 nil)))

(defn function?
 "Returns true if argument is a function or a symbol that resolves to"

```

```

a function (not a macro)."
{:added "1.1"}
[x]
(if (symbol? x)
(when-let [v (resolve x)]
(when-let [value (get-possibly-unbound-var v)]
(and (fn? value)
(not (:macro (meta v))))))
(fn? x)))

(defn assert-predicate
"Returns generic assertion code for any functional predicate. The
'expected' argument to 'report' will contain the original form, the
'actual' argument will contain the form with all its sub-forms
evaluated. If the predicate returns false, the 'actual' form will
be wrapped in (not...)."
{:added "1.1"}
[msg form]
(let [args (rest form)
 pred (first form)]
'(let [values# (list `@args)
 result# (apply `pred values#)]
 (if result#
 (do-report {:type :pass, :message `msg,
 :expected `form, :actual (cons `pred values#)})
 (do-report {:type :fail, :message `msg,
 :expected `form,
 :actual (list `'(not (cons `pred values#)))})
 result#)))

(defn assert-any
"Returns generic assertion code for any test, including macros, Java
method calls, or isolated symbols."
{:added "1.1"}
[msg form]
'(let [value# `form]
 (if value#
 (do-report {:type :pass, :message `msg,
 :expected `form, :actual value#})
 (do-report {:type :fail, :message `msg,
 :expected `form, :actual value#}))
 value#))

;; ASSERTION METHODS

;; You don't call these, but you can add methods to extend the 'is'
;; macro. These define different kinds of tests, based on the first
;; symbol in the test expression.

```

```
(defmulti assert Expr
 (fn [msg form]
 (cond
 (nil? form) :always-fail
 (seq? form) (first form)
 :else :default)))

(defmethod assert Expr :always-fail [msg form]
 ;; nil test: always fail
 '(do-report {:type :fail, :message ~msg}))

(defmethod assert Expr :default [msg form]
 (if (and (sequential? form) (function? (first form)))
 (assert-predicate msg form)
 (assert-any msg form)))

(defmethod assert Expr 'instance? [msg form]
 ;; Test if x is an instance of y.
 '(let [klass# ~(nth form 1)
 object# ~(nth form 2)]
 (let [result# (instance? klass# object#)]
 (if result#
 (do-report {:type :pass, :message ~msg,
 :expected '~form, :actual (class object#)})
 (do-report {:type :fail, :message ~msg,
 :expected '~form, :actual (class object#)}))
 result#)))

(defmethod assert Expr 'thrown? [msg form]
 ;; (is (thrown? c expr))
 ;; Asserts that evaluating expr throws an exception of class c.
 ;; Returns the exception thrown.
 '(let [klass (second form)
 body (nthnext form 2)]
 '(try ~@body
 (do-report {:type :fail, :message ~msg,
 :expected '~form, :actual nil})
 (catch ~klass e#
 (do-report {:type :pass, :message ~msg,
 :expected '~form, :actual e#})
 e#)))))

(defmethod assert Expr 'thrown-with-msg? [msg form]
 ;; (is (thrown-with-msg? c re expr))
 ;; Asserts that evaluating expr throws an exception of class c.
 ;; Also asserts that the message string of the exception matches
 ;; (with re-find) the regular expression re.
 '(let [klass (nth form 1)
 re (nth form 2)
```

```

 body (nthnext form 3)]
 '(try ~@body
 (do-report {:type :fail, :message ~msg, :expected '~form,
 :actual nil})
 (catch ~klass e#
 (let [m# (.getMessage e#)]
 (if (re-find ~re m#)
 (do-report {:type :pass, :message ~msg,
 :expected '~form, :actual e#})
 (do-report {:type :fail, :message ~msg,
 :expected '~form, :actual e#})))
 e#)))

(defmacro try-expr
 "Used by the 'is' macro to catch unexpected exceptions.
 You don't call this."
 {:added "1.1"}
 [msg form]
 '(try ~(assert-expr msg form)
 (catch Throwable t#
 (do-report {:type :error, :message ~msg,
 :expected '~form, :actual t#})))))

;;; ASSERTION MACROS

;; You use these in your tests.

(defmacro is
 "Generic assertion macro. 'form' is any predicate test.
 'msg' is an optional message to attach to the assertion.

 Example: (is (= 4 (+ 2 2)) \"Two plus two should be 4\")"

Special forms:

(is (thrown? c body)) checks that an instance of c is thrown from
body, fails if not; then returns the thing thrown.

(is (thrown-with-msg? c re body)) checks that an instance of c is
thrown AND that the message on the exception matches (with
re-find) the regular expression re."
{:added "1.1"}
([form] '(is ~form nil))
([form msg] '(try-expr ~msg ~form)))

(defmacro are
 "Checks multiple assertions with a template expression.
```

See clojure.template/do-template for an explanation of templates.

Example: (are [x y] (= x y)  
           2 (+ 1 1)  
           4 (\* 2 2))  
 Expands to:  
 (do (is (= 2 (+ 1 1)))  
     (is (= 4 (\* 2 2))))

Note: This breaks some reporting features, such as line numbers."

```
{:added "1.1"}
[argv expr & args]
'(temp/do-template ~argv (is ~expr) ~@args))

(defmacro testing
 "Adds a new string to the list of testing contexts. May be nested,
 but must occur inside a test function (deftest)."
 {:added "1.1"}
 [string & body]
 '(binding [*testing-contexts* (conj *testing-contexts* ~string)]
 ~@body))
```

;;; DEFINING TESTS

```
(defmacro with-test
 "Takes any definition form (that returns a Var) as the first argument.
 Remaining body goes in the :test metadata function for that Var.

 When *load-tests* is false, only evaluates the definition, ignoring
 the tests."
 {:added "1.1"}
 [definition & body]
 (if *load-tests*
 '(doto ~definition (alter-meta! assoc :test (fn [] ~@body)))
 definition))
```

```
(defmacro deftest
 "Defines a test function with no arguments. Test functions may call
 other tests, so tests may be composed. If you compose tests, you
 should also define a function named test-ns-hook; run-tests will
 call test-ns-hook instead of testing all vars.
```

Note: Actually, the test body goes in the :test metadata on the var, and the real function (the value of the var) calls test-var on itself.

```

When *load-tests* is false, deftest is ignored."
{:added "1.1"}
[name & body]
(when *load-tests*
 '(def ~(vary-meta name assoc :test `(fn [] ~@body))
 (fn [] (test-var (var ~name)))))

(defmacro deftest-
 "Like deftest but creates a private var."
{:added "1.1"}
[name & body]
(when *load-tests*
 '(def ~(vary-meta name assoc :test `(fn [] ~@body) :private true)
 (fn [] (test-var (var ~name)))))

(defmacro set-test
 "Experimental.
Sets :test metadata of the named var to a fn with the given body.
The var must already exist. Does not modify the value of the var.

When *load-tests* is false, set-test is ignored."
{:added "1.1"}
[name & body]
(when *load-tests*
 '(alter-meta! (var ~name) assoc :test (fn [] ~@body)))

;; DEFINING FIXTURES

(defn- add-ns-meta
 "Adds elements in coll to the current namespace metadata as the
value of key."
{:added "1.1"}
[key coll]
(alter-meta! *ns* assoc key coll))

(defmulti use-fixtures
 "Wrap test runs in a fixture function to perform setup and
teardown. Using a fixture-type of :each wraps every test
individually, while:once wraps the whole run in a single function."
{:added "1.1"}
(fn [fixture-type & args] fixture-type))

(defmethod use-fixtures :each [fixture-type & args]
 (add-ns-meta ::each-fixtures args))

(defmethod use-fixtures :once [fixture-type & args]
 (add-ns-meta ::once-fixtures args))

```

```
(defn- default-fixture
 "The default, empty, fixture function. Just calls its argument."
 {:added "1.1"}
 [f]
 (f))

(defn compose-fixtures
 "Composes two fixture functions, creating a new fixture function
 that combines their behavior."
 {:added "1.1"}
 [f1 f2]
 (fn [g] (f1 (fn [] (f2 g)))))

(defn join-fixtures
 "Composes a collection of fixtures, in order. Always returns a valid
 fixture function, even if the collection is empty."
 {:added "1.1"}
 [fixtures]
 (reduce compose-fixtures default-fixture fixtures))

;; RUNNING TESTS: LOW-LEVEL FUNCTIONS

(defn test-var
 "If v has a function in its :test metadata, calls that function,
 with *testing-vars* bound to (conj *testing-vars* v)."
 {:dynamic true, :added "1.1"}
 [v]
 (when-let [t (:test (meta v))]
 (binding [*testing-vars* (conj *testing-vars* v)]
 (do-report {:type :begin-test-var, :var v})
 (inc-report-counter :test)
 (try (t)
 (catch Throwable e
 (do-report {:type :error,
 :message "Uncaught exception, not in assertion."
 :expected nil, :actual e})))
 (do-report {:type :end-test-var, :var v})))))

(defn test-all-vars
 "Calls test-var on every var interned in the namespace, with fixtures."
 {:added "1.1"}
 [ns]
 (let [once-fixture-fn (join-fixtures (::_once-fixtures (meta ns)))
 each-fixture-fn (join-fixtures (::_each-fixtures (meta ns)))]
 (once-fixture-fn
 (fn []
```

```

(doseq [v (vals (ns-interns ns))]
 (when (:test (meta v))
 (each-fixture-fn (fn [] (test-var v)))))))

(defn test-ns
 "If the namespace defines a function named test-ns-hook, calls that.
 Otherwise, calls test-all-vars on the namespace. 'ns' is a
 namespace object or a symbol.

 Internally binds *report-counters* to a ref initialized to
 initial-report-counters. Returns the final, dereferenced state of
 report-counters."
 {:added "1.1"}
 [ns]
 (binding [*report-counters* (ref *initial-report-counters*)]
 (let [ns-obj (the-ns ns)]
 (do-report {:type :begin-test-ns, :ns ns-obj})
 ;; If the namespace has a test-ns-hook function, call that:
 (if-let [v (find-var
 (symbol (str (ns-name ns-obj) "test-ns-hook")))]
 ((var-get v))
 ;; Otherwise, just test every var in the namespace.
 (test-all-vars ns-obj)
 (do-report {:type :end-test-ns, :ns ns-obj}))
 @*report-counters*))

;; RUNNING TESTS: HIGH-LEVEL FUNCTIONS

(defn run-tests
 "Runs all tests in the given namespaces; prints results.
 Defaults to current namespace if none given. Returns a map
 summarizing test results."
 {:added "1.1"}
 ([] (run-tests *ns*))
 ([& namespaces]
 (let [summary (assoc (apply merge-with + (map test-ns namespaces))
 :type :summary)]
 (do-report summary)
 summary)))
 (defn run-all-tests
 "Runs all tests in all namespaces; prints results.
 Optional argument is a regular expression; only namespaces with
 names matching the regular expression (with re-matches) will be
 tested."
 {:added "1.1"}
 ([] (apply run-tests (all-ns)))
 ([re] (apply run-tests

```

```
(filter #(re-matches re (name (ns-name %))) (all-ns)))))

(defn successful?
 "Returns true if the given test summary indicates all tests
 were successful, false otherwise."
 {:added "1.1"}
 [summary]
 (and (zero? (:fail summary 0))
 (zero? (:error summary 0))))
```

—————

## 11.34 version.properties

— version.properties —

```
clojure.version.major=1
clojure.version.minor=3
clojure.version.incremental=0
clojure.version.qualifier=master
clojure.version.interim=interim
```

—————

## 11.35 walk.clj

— walk.clj —

```
\getchunk{Clojure Copyright}

;;; walk.clj - generic tree walker with replacement

;; by Stuart Sierra
;; December 15, 2008

;; CHANGE LOG:
;;
;; * December 15, 2008: replaced 'walk' with 'prewalk' & 'postwalk'
;;
;; * December 9, 2008: first version

(ns
 ^{:author "Stuart Sierra",
```

```

:doc "This file defines a generic tree walker for Clojure data
structures. It takes any data structure (list, vector, map, set,
seq), calls a function on every element, and uses the return value
of the function in place of the original. This makes it fairly
easy to write recursive search-and-replace functions, as shown in
the examples.

Note: \"walk\" supports all Clojure data structures EXCEPT maps
created with sorted-map-by. There is no (obvious) way to retrieve
the sorting function."}
clojure.walk)

(defn walk
 "Traverses form, an arbitrary data structure. inner and outer are
functions. Applies inner to each element of form, building up a
data structure of the same type, then applies outer to the result.
Recognizes all Clojure data structures except sorted-map-by.
Consumes seqs as with doall."
{:added "1.1"}
[inner outer form]
(cond
 (list? form) (outer (apply list (map inner form)))
 (seq? form) (outer (doall (map inner form)))
 (vector? form) (outer (vec (map inner form)))
 (map? form) (outer (into (if (sorted? form) (sorted-map) {})
 (map inner form)))
 (set? form) (outer (into (if (sorted? form) (sorted-set) #{})
 (map inner form)))
 :else (outer form)))

(defn postwalk
 "Performs a depth-first, post-order traversal of form. Calls f on
each sub-form, uses f's return value in place of the original.
Recognizes all Clojure data structures except sorted-map-by.
Consumes seqs as with doall."
{:added "1.1"}
[f form]
(walk (partial postwalk f) f form))

(defn prewalk
 "Like postwalk, but does pre-order traversal."
{:added "1.1"}
[f form]
(walk (partial prewalk f) identity (f form)))

;; Note: I wanted to write:
;;
;; (defn walk
;; [f form]
```

```

;; (let [pf (partial walk f)]
;; (if (coll? form)
;; (f (into (empty form) (map pf form)))
;; (f form)))
;;
;; but this throws a ClassCastException when applied to a map.

(defn postwalk-demo
 "Demonstrates the behavior of postwalk by printing each form as it is
walked. Returns form."
{:added "1.1"}
[form]
(postwalk (fn [x] (print "Walked: ") (prn x) x) form))

(defn prewalk-demo
 "Demonstrates the behavior of prewalk by printing each form as it is
walked. Returns form."
{:added "1.1"}
[form]
(prewalk (fn [x] (print "Walked: ") (prn x) x) form))

(defn keywordize-keys
 "Recursively transforms all map keys from strings to keywords."
{:added "1.1"}
[m]
(let [f (fn [[k v]] (if (string? k) [(keyword k) v] [k v]))]
 ;; only apply to maps
 (postwalk (fn [x] (if (map? x) (into {} (map f x)) x)) m)))

(defn stringify-keys
 "Recursively transforms all map keys from keywords to strings."
{:added "1.1"}
[m]
(let [f (fn [[k v]] (if (keyword? k) [(name k) v] [k v]))]
 ;; only apply to maps
 (postwalk (fn [x] (if (map? x) (into {} (map f x)) x)) m)))

(defn prewalk-replace
 "Recursively transforms form by replacing keys in smap with their
values. Like clojure/replace but works on any data structure. Does
replacement at the root of the tree first."
{:added "1.1"}
[smap form]
(prewalk (fn [x] (if (contains? smap x) (smap x) x)) form))

(defn postwalk-replace
 "Recursively transforms form by replacing keys in smap with their
values. Like clojure/replace but works on any data structure. Does
replacement at the leaves of the tree first."

```

```

{:added "1.1"}
[smap form]
(postwalk (fn [x] (if (contains? smap x) (smap x) x)) form))

(defn macroexpand-all
 "Recursively performs all possible macroexpansions in form."
 {:added "1.1"}
 [form]
 (prewalk (fn [x] (if (seq? x) (macroexpand x) x)) form))

```

---

## 11.36 xml.clj

— xml.clj —

```

\getchunk{Clojure Copyright}

(ns ^{:doc "XML reading/writing."
 :author "Rich Hickey"}
 clojure.xml
 (:import (org.xml.sax ContentHandler Attributes SAXException)
 (javax.xml.parsers SAXParser SAXParserFactory)))

(def ^:dynamic *stack*)
(def ^:dynamic *current*)
(def ^:dynamic *state*) ; :element :chars :between
(def ^:dynamic *sb*)

(defstruct element :tag :attrs :content)

(def tag (accessor element :tag))
(def attrs (accessor element :attrs))
(def content (accessor element :content))

(def content-handler
 (let [push-content
 (fn [e c] (assoc e :content (conj (or (:content e) []) c)))
 push-chars
 (fn []
 (when (and (= *state* :chars)
 (some (complement
 #(Character/isWhitespace (char %)))
 (str *sb*)))
 (set! *current*
 (push-content *current* (str *sb*))))))
]
```

```

(new clojure.lang.XMLHandler
 (proxy [ContentHandler] []
 (startElement [uri local-name q-name ^Attributes atts]
 (let [attrs (fn [ret i]
 (if (neg? i)
 ret
 (recur
 (assoc ret
 (clojure.lang.Keyword/intern
 (symbol (.getQName atts i)))
 (.getValue atts (int i)))
 (dec i))))
 e (struct element
 (. clojure.lang.Keyword
 (intern (symbol q-name)))
 (when (pos? (.getLength atts))
 (attrs {} (dec (.getLength atts))))))
 (push-chars)
 (set! *stack* (conj *stack* *current*))
 (set! *current* e)
 (set! *state* :element))
 nil)
 (endElement [uri local-name q-name]
 (push-chars)
 (set! *current* (push-content (peek *stack*) *current*))
 (set! *stack* (pop *stack*))
 (set! *state* :between)
 nil)
 (characters [^chars ch start length]
 (when-not (= *state* :chars)
 (set! *sb* (new StringBuilder)))
 (let [^StringBuilder sb *sb*]
 (.append sb ch (int start) (int length))
 (set! *state* :chars)))
 nil)
 (setDocumentLocator [locator])
 (startDocument [])
 (endDocument [])
 (startPrefixMapping [prefix uri])
 (endPrefixMapping [prefix])
 (ignorableWhitespace [ch start length])
 (processingInstruction [target data])
 (skippedEntity [name])
))))

(defn startparse-sax [s ch]
 (.. SAXParserFactory (newInstance) (newSAXParser) (parse s ch)))

(defn parse
 "Parses and loads the source s, which can be a File, InputStream or"

```

```

String naming a URI. Returns a tree of the xml/element struct-map,
which has the keys :tag, :attrs, and :content. and accessor fns tag,
attrs, and content. Other parsers can be supplied by passing
startparse, a fn taking a source and a ContentHandler and returning
a parser"
{:added "1.0"}
([s] (parse s startparse-sax))
([s startparse]
 (binding [*stack* nil
 current (struct element)
 state :between
 sb nil]
 (startparse s content-handler)
 ((:content *current*) 0)))

(defn emit-element [e]
 (if (instance? String e)
 (println e)
 (do
 (print (str "<" (name (:tag e)))))
 (when (:attrs e)
 (doseq [attr (:attrs e)]
 (print (str " " (name (key attr)) "=" (val attr)'""))
 (if (:content e)
 (do
 (println ">")
 (doseq [c (:content e)]
 (emit-element c))
 (println (str "</" (name (:tag e)) ">")))
 (println "/>")))))
 (defn emit [x]
 (println "<?xml version='1.0' encoding='UTF-8'?>")
 (emit-element x))

 ;(export '(tag attrs content parse element emit emit-element))

 ;(load-file "/Users/rich/dev/clojure/src/xml.clj")
 ;(def x (xml/parse "http://arstechnica.com/journals.rssx"))

 ——————
)
)
)
)

```

### 11.37 zip.clj

— zip.clj —

\getchunk{Clojure Copyright}

```

;functional hierarchical zipper, with navigation, editing and
;enumeration see Huet

(ns ^{:doc "Functional hierarchical zipper, with navigation, editing,
and enumeration. See Huet"
 :author "Rich Hickey"}
 clojure.zip
 (:refer-clojure :exclude (replace remove next)))

(defn zipper
 "Creates a new zipper structure.

branch? is a fn that, given a node, returns true if can have
children, even if it currently doesn't.

children is a fn that, given a branch node, returns a seq of its
children.

make-node is a fn that, given an existing node and a seq of
children, returns a new branch node with the supplied children.
root is the root node."
 {:added "1.0"}
 [branch? children make-node root]
 ^{:zip/branch? branch? :zip/children children
 :zip/make-node make-node}
 [root nil])

(defn seq-zip
 "Returns a zipper for nested sequences, given a root sequence"
 {:added "1.0"}
 [root]
 (zipper seq?
 identity
 (fn [node children] (with-meta children (meta node)))
 root))

(defn vector-zip
 "Returns a zipper for nested vectors, given a root vector"
 {:added "1.0"}
 [root]
 (zipper vector?
 seq
 (fn [node children] (with-meta (vec children) (meta node)))
 root))

(defn xml-zip
 "Returns a zipper for xml elements (as from xml/parse),
given a root element"
 {:added "1.0"})

```

```

[root]
 (zipper (complement string?)
 (comp seq :content)
 (fn [node children]
 (assoc node :content
 (and children (apply vector children))))
 root))

(defn node
 "Returns the node at loc"
 {:added "1.0"}
 [loc] (loc 0))

(defn branch?
 "Returns true if the node at loc is a branch"
 {:added "1.0"}
 [loc]
 ((:zip/branch? (meta loc)) (node loc)))

(defn children
 "Returns a seq of the children of node at loc, which must be a branch"
 {:added "1.0"}
 [loc]
 (if (branch? loc)
 ((:zip/children (meta loc)) (node loc))
 (throw (Exception. "called children on a leaf node"))))

(defn make-node
 "Returns a new branch node, given an existing node and new
 children. The loc is only used to supply the constructor."
 {:added "1.0"}
 [loc node children]
 ((:zip/make-node (meta loc)) node children))

(defn path
 "Returns a seq of nodes leading to this loc"
 {:added "1.0"}
 [loc]
 (:pnodes (loc 1)))

(defn lefts
 "Returns a seq of the left siblings of this loc"
 {:added "1.0"}
 [loc]
 (seq (:l (loc 1)))))

(defn rights
 "Returns a seq of the right siblings of this loc"
 {:added "1.0"}
 [loc]
 (seq (:r (loc 1)))))


```

```
(:r (loc 1)))

(defn down
 "Returns the loc of the leftmost child of the node at this loc, or
 nil if no children"
 {:added "1.0"}
 [loc]
 (when (branch? loc)
 (let [[node path] loc
 [c & cnext :as cs] (children loc)]
 (when cs
 (with-meta [c {:l []}
 :pnodes
 (if path (conj (:pnodes path) node) [node])
 :ppath path
 :r cnext] (meta loc)))))

(defn up
 "Returns the loc of the parent of the node at this loc, or nil if at
 the top"
 {:added "1.0"}
 [loc]
 (let [[node {:l :l, :ppath :ppath, :pnodes :pnodes, :r :r, :changed? :changed?, :as :path}] loc]
 (when pnodes
 (let [pnode (peek pnodes)]
 (with-meta (if changed?
 [(make-node loc pnode (concat l (cons node r)))
 (and ppath (assoc ppath :changed? true))]
 [pnode ppath])
 (meta loc))))))

(defn root
 "zips all the way up and returns the root node, reflecting any
 changes."
 {:added "1.0"}
 [loc]
 (if (= :end (loc 1))
 (node loc)
 (let [p (up loc)]
 (if p
 (recur p)
 (node loc)))))

(defn right
 "Returns the loc of the right sibling of the node at this loc, or nil"
 {:added "1.0"}
 [loc]
 (let [[node {:l :l, :r & rnxt :as rs} :r :as path] loc]
```

```

(defn rightmost
 "Returns the loc of the rightmost sibling of the node at this loc,
 or self"
 {:added "1.0"}
 [loc]
 (let [[node {l :l r :r :as path}] loc]
 (if (and path r)
 (with-meta [(last r)
 (assoc path :l (apply conj l node (butlast r)) :r nil)]
 (meta loc))
 loc)))

(defn left
 "Returns the loc of the left sibling of the node at this loc, or nil"
 {:added "1.0"}
 [loc]
 (let [[node {l :l r :r :as path}] loc]
 (when (and path (seq l))
 (with-meta [(peek 1) (assoc path :l (pop l) :r (cons node r))]
 (meta loc)))))

(defn leftmost
 "Returns the loc of the leftmost sibling of the node at this loc,
 or self"
 {:added "1.0"}
 [loc]
 (let [[node {l :l r :r :as path}] loc]
 (if (and path (seq l))
 (with-meta [(first l)
 (assoc path :l [] :r (concat (rest l) [node] r))]
 (meta loc))
 loc)))

(defn insert-left
 "Inserts the item as the left sibling of the node at this loc,
 without moving"
 {:added "1.0"}
 [loc item]
 (let [[node {l :l :as path}] loc]
 (if (nil? path)
 (throw (new Exception "Insert at top"))
 (with-meta [node (assoc path :l (conj l item) :changed? true)]
 (meta loc)))))

(defn insert-right
 "Inserts the item as the right sibling of the node at this loc,
 without moving"

```

```

{:added "1.0"}
[loc item]
(let [[node {r :r :as path}]] loc]
(if (nil? path)
(throw (new Exception "Insert at top"))
(with-meta [node (assoc path :r (cons item r) :changed? true)]
(meta loc)))))

(defn replace
"Replaces the node at this loc, without moving"
{:added "1.0"}
[loc node]
(let [[_ path] loc]
(with-meta [node (assoc path :changed? true)] (meta loc)))))

(defn edit
"Replaces the node at this loc with the value of (f node args)"
{:added "1.0"}
[loc f & args]
(replace loc (apply f (node loc) args)))

(defn insert-child
"Inserts the item as the leftmost child of the node at this loc,
without moving"
{:added "1.0"}
[loc item]
(replace loc
(make-node loc (node loc) (cons item (children loc)))))

(defn append-child
"Inserts the item as the rightmost child of the node at this loc,
without moving"
{:added "1.0"}
[loc item]
(replace loc
(make-node loc (node loc) (concat (children loc) [item]))))

(defn next
"Moves to the next loc in the hierarchy, depth-first. When reaching
the end, returns a distinguished loc detectable via end?. If already
at the end, stays there."
{:added "1.0"}
[loc]
(if (= :end (loc 1))
loc
(or
(and (branch? loc) (down loc))
(right loc)
(loop [p loc]
(if (up p)

```

```

(or (right (up p)) (recur (up p)))
[(node p) :end]))))

(defn prev
 "Moves to the previous loc in the hierarchy, depth-first. If already
 at the root, returns nil."
 {:added "1.0"}
 [loc]
 (if-let [lloc (left loc)]
 (loop [loc lloc]
 (if-let [child (and (branch? loc) (down loc))]
 (recur (rightmost child))
 loc)
 (up loc)))

(defn end?
 "Returns true if loc represents the end of a depth-first walk"
 {:added "1.0"}
 [loc]
 (= :end (loc 1)))

(defn remove
 "Removes the node at loc, returning the loc that would have preceded
 it in a depth-first walk."
 {:added "1.0"}
 [loc]
 (let [[node {l :l, ppath :ppath, pnodes
 :pnodes, rs :r, :as path}] loc]
 (if (nil? path)
 (throw (new Exception "Remove at top"))
 (if (pos? (count l))
 (loop [loc (with-meta [(peek l)
 (assoc path :l (pop l) :changed? true)]
 (meta loc))]
 (if-let [child (and (branch? loc) (down loc))]
 (recur (rightmost child))
 loc)
 (with-meta [(make-node loc (peek pnodes) rs)
 (and ppath (assoc ppath :changed? true))])
 (meta loc))))))

(comment

(load-file "/Users/rich/dev/clojure/src/zip.clj")
(refer 'zip)
(def data '[[a * b] + [c * d]])
(def dz (vector-zip data))

(right (down (right (right (down dz)))))
(lefts (right (down (right (right (down dz)))))))

```

```

(rights (right (down (right (right (down dz)))))))
(up (up (right (down (right (right (down dz)))))))
(path (right (down (right (right (down dz))))))

(-> dz down right right down right)
(-> dz down right right down right (replace '/') root)
(-> dz next next (edit str) next next next (replace '/') root)
(-> dz next next next next next next next next remove root)
(-> dz next next next next next next next next remove
 (insert-right 'e) root)
(-> dz next next next next next next next next remove up
 (append-child 'e) root)

(end? (-> dz next next next next next next next next remove next))

(-> dz next remove next remove root)

(loop [loc dz]
 (if (end? loc)
 (root loc)
 (recur (next (if (= '* (node loc))
 (replace loc '/')
 loc)))))

(loop [loc dz]
 (if (end? loc)
 (root loc)
 (recur (next (if (= '* (node loc))
 (remove loc)
 loc)))))

)

```

—————

## 11.38 pom-template.xml

— pom-template.xml —

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>org.clojure</groupId>
 <artifactId>clojure</artifactId>
 <name>clojure</name>

```

```
<version>@clojure-version@</version>
<url>http://clojure.org/</url>

<description>
 Clojure core environment and runtime library.
</description>

<licenses>
 <license>
 <name>Eclipse Public License 1.0</name>
 <url>http://opensource.org/licenses/eclipse-1.0.php</url>
 <distribution>repo</distribution>
 </license>
</licenses>

</project>
```

---

# Chapter 12

## test/clojure

### 12.1 test/test-clojure.clj

```
— test/test-clojure.clj —
\getchunk{Clojure Copyright}
;; clojure.test-clojure
;;
;; Tests for the facilities provided by Clojure
;;
;; scgilardi (gmail)
;; Created 22 October 2008

(ns clojure.test-clojure
 (:require [clojure.test :as t])
 (:gen-class))

(def test-names
 [:reader
 :printer
 :compilation
 :evaluation
 :special
 :macros
 :metadata
 :ns-libs
 :logic
 :predicates
 :control
 :data-structures
 :numbers)
```

```
:sequences
:for
:multimethods
:other-functions
:vars
:refs
:agents
:atoms
:parallel
:java-interop
:test
:test-fixtures
;; libraries
:clojure-set
:clojure-xml
:clojure-zip
:protocols
:genclass
:main
:vectors
:annotations
:pprint
:serialization
:rt
:repl
:java.io
:string
:java.javadoc
:java.shell
:transients
:def
:keywords
:data
:reflect
:errors
])

(def test-namespaces
 (map #(symbol (str "clojure.test-clojure." (name %)))
 test-names))

(defn run
 "Runs all defined tests"
 []
 (println "Loading tests...")
 (apply require :reload-all test-namespaces)
 (apply t/run-tests test-namespaces))

(defn run-ant
 "Runs all defined tests, prints report to *err*, throw if
```

```

failures. This works well for running in an ant java task."
[]

(let [rpt t/report]
 (binding [;; binding to *err* because, in ant, when the test
 ;; target runs after compile-clojure, *out* doesn't
 ;; print anything
 out *err*
 t/*test-out* *err*
 t/report
 (fn report [m]
 (if (= :summary (:type m))
 (do (rpt m)
 (if (or (pos? (:fail m)) (pos? (:error m)))
 (throw (new Exception (str
 (:fail m) " failures, "
 (:error m) " errors."))))
 (rpt m)))
 (run)))))

(defn -main
 "Run all defined tests from the command line"
 [& args]
 (run)
 (System/exit 0))

```

---

## 12.2 test/test'helper.clj

— test/test'helper.clj —

```

\getchunk{Clojure Copyright}

;; clojure.test-helper
;;
;; Utility functions shared by various tests in the Clojure
;; test suite
;;
;; tomfaulhaber (gmail)
;; Created 04 November 2010

(ns clojure.test-helper
 (:use clojure.test))

(let [nl (System/getProperty "line.separator")]
 (defn platform-newlines [s] (.replace s "\n" nl)))

```

```
(defn temp-ns
 "Create and return a temporary ns, using clojure.core + uses"
 [& uses]
 (binding [*ns* *ns*]
 (in-ns (gensym))
 (apply clojure.core/use 'clojure.core uses)
 ns))

(defmacro eval-in-temp-ns [& forms]
 '(binding [*ns* *ns*]
 (in-ns (gensym))
 (clojure.core/use 'clojure.core)
 (eval
 '(do ~@forms)))))

(defn causes
 [^Throwable throwable]
 (loop [causes []
 t throwable]
 (if t (recur (conj causes t) (.getCause t)) causes)))

;; this is how I wish clojure.test/thrown? worked...
;; Does body throw expected exception, anywhere in the .getCause chain?
(defmethod assert Expr 'fails-with-cause?
 [msg [_ exception-class msg-re & body :as form]]
 '(try
 ^@body
 (report {:type :fail, :message ^msg, :expected '^form, :actual nil})
 (catch Throwable t#
 (if (some (fn [cause#]
 (and
 (= ^exception-class (class cause#))
 (re-find ^msg-re (.getMessage cause#))))
 (causes t#)))
 (report {:type :pass, :message ^msg,
 :expected '^form, :actual t#})
 (report {:type :fail, :message ^msg,
 :expected '^form, :actual t#})))))

(defn get-field
 "Access to private or protected field. field-name is a symbol or
 keyword."
 ([klass field-name]
 (get-field klass field-name nil))
 ([klass field-name inst]
 (-> klass (.getDeclaredField (name field-name))
 (doto (.setAccessible true))
 (.get inst))))
```

```
(defn set-var-roots
 [maplike]
 (doseq [[var val] maplike]
 (alter-var-root var (fn [_] val)))))

(defn with-var-roots*
 "Temporarily set var roots, run block, then put original roots back."
 [root-map f & args]
 (let [originals (doall (map (fn [[var _]] [var @var]) root-map))]
 (set-var-roots root-map)
 (try
 (apply f args)
 (finally
 (set-var-roots originals)))))

(defmacro with-var-roots
 [root-map & body]
 `'(with-var-roots* ~root-map (fn [] ~@body)))

(defn exception
 "Use this function to ensure that execution of a program doesn't
 reach certain point."
 []
 (throw (new Exception "Exception which should never occur")))

```

## 12.3 test/agents.clj

— test/agents.clj —

```
\getchunk{Clojure Copyright}

;; Author: Shawn Hoover

(ns clojure.test-clojure.agents
 (:use clojure.test)
 (:import [java.util.concurrent CountDownLatch TimeUnit]))

(deftest handle-all-throwables-during-agent-actions
 ;; Bug fixed in r1198;
 ;; previously hung Clojure or didn't report agent errors
 ;; after OutOfMemoryError, yet wouldn't execute new actions.
 (let [agt (agent nil)]
 (send agt
 (fn [state] (throw (Throwable. "just testing Throwables"))))
 (try
```

```

;; Let the action finish; eat the "agent has errors" error
;; that bubbles up
(await-for 100 agt)
(catch RuntimeException _))
(is (instance? Throwable (first (agent-errors agt))))
(is (= 1 (count (agent-errors agt)))))

;; And now send an action that should work
(clear-agent-errors agt)
(is (= nil @agt))
(send agt nil?)
(is (true? (await-for 100 agt)))
(is (true? @agt)))))

(deftest default-modes
(is (= :fail (error-mode (agent nil))))
(is (= :continue (error-mode (agent nil :error-handler println)))))

(deftest continue-handler
(let [err (atom nil)
 agt (agent 0 :error-mode :continue
 :error-handler #(reset! err %&))]
 (send agt /)
 (is (true? (await-for 100 agt)))
 (is (= 0 @agt))
 (is (nil? (agent-error agt)))
 (is (= agt (first @err)))
 (is (true? (instance? ArithmeticException (second @err))))))

(deftest fail-handler
(let [err (atom nil)
 agt (agent 0 :error-mode :fail :error-handler #(reset! err %&))]
 (send agt /)
 (Thread/sleep 100)
 (is (true? (instance? ArithmeticException (agent-error agt))))
 (is (= 0 @agt))
 (is (= agt (first @err)))
 (is (true? (instance? ArithmeticException (second @err))))
 (is (thrown? RuntimeException (send agt inc))))))

(deftest can-send-from-handler-before-popping-action-that-caused-error
(let [latch (CountDownLatch. 1)
 target-agent (agent :before-error)
 handler (fn [agt err]
 (send target-agent
 (fn [_] (.countDown latch))))
 failing-agent (agent nil :error-handler handler)]
 (send failing-agent (fn [] (throw (RuntimeException.))))
 (is (.await latch 10 TimeUnit/SECONDS)))))


```

```
(deftest
 can-send-to-self-from-handler-before-popping-action-that-caused-error
 (let [latch (CountDownLatch. 1)
 handler (fn [agt err]
 (send *agent*
 (fn [_] (.countDown latch))))
 failing-agent (agent nil :error-handler handler)]
 (send failing-agent (fn [] (throw (RuntimeException.))))
 (is (.await latch 10 TimeUnit/SECONDS)))

(deftest restart-no-clear
 (let [p (promise)
 agt (agent 1 :error-mode :fail)]
 (send agt (fn [v] @p))
 (send agt /)
 (send agt inc)
 (send agt inc)
 (deliver p 0)
 (Thread/sleep 100)
 (is (= 0 @agt))
 (is (= ArithmeticException (class (agent-error agt))))
 (restart-agent agt 10)
 (is (true? (await-for 100 agt)))
 (is (= 12 @agt))
 (is (nil? (agent-error agt)))))

(deftest restart-clear
 (let [p (promise)
 agt (agent 1 :error-mode :fail)]
 (send agt (fn [v] @p))
 (send agt /)
 (send agt inc)
 (send agt inc)
 (deliver p 0)
 (Thread/sleep 100)
 (is (= 0 @agt))
 (is (= ArithmeticException (class (agent-error agt))))
 (restart-agent agt 10 :clear-actions true)
 (is (true? (await-for 100 agt)))
 (is (= 10 @agt))
 (is (nil? (agent-error agt)))
 (send agt inc)
 (is (true? (await-for 100 agt)))
 (is (= 11 @agt))
 (is (nil? (agent-error agt)))))

(deftest invalid-restart
 (let [p (promise)
 agt (agent 2 :error-mode :fail :validator even?)]
 (is (thrown? RuntimeException (restart-agent agt 4))))
```

```

(send agt (fn [v] @p))
(send agt (partial + 2))
(send agt (partial + 2))
(deliver p 3)
(Thread/sleep 100)
(is (= 2 @agt))
(is (= IllegalStateException (class (agent-error agt))))
(is (thrown? RuntimeException (restart-agent agt 5)))
(restart-agent agt 6)
(is (true? (await-for 100 agt)))
(is (= 10 @agt))
(is (nil? (agent-error agt)))))

(deftest earmuff-agent-bound
(let [a (agent 1)]
 (send a (fn [_] *agent*)
 (await a)
 (is (= a @a)))))

(def ^:dynamic *bind-me* :root-binding)

(deftest thread-conveyance-to-agents
(let [a (agent nil)]
 (doto (Thread.
 (fn []
 (binding [*bind-me* :thread-binding]
 (send a (constantly *bind-me*)))
 (await a)))
 (.start)
 (.join)))
 (is (= @a :thread-binding)))))

; http://clojure.org/agents

; agent
; deref, @-reader-macro, agent-errors
; send send-off clear-agent-errors
; await await-for
; set-validator get-validator
; add-watch remove-watch
; shutdown-agents

```

---

## 12.4 test/annotations.clj

— test/annotations.clj —

```
\getchunk{Clojure Copyright}

;; Authors: Stuart Halloway, Rich Hickey

(ns clojure.test-clojure.annotations
 (:use clojure.test))

(defn vm-has-ws-annotations?
 "Does the vm have the ws annotations we use to test some
 annotation features. If not, fall back to Java 5 tests."
 []
 (try
 (doseq [n ["javax.xml.ws.soap.Addressing"
 "javax.xml.ws.WebServiceRef"
 "javax.xml.ws.WebServiceRefs"]]
 (Class.forName n))
 true
 (catch ClassNotFoundException e
 false)))

(if (vm-has-ws-annotations?)
 (load "annotations/java_6_and_later")
 (load "annotations/java_5"))

```

## 12.5 test/atoms.clj

— test/atoms.clj —

```
\getchunk{Clojure Copyright}

;; Author: Frantisek Sodomka

(ns clojure.test-clojure.atoms
 (:use clojure.test))

; http://clojure.org/atoms

; atom
; deref, @-reader-macro
```

```
; swap! reset!
; compare-and-set!
```

---

## 12.6 test/clojure.set.clj

— test/clojure.set.clj —

```
\getchunk{Clojure Copyright}

;; Author: Frantisek Sodomka

(ns clojure.test-clojure.clojure-set
 (:use clojure.test)
 (:require [clojure.set :as set]))

(deftest test-union
 (are [x y] (= x y)
 (set/union) #{})
 ; identity
 (set/union #{}) #{}
 (set/union #{1}) #{1}
 (set/union #{1 2 3}) #{1 2 3}
 ; 2 sets, at least one is empty
 (set/union #{}) #{}
 (set/union #{}) #{1}
 (set/union #{}) #{1 2 3}
 (set/union #{1}) #{}
 (set/union #{1 2 3}) #{1 2 3}
 ; 2 sets
 (set/union #{1} #{2}) #{1 2}
 (set/union #{1} #{1 2}) #{1 2}
 (set/union #{2} #{1 2}) #{1 2}
 (set/union #{1 2} #{3}) #{1 2 3}
 (set/union #{1 2} #{2 3}) #{1 2 3}
 ; 3 sets, some are empty
 (set/union #{}) #{} #{}
 (set/union #{1} #{}) #{1}
 (set/union #{}) #{1} #{}
 (set/union #{}) #{1} #{1}
```

```

(set/union #{1 2} #{2 3} #{}) #{1 2 3}

; 3 sets
(set/union #{1 2} #{3 4} #{5 6}) #{1 2 3 4 5 6}
(set/union #{1 2} #{2 3} #{1 3 4}) #{1 2 3 4}

; different data types
(set/union #{1 2} #{:a :b} #{nil}
 #{false true} #{\c "abc"} #{[] [1 2]}
 #{()} {:a 1}} #{#{}} #{1 2}}
 #{1 2 :a :b nil false true \c "abc"
 [] [1 2] {} {:a 1}} #{} #{1 2}}

; different types of sets
(set/union (hash-set) (hash-set 1 2) (hash-set 2 3))
 (hash-set 1 2 3)
(set/union (sorted-set) (sorted-set 1 2) (sorted-set 2 3))
 (sorted-set 1 2 3)
(set/union (hash-set) (hash-set 1 2) (hash-set 2 3))
 (sorted-set) (sorted-set 4 5) (sorted-set 5 6))
 (hash-set 1 2 3 4 5 6) ; also equals (sorted-set 1 2 3 4 5 6)
))

(deftest test-intersection
 ; at least one argument is needed
 (is (thrown? IllegalArgumentException (set/intersection)))

 (are [x y] (= x y)
 ; identity
 (set/intersection #{}) #{}
 (set/intersection #{1}) #{1}
 (set/intersection #{1 2 3}) #{1 2 3}

 ; 2 sets, at least one is empty
 (set/intersection #{}) #{}
 (set/intersection #{}) #{1}
 (set/intersection #{}) #{1 2 3})
 (set/intersection #{1}) #{}
 (set/intersection #{1 2 3}) #{}

 ; 2 sets
 (set/intersection #{1 2} #{1 2}) #{1 2}
 (set/intersection #{1 2} #{3 4}) #{}
 (set/intersection #{1 2} #{1}) #{1}
 (set/intersection #{1 2} #{2}) #{2}
 (set/intersection #{1 2 4} #{2 3 4 5}) #{2 4}

 ; 3 sets, some are empty
 (set/intersection #{}) #{} #{}
 (set/intersection #{1}) #{} #{}
)
)
```

```

(set/intersection #{1} #{1} #{}) #{}
(set/intersection #{1} #{} #{1}) #{}
(set/intersection #{1 2} #{2 3} #{}) #{}

; 3 sets
(set/intersection #{1 2} #{2 3} #{5 2}) #{2}
(set/intersection #{1 2 3} #{1 3 4} #{1 3}) #{1 3}
(set/intersection #{1 2 3} #{3 4 5} #{8 2 3}) #{3}

; different types of sets
(set/intersection (hash-set 1 2) (hash-set 2 3)) #{2}
(set/intersection (sorted-set 1 2) (sorted-set 2 3)) #{2}
(set/intersection
 (hash-set 1 2) (hash-set 2 3)
 (sorted-set 1 2) (sorted-set 2 3)) #{2})

(deftest test-difference
 (are [x y] (= x y)
 ; identity
 (set/difference #{}) #{}
 (set/difference #{1}) #{1}
 (set/difference #{1 2 3}) #{1 2 3}

 ; 2 sets
 (set/difference #{1 2} #{1 2}) #{}
 (set/difference #{1 2} #{3 4}) #{1 2}
 (set/difference #{1 2} #{1}) #{2}
 (set/difference #{1 2} #{2}) #{1}
 (set/difference #{1 2 4} #{2 3 4 5}) #{1}

 ; 3 sets
 (set/difference #{1 2} #{2 3} #{5 2}) #{1}
 (set/difference #{1 2 3} #{1 3 4} #{1 3}) #{2}
 (set/difference #{1 2 3} #{3 4 5} #{8 2 3}) #{1}))

(deftest test-select
 (are [x y] (= x y)
 (set/select integer? #{}) #{}
 (set/select integer? #{1 2}) #{1 2}
 (set/select integer? #{1 2 :a :b :c}) #{1 2}
 (set/select integer? #{:a :b :c}) #{}))

(def compositions
 #{{:name "Art of the Fugue" :composer "J. S. Bach"}
 {:name "Musical Offering" :composer "J. S. Bach"}
 {:name "Requiem" :composer "Giuseppe Verdi"}
 {:name "Requiem" :composer "W. A. Mozart"}})

(deftest test-project
 (are [x y] (= x y)

```

```

(set/project compositions [:name]) #{{:name "Art of the Fugue"}
 {:name "Requiem"}
 {:name "Musical Offering"}}
(set/project compositions [:composer]) #{{:composer "W. A. Mozart"}
 {:composer "Giuseppe Verdi"}
 {:composer "J. S. Bach"}}
(set/project compositions [:year]) #{{}}
(set/project #{{}} [:name]) #{{}}))

(deftest test-rename
 (are [x y] (= x y)
 (set/ rename compositions {:name :title})
 #{{:title "Art of the Fugue" :composer "J. S. Bach"}
 {:title "Musical Offering" :composer "J. S. Bach"}
 {:title "Requiem" :composer "Giuseppe Verdi"}
 {:title "Requiem" :composer "W. A. Mozart"}}
 (set/ rename compositions {:year :decade})
 #{{:name "Art of the Fugue" :composer "J. S. Bach"}
 {:name "Musical Offering" :composer "J. S. Bach"}
 {:name "Requiem" :composer "Giuseppe Verdi"}
 {:name "Requiem" :composer "W. A. Mozart"}}
 (set/ rename #{{}} {:year :decade}) #{{}}))

(deftest test-rename-keys
 (are [x y] (= x y)
 (set/ rename-keys {:a "one" :b "two"} {:a :z}) {:z "one" :b "two"}
)))

(deftest test-index
 (are [x y] (= x y)
 (set/ index #{{:c 2} {:b 1} {:a 1 :b 2}} [:b]) #{{:b 2}
 #{{:a 1 :b 2}}, {:b 1} #{{:b 1}} {} #{{:c 2}}}
)))

(deftest test-join
 (are [x y] (= x y)
 (set/ join compositions compositions) compositions
 (set/ join compositions
 #{{:name "Art of the Fugue" :genre "Classical"}})
 #{{:name "Art of the Fugue" :composer "J. S. Bach"
 :genre "Classical"}}
)))

(deftest test-map-invert
 (are [x y] (= x y)
 (set/ map-invert {:a "one" :b "two"}) {"one" :a "two" :b})))

(deftest test-subset?
 (are [sub super] (set/ subset? sub super)
 #{} #{}))

```

```

#{[]} #{1}
#{1} #{1}
#{1 2} #{1 2}
#{1 2} #{1 2 42}
#{false} #{false}
#{nil} #{nil}
#{nil} #{nil false}
#{1 2 nil} #{1 2 nil 4)
(are [notsub super] (not (set/subset? notsub super))
 #{1} #{}
 #{2} #{1}
 #{1 3} #{1}
 #{nil} #{false}
 #{false} #{nil}
 #{false nil} #{nil}
 #{1 2 nil} #{1 2}))

(deftest test-superset?
 (are [super sub] (set/superset? super sub)
 #{[]} #{}
 #{1} #{}
 #{1} #{1}
 #{1 2} #{1 2}
 #{1 2 42} #{1 2}
 #{false} #{false}
 #{nil} #{nil}
 #{false nil} #{false}
 #{1 2 4 nil false} #{1 2 nil})
 (are [notsuper sub] (not (set/superset? notsuper sub))
 #{} #{1}
 #{2} #{1}
 #{1} #{1 3}
 #{nil} #{false}
 #{false} #{nil}
 #{nil} #{false nil}
 #{nil 2 3} #{false nil 2 3}))
```

---

## 12.7 test/clojure/xml.clj

— test/clojure/xml.clj —

```
\getchunk{Clojure Copyright}

;; Author: Frantisek Sodomka
```

```
(ns clojure.test-clojure.clojure-xml
 (:use clojure.test)
 (:require [clojure.xml :as xml]))

; parse
; emit-element
; emit
```

— — — — —

## 12.8 test/clojure'zip.clj

— test/clojure'zip.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka

(ns clojure.test-clojure.clojure-zip
 (:use clojure.test)
 (:require [clojure.zip :as zip]))

; zipper
;
; seq-zip
;
; vector-zip
;
; xml-zip
;
;
; node
;
; branch?
;
; children
;
; make-node
;
; path
;
; lefts
;
; rights
;
; down
;
; up
;
; root
;
; right
;
; rightmost
```

```
; left
; leftmost
;
; insert-left
; insert-right
; replace
; edit
; insert-child
; append-child
; next
; prev
; end?
; remove
```



## 12.9 test/compilation.clj

— test/compilation.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka

(ns clojure.test-clojure.compilation
 (:use clojure.test))

; http://clojure.org/compilation

; compile
; gen-class, gen-interface

(deftest test-compiler-metadata
 (let [m (meta #'when)]
 (are [x y] (= x y)
 (list? (:arglists m)) true
 (> (count (:arglists m)) 0) true

 (string? (:doc m)) true
 (> (.length (:doc m)) 0) true

 (string? (:file m)) true
 (> (.length (:file m)) 0) true))
```

```

(integer? (:line m)) true
(> (:line m) 0) true

(:macro m) true
(:name m) 'when)))

(deftest test-embedded-constants
 (testing "Embedded constants"
 (is (eval '(= Boolean/TYPE ~Boolean/TYPE)))
 (is (eval '(= Byte/TYPE ~Byte/TYPE)))
 (is (eval '(= Character/TYPE ~Character/TYPE)))
 (is (eval '(= Double/TYPE ~Double/TYPE)))
 (is (eval '(= Float/TYPE ~Float/TYPE)))
 (is (eval '(= Integer/TYPE ~Integer/TYPE)))
 (is (eval '(= Long/TYPE ~Long/TYPE)))
 (is (eval '(= Short/TYPE ~Short/TYPE)))))

(deftest test-compiler-resolution
 (testing
 "resolve nonexistent class create should return nil (assembly #262)"
 (is (nil? (resolve 'NonExistentClass.)))))

(deftest test-no-recur-across-try
 (testing "don't recur to function from inside try"
 (is (thrown? Exception (eval '(fn [x] (try (recur 1)))))))
 (testing "don't recur to loop from inside try"
 (is (thrown? Exception (eval '(loop [x] (try (recur 1)))))))
 (testing "don't get confused about what the recur is targeting"
 (is (thrown? Exception (eval '(loop [x] (try (fn [x] (recur 1)))))))
 (testing "don't allow recur across binding"
 (is (thrown? Exception (eval '(fn [x] (binding [+ *] (recur 1)))))))
 (testing "allow loop/recur inside try"
 (is (try
 (eval '(try (loop [x 3] (if (zero? x) x (recur (dec x))))))
 (catch Exception _))))
 (testing "allow fn/recur inside try"
 (is (try
 (eval '(try
 ((fn [x]
 (if (zero? x)
 x
 (recur (dec x))))
 3)))
 (catch Exception _))))))

 ——————

```

## 12.10 test/control.clj

— test/control.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka, Mike Hinckey, Stuart Halloway

;;
;; Test "flow control" constructs.
;;

(ns clojure.test-clojure.control
 (:use clojure.test
 [clojure.test-helper :only (exception)]))

;; *** Helper functions ***

(defn maintains-identity [f]
 (are [x] (= (f x) x)
 nil
 false true
 0 42
 0.0 3.14
 2/3
 0M 1M
 \c
 "" "abc"
 'sym
 :kw
 () '(1 2)
 [] [1 2]
 {} f:a 1 :b 2}
 #{ } #{1 2})))

; http://clojure.org/special_forms
; http://clojure.org/macros

(deftest test-do
 (are [x y] (= x y)
 ; no params => nil
 (do) nil

 ; return last
 (do 1) 1
 (do 1 2) 2
 (do 1 2 3 4 5) 5
```

```

; evaluate and return last
(let [a (atom 0)]
 (do (reset! a (+ @a 1)) ; 1
 (reset! a (+ @a 1)) ; 2
 (reset! a (+ @a 1)) ; 3
 @a) 3)

; identity (= (do x) x)
(maintains-identity (fn [] (do _))))

;; loop/recur
(deftest test-loop
 (are [x y] (= x y)
 1 (loop []
 1)
 3 (loop [a 1]
 (if (< a 3)
 (recur (inc a))
 a))
 [2 4 6] (loop [a []
 b [1 2 3]]
 (if (seq b)
 (recur (conj a (* 2 (first b)))
 (next b))
 a))
 [6 4 2] (loop [a []
 b [1 2 3]]
 (if (seq b)
 (recur (conj a (* 2 (first b)))
 (next b))
 a)))
)
)

;; throw, try
; if: see logic.clj

(deftest test-when
 (are [x y] (= x y)
 1 (when true 1)
 nil (when true)
 nil (when false)
 nil (when false (exception)))
))

(deftest test-when-not

```

```

(are [x y] (= x y)
 1 (when-not false 1)
 nil (when-not true)
 nil (when-not false)
 nil (when-not true (exception))
))

(deftest test-if-not
 (are [x y] (= x y)
 1 (if-not false 1)
 1 (if-not false 1 (exception))
 nil (if-not true 1)
 2 (if-not true 1 2)
 nil (if-not true (exception))
 1 (if-not true (exception) 1)
))

(deftest test-when-let
 (are [x y] (= x y)
 1 (when-let [a 1]
 a)
 2 (when-let [[a b] '(1 2)]
 b)
 nil (when-let [a false]
 (exception)))
))

(deftest test-if-let
 (are [x y] (= x y)
 1 (if-let [a 1]
 a)
 2 (if-let [[a b] '(1 2)]
 b)
 nil (if-let [a false]
 (exception))
 1 (if-let [a false]
 a 1)
 1 (if-let [[a b] nil]
 b 1)
 1 (if-let [a false]
 (exception)
 1)
))

(deftest test-when-first
 (are [x y] (= x y)
 1 (when-first [a [1 2]]
 a)
 2 (when-first [[a b] '((1 2) 3)]
 b)
))

```

```

nil (when-first [a nil]
 (exception))
))

(deftest test-cond
 (are [x y] (= x y)
 (cond) nil

 (cond nil true) nil
 (cond false true) nil

 (cond true 1 true (exception)) 1
 (cond nil 1 false 2 true 3 true 4) 3
 (cond nil 1 false 2 true 3 true (exception)) 3)

 ; false
 (are [x] (= (cond x :a true :b) :b)
 nil false)

 ; true
 (are [x] (= (cond x :a true :b) :a)
 true
 0 42
 0.0 3.14
 2/3
 #M 1M
 \c
 "" "abc"
 'sym
 :kw
 () '(1 2)
 [] [1 2]
 {} {:a 1 :b 2}
 #{ } #{1 2})

 ; evaluation
 (are [x y] (= x y)
 (cond (> 3 2) (+ 1 2) true :result true (exception)) 3
 (cond (< 3 2) (+ 1 2) true :result true (exception)) :result)

 ; identity (= (cond true x) x)
 (maintains-identity (fn [_] (cond true _))))

(deftest test-condp
 (are [x] (= :pass x)
 (condp = 1
 1 :pass
 2 :fail)

```

```

(condp = 1
 2 :fail
 1 :pass)
(condp = 1
 2 :fail
 :pass)
(condp = 1
 :pass)
(condp = 1
 2 :fail
 ;;= doc of condp says result-expr is returned
 ;;= shouldn't it say similar to cond: "evaluates and returns
 ;;= the value of the corresponding expr and doesn't evaluate
 ;;= any of the other tests or exprs."
 (identity :pass))
(condp + 1
 1 :>> #(if (= % 2) :pass :fail))
(condp + 1
 1 :>> #(if (= % 3) :fail :pass))
)
(is (thrown? IllegalArgumentException
 (condp = 1)
))
(is (thrown? IllegalArgumentException
 (condp = 1
 2 :fail)
))
)

; [for, doseq (for.clj)]

(deftest test-dotimes
 ;; dotimes always returns nil
 (is (= nil (dotimes [n 1] n)))
 ;; test using an atom since dotimes is for modifying
 ;; test executes n times
 (is (= 3
 (let [a (atom 0)]
 (dotimes [n 3]
 (swap! a inc))
 @a)
)))
 ;; test all values of n
 (is (= [0 1 2]
 (let [a (atom [])]
 (dotimes [n 3]
 (swap! a conj n))
 @a)))
 (is (= []

```

```

(let [a (atom [])]
 (dotimes [n 0]
 (swap! a conj n))
 @a)))
)

(deftest test-while
 (is (= nil (while nil (throw (Exception. "never")))))
 (is (= [0 nil]
 ;;= a will dec to 0
 ;;= while always returns nil
 (let [a (atom 3)
 w (while (pos? @a)
 (swap! a dec))]
 [@a w])))
 (is (thrown? Exception
 (while true (throw (Exception. "expected to throw")))))
)

; locking, monitor-enter, monitor-exit

; case
(deftest test-case
 (testing "can match many kinds of things"
 (let [two 2
 test-fn
 #(case %
 1 :number
 "foo" :string
 \a :char
 pow :symbol
 :zap :keyword
 (2 \b "bar") :one-of-many
 [1 2] :sequential-thing
 {:a 2} :map
 {:r 2 :d 2} :droid
 #{2 3 4 5} :set
 [1 [[[2]]]] :deeply-nested
 :default)]
 (are [result input] (= result (test-fn input))
 :number 1
 :string "foo"
 :char \a
 :keyword :zap
 :symbol 'pow
 :one-of-many 2
 :one-of-many \b
 :one-of-many "bar"
 :sequential-thing [1 2]
 :sequential-thing (list 1 2))))
```

```

:sequential-thing [1 two]
:map {:a 2}
:map {:a two}
:set #{2 3 4 5}
:set #{two 3 4 5}
:default #{2 3 4 5 6}
:droid {:r 2 :d 2}
:deeply-nested [1 [[[two]]]]
:default :anything-not-appearing-above)))
(testing "throws IllegalArgumentException if no match"
 (is (thrown-with-msg?
 IllegalArgumentException #'No matching clause: 2"
 (case 2 1 :ok))))
(testing "sorting doesn't matter"
 (let [test-fn
 #(case %
 {:b 2 :a 1} :map
 #{3 2 1} :set
 :default)]
 (are [result input] (= result (test-fn input))
 :map {:a 1 :b 2}
 :map (sorted-map :a 1 :b 2)
 :set #{3 2 1}
 :set (sorted-set 2 1 3))))
(testing "test constants are *not* evaluated"
 (let [test-fn
 ;; never write code like this...
 #(case %
 (throw (RuntimeException. "boom")) :piece-of-throw-expr
 :no-match)])
 (are [result input] (= result (test-fn input))
 :piece-of-throw-expr 'throw
 :piece-of-throw-expr '[RuntimeException. "boom"]
 :no-match nil))))

```

---

## 12.11 test/data.clj

— test/data.clj —

```

\getchunk{Clojure Copyright}

(ns clojure.test-clojure.data
 (:use clojure.data clojure.test))

(deftest diff-test

```

```
(are [d x y] (= d (diff x y))
 [nil nil nil] nil nil
 [1 2 nil] 1 2
 [nil nil [1 2 3]] [1 2 3] '(1 2 3)
 [1 [:a :b] nil] 1 [:a :b]
 [{:a 1} :b nil] {:a 1} :b
 [:team #{:p1 :p2} nil] :team #{:p1 :p2}
 [{0 :a} [:a] nil] {0 :a} [:a]
 [nil [nil 2] [1]] [1] [1 2]
 [nil nil [1 2]] [1 2] (into-array [1 2])
 [{:a} #{:b} #{:c :d}] #{:a :c :d} #{:b :c :d}
 [nil nil {:a 1}] {:a 1} {:a 1}
 [{:a #{}2} {:a #{}4} {:a #{}3}] {:a #{}2 3} {:a #{}3 4}
 [{:a {:c [1]} } {:a {:c [0]} }]
 {:a {:c [nil 2] :b 1}}]
 {:a {:b 1 :c [1 2]} } {:a {:b 1 :c [0 2]}}))
```

---

## 12.12 test/data`structures.clj

— test/data`structures.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka

(ns clojure.test-clojure.data-structures
 (:use clojure.test))

;; *** Helper functions **

(defn diff [s1 s2]
 (seq (reduce disj (set s1) (set s2)))))

;; *** General **

(defstruct equality-struct :a :b)

(deftest test-equality
 ; nil is not equal to any other value
 (are [x] (not (= nil x)))
 true false
```

```

0 0.0
\space
"" #"""
() [] #{ } {}
(lazy-seq nil) ; SVN 1292: fixed (= (lazy-seq nil) nil)
(lazy-seq ())
(lazy-seq [])
(lazy-seq {})
(lazy-seq #{})
(lazy-seq "")
(lazy-seq (into-array []))
(new Object))

; numbers equality across types (see tests below - NOT IMPLEMENTED YET)

; ratios
(is (== 1/2 0.5))
(is (== 1/1000 0.001))
(is (not= 2/3 0.666666666666666))

; vectors equal other seqs by items equality
(are [x y] (= x y)
 '() [] ; regression fixed in r1208; was not equal
 '(1) [1]
 '(1 2) [1 2]

 [] '() ; same again, but vectors first
 [1] '(1)
 [1 2] '(1 2))
(is (not= [1 2] '(2 1))) ; order of items matters

; list and vector vs. set and map
(are [x y] (not= x y)
 ; only () equals []
 () #{}
 () {}
 [] #{}
 [] {}
 #{} {}
 ; only '(1) equals [1]
 '(1) #{1}
 [1] #{1})

; sorted-map, hash-map and array-map - classes differ,
; but content is equal

;; TODO: reimplement all-are with new do-template?
;; (all-are (not= (class _1) (class _2))
;; (sorted-map :a 1)
;; (hash-map :a 1))

```



```
[1 2 3] 3

#{ } 0
#{1} 1
#{1 2 3} 3

{} 0
{:a 1} 1
{:a 1 :b 2 :c 3} 3

"" 0
"a" 1
"abc" 3

(into-array []) 0
(into-array [1]) 1
(into-array [1 2 3]) 3

(java.util.ArrayList. []) 0
(java.util.ArrayList. [1]) 1
(java.util.ArrayList. [1 2 3]) 3

(java.util.HashMap. {}) 0
(java.util.HashMap. {:a 1}) 1
(java.util.HashMap. {:a 1 :b 2 :c 3}) 3)

; different types
(are [x] (= (count [x]) 1)
 nil true false
 0 0.0 "" \space
 () [] #{ } {}))

(deftest test-conj
 ; doesn't work on strings or arrays
 (is (thrown? ClassCastException (conj "" \a)))
 (is (thrown? ClassCastException (conj (into-array []) 1)))

 (are [x y] (= x y)
 (conj nil 1) '(1)
 (conj nil 3 2 1) '(1 2 3)

 (conj nil nil) '(nil)
 (conj nil nil nil) '(nil nil)
 (conj nil nil nil 1) '(1 nil nil)

 ; list -> conj puts the item at the front of the list
 (conj () 1) '(1)
 (conj () 1 2) '(2 1))
```

```

(conj '(2 3) 1) '(1 2 3)
(conj '(2 3) 1 4 3) '(3 4 1 2 3)

(conj () nil) '(nil)
(conj () ()) '()

; vector -> conj puts the item at the end of the vector
(conj [] 1) [1]
(conj [] 1 2) [1 2]

(conj [2 3] 1) [2 3 1]
(conj [2 3] 1 4 3) [2 3 1 4 3]

(conj [] nil) [nil]
(conj [] []) []

; map -> conj expects another (possibly single entry) map as the
; item, and returns a new map which is the old map plus the
; entries from the new, which may overwrite entries of the old.
; conj also accepts a MapEntry or a vector of two items
; (key and value).

(conj {} {}) {}
(conj {} {:a 1}) {:a 1}
(conj {} {:a 1 :b 2}) {:a 1 :b 2}
(conj {} {:a 1 :b 2} {:c 3}) {:a 1 :b 2 :c 3}
(conj {} {:a 1 :b 2} {:a 3 :c 4}) {:a 3 :b 2 :c 4}

(conj {:a 1} {:a 7}) {:a 7}
(conj {:a 1} {:b 2}) {:a 1 :b 2}
(conj {:a 1} {:a 7 :b 2}) {:a 7 :b 2}
(conj {:a 1} {:a 7 :b 2} {:c 3}) {:a 7 :b 2 :c 3}
(conj {:a 1} {:a 7 :b 2} {:b 4 :c 5}) {:a 7 :b 4 :c 5}

(conj {} (first {:a 1})) {:a 1} ; MapEntry
(conj {:a 1} (first {:b 2})) {:a 1 :b 2}
(conj {:a 1} (first {:a 7})) {:a 7}
(conj {:a 1} (first {:b 2}) (first {:a 5})) {:a 5 :b 2}

(conj {} [:a 1]) {:a 1} ; vector
(conj {:a 1} [:b 2]) {:a 1 :b 2}
(conj {:a 1} [:a 7]) {:a 7}
(conj {:a 1} [:b 2] [:a 5]) {:a 5 :b 2}

(conj {} {nil {}}) {nil {}}
(conj {} {{}} nil) {{} nil}
(conj {} {{} {}}) {{} {}}

; set
(conj #{} 1) #{1}

```

```

(conj #{}) 1 2 3) #{1 2 3}

(conj #{2 3} 1) #{3 1 2}
(conj #{3 2} 1) #{1 2 3}

(conj #{2 3} 2) #{2 3}
(conj #{2 3} 2 3) #{2 3}
(conj #{2 3} 4 1 2 3) #{1 2 3 4}

(conj #{}) nil) #{nil}
(conj #{}) #{}) #{#{}})))

;; *** Lists and Vectors ***

(deftest test-peek
; doesn't work for sets and maps
(is (thrown? ClassCastException (peek #{1})))
(is (thrown? ClassCastException (peek {:a 1})))

(are [x y] (= x y)
 (peek nil) nil

 ; list = first
 (peek ()) nil
 (peek '(1)) 1
 (peek '(1 2 3)) 1

 (peek '(nil)) nil ; special cases
 (peek '(1 nil)) 1
 (peek '(nil 2)) nil
 (peek '()) ()
 (peek '() nil)) ()
 (peek '() 2 nil)) ()

 ; vector = last
 (peek []) nil
 (peek [1]) 1
 (peek [1 2 3]) 3

 (peek [nil]) nil ; special cases
 (peek [1 nil]) nil
 (peek [nil 2]) 2
 (peek [[]]) []
 (peek [[] nil]) nil
 (peek [[] 2 nil]) nil)))

(deftest test-pop
; doesn't work for sets and maps

```



```

(list 1) '(1)
(list 1 2) '(1 2)

; nesting
(list 1 (list 2 3) (list 3 (list 4 5 (list 6 (list 7)))))
 '(1 (2 3) (3 (4 5 (6 (7)))))

; different data structures
(list true false nil)
 '(true false nil)
(list 1 2.5 2/3 "ab" \x 'cd :kw)
 '(1 2.5 2/3 "ab" \x cd :kw)
(list (list 1 2) [3 4] {:a 1 :b 2} #{:c :d})
 '((1 2) [3 4] {:a 1 :b 2} #{:c :d})

; evaluation
(list (+ 1 2) [(+ 2 3) 'a] (list (* 2 3) 8))
 '(3 [5 a] (6 8))

; special cases
(list nil) '(nil)
(list 1 nil) '(1 nil)
(list nil 2) '(nil 2)
(list ()) '()
(list 1 ()) '(1 ())
(list () 2) '(() 2))

;; *** Maps (IPersistentMap) ***

(deftest test-find
 (are [x y] (= x y)
 (find {}) :a) nil

 (find {:a 1} :a) [:a 1]
 (find {:a 1} :b) nil

 (find {:a 1 :b 2} :a) [:a 1]
 (find {:a 1 :b 2} :b) [:b 2]
 (find {:a 1 :b 2} :c) nil

 (find {} nil) nil
 (find {:a 1} nil) nil
 (find {:a 1 :b 2} nil) nil))

(deftest test-contains?
 ; contains? is designed to work preferably on maps and sets
 (are [x y] (= x y)
 (contains? {}) :a) false

```

```

(contains? {} nil) false

(contains? {:a 1} :a) true
(contains? {:a 1} :b) false
(contains? {:a 1} nil) false

(contains? {:a 1 :b 2} :a) true
(contains? {:a 1 :b 2} :b) true
(contains? {:a 1 :b 2} :c) false
(contains? {:a 1 :b 2} nil) false

; sets
(contains? #{}) false
(contains? #{}) nil

(contains? #{1} 1) true
(contains? #{1} 2) false
(contains? #{1} nil) false

(contains? #{1 2 3} 1) true
(contains? #{1 2 3} 3) true
(contains? #{1 2 3} 10) false
(contains? #{1 2 3} nil) false

; numerically indexed collections (e.g. vectors and Java arrays)
; => test if the numeric key is WITHIN THE RANGE OF INDEXES
(are [x y] (= x y)
 (contains? [] 0) false
 (contains? [] -1) false
 (contains? [] 1) false

 (contains? [1] 0) true
 (contains? [1] -1) false
 (contains? [1] 1) false

 (contains? [1 2 3] 0) true
 (contains? [1 2 3] 2) true
 (contains? [1 2 3] 3) false
 (contains? [1 2 3] -1) false

; arrays
(contains? (into-array []) 0) false
(contains? (into-array []) -1) false
(contains? (into-array []) 1) false

(contains? (into-array [1]) 0) true
(contains? (into-array [1]) -1) false
(contains? (into-array [1]) 1) false

(contains? (into-array [1 2 3]) 0) true

```

```

(contains? (into-array [1 2 3]) 2) true
(contains? (into-array [1 2 3]) 3) false
(contains? (into-array [1 2 3]) -1) false)

; 'contains?' operates constant or logarithmic time,
; it WILL NOT perform a linear search for a value.
(are [x] (= x false)
 (contains? '(1 2 3) 0)
 (contains? '(1 2 3) 1)
 (contains? '(1 2 3) 3)
 (contains? '(1 2 3) 10)
 (contains? '(1 2 3) nil)
 (contains? '(1 2 3) ())))

(deftest test-keys
 (are [x y] (= x y) ; other than map data structures
 (keys ()) nil
 (keys []) nil
 (keys #{}) nil
 (keys "") nil)

 (are [x y] (= x y)
 ; (class {:a 1}) => clojure.lang.PersistentArrayMap
 (keys {:a 1}) '(:a)
 ; (keys {:a 1 :b 2}) '(:a :b)
 (diff (keys {:a 1 :b 2}) '(:a :b)) nil

 ; (class (sorted-map :a 1)) => clojure.lang.PersistentTreeMap
 (keys (sorted-map)) nil
 (keys (sorted-map :a 1)) '(:a)
 ; (keys (sorted-map :a 1 :b 2)) '(:a :b)
 (diff (keys (sorted-map :a 1 :b 2)) '(:a :b)) nil

 ; (class (hash-map :a 1)) => clojure.lang.PersistentHashMap
 (keys (hash-map)) nil
 (keys (hash-map :a 1)) '(:a)
 ; (keys (hash-map :a 1 :b 2)) '(:a :b)
 (diff (keys (hash-map :a 1 :b 2)) '(:a :b)) nil))

(deftest test-vals
 (are [x y] (= x y) ; other than map data structures
 (vals ()) nil
 (vals []) nil
 (vals #{}) nil
 (vals "") nil)

 (are [x y] (= x y)
)

```

```

; (class {:a 1}) => clojure.lang.PersistentArrayMap
(vals {}) nil
(vals {:a 1}) '(1)
; (vals {:a 1 :b 2}) '(1 2)
(diff (vals {:a 1 :b 2}) '(1 2)) nil

; (class (sorted-map :a 1)) => clojure.lang.PersistentTreeMap
(vals (sorted-map)) nil
(vals (sorted-map :a 1)) '(1)
; (vals (sorted-map :a 1 :b 2)) '(1 2)
(diff (vals (sorted-map :a 1 :b 2)) '(1 2)) nil

; (class (hash-map :a 1)) => clojure.lang.PersistentHashMap
(vals (hash-map)) nil
(vals (hash-map :a 1)) '(1)
; (vals (hash-map :a 1 :b 2)) '(1 2)
(diff (vals (hash-map :a 1 :b 2)) '(1 2)) nil)))

(deftest test-key
(are [x] (= (key (first (hash-map x :value))) x)
nil
false true
0 42
0.0 3.14
2/3
OM 1M
\c
"" "abc"
'sym
:kw
() '(1 2)
[] [1 2]
{} {:a 1 :b 2}
#{ } #{1 2})))

(deftest test-val
(are [x] (= (val (first (hash-map :key x))) x)
nil
false true
0 42
0.0 3.14
2/3
OM 1M
\c
"" "abc"
'sym
:kw
() '(1 2)

```

```

[] [1 2]
{} {:a 1 :b 2}
#{()} #{1 2})

(deftest test-get
 (let [m {:a 1, :b 2, :c {:d 3, :e 4}, :f nil, :g false, nil {:h 5}}]
 (is (thrown? IllegalArgumentException (get-in {:a 1} 5)))
 (are [x y] (= x y)
 (get m :a) 1
 (get m :e) nil
 (get m :e 0) 0
 (get m :b 0) 2
 (get m :f 0) nil

 (get-in m [:c :e]) 4
 (get-in m '(:c :e)) 4
 (get-in m [:c :x]) nil
 (get-in m [:f]) nil
 (get-in m [:g]) false
 (get-in m [:h]) nil
 (get-in m []) m
 (get-in m nil) m

 (get-in m [:c :e] 0) 4
 (get-in m '(:c :e) 0) 4
 (get-in m [:c :x] 0) 0
 (get-in m [:b] 0) 2
 (get-in m [:f] 0) nil
 (get-in m [:g] 0) false
 (get-in m [:h] 0) 0
 (get-in m [:x :y] {:y 1}) {:y 1}
 (get-in m [] 0) m
 (get-in m nil 0) m)))

;; *** Sets ***

(deftest test-hash-set
 (are [x] (set? x)
 #{}
 #{1 2}
 (hash-set)
 (hash-set 1 2))

 ; order isn't important
 (are [x y] (= x y)
 #{1 2} #{2 1}
 #{3 1 2} #{1 2 3}
 (hash-set 1 2) (hash-set 2 1)
 (hash-set 3 1 2) (hash-set 1 2 3))

```

```

(are [x y] (= x y)
 ; equal classes
 (class #{}) (class (hash-set))
 (class #{1 2}) (class (hash-set 1 2))

 ; creating
 (hash-set) #{}
 (hash-set 1) #{1}
 (hash-set 1 2) #{1 2}

 ; nesting
 (hash-set 1 (hash-set 2 3)
 (hash-set 3 (hash-set 4 5 (hash-set 6 (hash-set 7))))))
 #{1 #{2 3} #{3 #{4 5 #{6 #{7}}}}}

 ; different data structures
 (hash-set true false nil)
 #{true false nil}
 (hash-set 1 2.5 2/3 "ab" \x 'cd :kw)
 #{1 2.5 2/3 "ab" \x 'cd :kw}
 (hash-set (list 1 2) [3 4] {:a 1 :b 2} #{:c :d})
 #{'(1 2) [3 4] {:a 1 :b 2} #{:c :d} }

 ; evaluation
 (hash-set (+ 1 2) [(+ 2 3) :a] (hash-set (* 2 3) 8))
 #{3 [5 :a] #{6 8} }

 ; special cases
 (hash-set nil) #{nil}
 (hash-set 1 nil) #{1 nil}
 (hash-set nil 2) #{nil 2}
 (hash-set #{}) #{#{}}
 (hash-set 1 #{}) #{1 #{}}
 (hash-set #{} 2) #{#{2} 2})

(deftest test-sorted-set
 ; only compatible types can be used
 (is (thrown? ClassCastException (sorted-set 1 "a")))
 (is (thrown? ClassCastException (sorted-set '(1 2) [3 4])))

 ; creates set?
 (are [x] (set? x)
 (sorted-set)
 (sorted-set 1 2))

 ; equal and unique
 (are [x] (and (= (sorted-set x) #{x})
 (= (sorted-set x x) (sorted-set x))))

```

```

nil
false true
0 42
0.0 3.14
2/3
0M 1M
\c
"" "abc"
'sym
:kw
() ; '(1 2)
[] [1 2]
{} ; {:a 1 :b 2}
#{ } ; #{1 2}
)
; cannot be cast to java.lang.Comparable
(is (thrown? ClassCastException (sorted-set '(1 2) '(1 2))))
(is (thrown? ClassCastException (sorted-set {:a 1 :b 2} {:a 1 :b 2})))
(is (thrown? ClassCastException (sorted-set #{1 2} #{1 2})))

(are [x y] (= x y)
 ; generating
 (sorted-set) #{}
 (sorted-set 1) #{1}
 (sorted-set 1 2) #{1 2}

 ; sorting
 (seq (sorted-set 5 4 3 2 1)) '(1 2 3 4 5)

 ; special cases
 (sorted-set nil) #{nil}
 (sorted-set 1 nil) #{nil 1}
 (sorted-set nil 2) #{nil 2}
 (sorted-set #{}) #{#{}}))

(deftest test-sorted-set-by
 ; only compatible types can be used
 ; NB: not a ClassCastException, but a RuntimeException is thrown,
 ; requires discussion on whether this should be symmetric with
 ; test-sorted-set

 (is (thrown? Exception (sorted-set-by < 1 "a"))))
 (is (thrown? Exception (sorted-set-by < '(1 2) [3 4])))

 ; creates set?
 (are [x] (set? x)
 (sorted-set-by <)
 (sorted-set-by < 1 2))

```

```

; equal and unique
(are [x] (and (= (sorted-set-by compare x) #{x})
 (= (sorted-set-by compare x x)
 (sorted-set-by compare x)))
 nil
 false true
 0 42
 0.0 3.14
 2/3
 0M 1M
 \c
 "" "abc"
 'sym
 :kw
 () ; '(1 2)
 [] [1 2]
 {} ; {:a 1 :b 2}
 #{} ; #{1 2}
)
; cannot be cast to java.lang.Comparable
; NB: not a ClassCastException, but a RuntimeException is thrown,
; requires discussion on whether this should be symmetric with
; test-sorted-set

(is (thrown? Exception (sorted-set-by compare '(1 2) '(1 2))))
(is (thrown? Exception (sorted-set-by compare {:a 1 :b 2}
 {:a 1 :b 2})))
(is (thrown? Exception (sorted-set-by compare #{1 2} #{1 2})))

(are [x y] (= x y)
 ; generating
 (sorted-set-by >) #{}
 (sorted-set-by > 1) #{1}
 (sorted-set-by > 1 2) #{1 2}

 ; sorting
 (seq (sorted-set-by < 5 4 3 2 1)) '(1 2 3 4 5)

 ; special cases
 (sorted-set-by compare nil) #{nil}
 (sorted-set-by compare 1 nil) #{nil 1}
 (sorted-set-by compare nil 2) #{nil 2}
 (sorted-set-by compare #{}) #{#{}}))

(deftest test-set
 ; set?
 (are [x] (set? (set x))
 () '(1 2)
 [] [1 2])

```

```

#{}
{} #{1 2}
{} {:a 1 :b 2}
(into-array []) (into-array [1 2])
"" "abc")

; unique
(are [x] (= (set [x x]) #{x})
 nil
 false true
 0 42
 0.0 3.14
 2/3
 0M 1M
 \c
 "" "abc"
 'sym
 :kw
 () '(1 2)
 [] [1 2]
 {} {:a 1 :b 2}
 #{} #{1 2})

; conversion
(are [x y] (= (set x) y)
 () #{}
 '(1 2) #{1 2}

 [] #{}
 [1 2] #{1 2}

 #{} #{} ; identity
 #{1 2} #{1 2} ; identity

 {} #{}
 {:a 1 :b 2} #{[:a 1] [:b 2]}

 (into-array []) #{}
 (into-array [1 2]) #{1 2}

 "" #{}
 "abc" #{\a \b \c}))

(deftest test-disj
 ; doesn't work on lists, vectors or maps
 (is (thrown? ClassCastException (disj '(1 2) 1)))
 (is (thrown? ClassCastException (disj [1 2] 1)))
 (is (thrown? ClassCastException (disj {:a 1} :a)))

 ; identity

```

```
(are [x] (= (disj x) x)
 nil
 #{}
 #{1 2 3}
 ; different data types
 #{nil
 false true
 0 42
 0.0 3.14
 2/3
 0M 1M
 \c
 "" "abc"
 'sym
 :kw
 [] [1 2]
 {} {:a 1 :b 2}
 #{} #{1 2} })

; type identity
(are [x] (= (class (disj x)) (class x))
 (hash-set)
 (hash-set 1 2)
 (sorted-set)
 (sorted-set 1 2))

(are [x y] (= x y)
 (disj nil :a) nil
 (disj nil :a :b) nil

 (disj #{} :a) #{}
 (disj #{} :a :b) #{}

 (disj #{:a} :a) #{}
 (disj #{:a} :a :b) #{}
 (disj #{:a} :c) #{:a}

 (disj #{:a :b :c :d} :a) #{:b :c :d}
 (disj #{:a :b :c :d} :a :d) #{:b :c}
 (disj #{:a :b :c :d} :a :b :c) #{:d}
 (disj #{:a :b :c :d} :d :a :c :b) #{}

 (disj #{nil} :a) #{nil}
 (disj #{nil} #{}) #{nil}
 (disj #{nil} nil) {}

 (disj #{#{} nil} nil) #{#{}}
 (disj #{#{} nil} #{}) {}
 (disj #{#{nil}} #{nil}) #{{} }))
```

```
;; *** Queues ***

(deftest test-queues
 (let [EMPTY clojure.lang.PersistentQueue/EMPTY]
 (are [x y] (= x y)
 EMPTY EMPTY
 (into EMPTY (range 50)) (into EMPTY (range 50))
 (range 5) (into EMPTY (range 5))
 (range 1 6) (-> EMPTY
 (into (range 6))
 pop))
 (are [x y] (not= x y)
 (range 5) (into EMPTY (range 6))
 (range 6) (into EMPTY (range 5))
 (range 0 6) (-> EMPTY
 (into (range 6))
 pop)
 (range 1 6) (-> EMPTY
 (into (range 7))
 pop))))
```

---

### 12.13 test/def.clj

— test/def.clj —

```
\getchunk{Clojure Copyright}

(ns clojure.test-clojure.def
 (:use clojure.test clojure.test-helper
 clojure.test-clojure.protocols))

(deftest defn-error-messages
 (testing "bad arglist forms"
 (is (fails-with-cause? IllegalArgumentException
 #'#"Parameter declaration arg1 should be a vector"
 (eval-in-temp-ns (defn foo (arg1 arg2))))))

(deftest dynamic-redefinition
 ;; too many contextual things for this kind of caching to work...
 (testing "classes are never cached, even if their bodies are the same"
 (is (= :b
 (eval
 '(do
```

```
(defmacro my-macro [] :a)
(defn do-macro [] (my-macro))
(defmacro my-macro [] :b)
(defn do-macro [] (my-macro))
(do-macro))))))
```

---

## 12.14 test/errors.clj

— test/errors.clj —

```
\getchunk{Clojure Copyright}

;; Tests for error handling and messages

(ns clojure.test-clojure.errors
 (:use clojure.test)
 (:import clojure.lang.ArityException))

(defn f0 [] 0)

(defn f1 [a] a)

(defmacro m0 [] `(identity 0))

(defmacro m1 [a] `(inc ~a))

(deftest arity-exception
 ;; IllegalArgumentException is pre-1.3
 (is (thrown-with-msg? IllegalArgumentException
 #"Wrong number of args \\"1\\" passed to"
 (f0 1)))
 (is (thrown-with-msg? ArityException
 #"Wrong number of args \\"0\\" passed to"
 (f1)))
 (is (thrown-with-msg? ArityException
 #"Wrong number of args \\"1\\" passed to"
 (macroexpand '(m0 1))))
 (is (thrown-with-msg? ArityException
 #"Wrong number of args \\"2\\" passed to"
 (macroexpand '(m1 1 2)))))
```

---

## 12.15 test/evaluation.clj

— test/evaluation.clj —

```
\getchunk{Clojure Copyright}

;; Tests for the Clojure functions documented at the URL:
;;
;; http://clojure.org/Evaluation
;;
;; by J. McConnell
;; Created 22 October 2008

(ns clojure.test-clojure.evaluation
 (:use clojure.test))

(import '(java.lang Boolean)
 '(clojure.lang Compiler Compiler$CompilerException))

(defmacro test-that
 "Provides a useful way for specifying the purpose of tests. If the
 first-level forms are lists that make a call to a clojure.test
 function, it supplies the purpose as the msg argument to those
 functions. Otherwise, the purpose just acts like a comment and
 the forms are run unchanged."
 [purpose & test-forms]
 (let [tests (map
 #(if (= (:ns (meta (resolve (first %))))
 (the-ns 'clojure.test))
 (concat % (list purpose))
 %)
 test-forms)]
 `(do ~@tests)))

(deftest Eval
 (is (= (eval '(+ 1 2 3)) (Compiler/eval '(+ 1 2 3))))
 (is (= (eval '(list 1 2 3)) '(1 2 3)))
 (is (= (eval '(list + 1 2 3)) (list clojure.core/+ 1 2 3)))
 (test-that "Non-closure fns are supported as code"
 (is (= (eval (eval '(list + 1 2 3))) 6)))
 (is (= (eval (list '+ 1 2 3)) 6)))

; not using Clojure's RT/classForName since a bug in it could hide
; a bug in eval's resolution
(defn class-for-name [name]
 (java.lang.Class.forName name))

(defmacro in-test-ns [& body]
```

```

'(binding [*ns* *ns*]
 (in-ns 'clojure.test-clojure.evaluation)
 ~@body))

;;; Literals tests ;;

(defmacro #^{:private true} evaluates-to-itself? [expr]
 '(let [v# ~expr
 q# (quote ~expr)]
 (is (= (eval q#) q#) (str q# " does not evaluate to itself"))))

(deftest Literals
 ; Strings, numbers, characters, nil and keywords should evaluate
 ; to themselves
 (evaluates-to-itself? "test")
 (evaluates-to-itself? "test"
 multi-line
 string")
 (evaluates-to-itself? 1)
 (evaluates-to-itself? 1.0)
 (evaluates-to-itself? 1.123456789)
 (evaluates-to-itself? 1/2)
 (evaluates-to-itself? 1M)
 (evaluates-to-itself? 9999999999999999)
 (evaluates-to-itself? \a)
 (evaluates-to-itself? \newline)
 (evaluates-to-itself? nil)
 (evaluates-to-itself? :test)
 ; Boolean literals should evaluate to Boolean.{TRUE|FALSE}
 (is (identical? (eval true) Boolean/TRUE))
 (is (identical? (eval false) Boolean/FALSE)))

;;; Symbol resolution tests ;;

(def foo "abc")
(in-ns 'resolution-test)
(def bar 123)
(def #^{:private true} baz 456)
(in-ns 'clojure.test-clojure.evaluation)

(defn a-match? [re s] (not (nil? (re-matches re s)))))

(defmacro throws-with-msg
 ([re form] '(throws-with-msg ~re ~form Exception))
 ([re form x] '(throws-with-msg
 ~re
 ~form
 ~(if (instance? Exception x) x Exception)
 ~(if (instance? String x) x nil)))
 ([re form class msg]

```

```

'(let [ex# (try
 ^form
 (catch ^class e# e#)
 (catch Exception e#
 (let [cause# (.getCause e#)]
 (if (= ^class (class cause#)) cause# (throw e#))))]
 (is (a-match? ^re (.toString ex#))
 (or ^msg
 (str "Expected exception that matched " (pr-str ^re)
 ", but got exception with message: \\" ex#)))))

(deftest SymbolResolution
 (test-that
 "If a symbol is namespace-qualified, the evaluated value is the value
 of the binding of the global var named by the symbol"
 (is (= (eval 'resolution-test/bar) 123)))

 (test-that
 "It is an error if there is no global var named by the symbol"
 (throws-with-msg
 "#\".*Unable to resolve symbol: bar.*\" (eval 'bar)))"

 (test-that
 "It is an error if the symbol reference is to a non-public var in a
 different namespace"
 (throws-with-msg
 "#\".*resolution-test/baz is not public.*\""
 (eval 'resolution-test/baz)
 Compiler$CompilerException))

 (test-that
 "If a symbol is package-qualified, its value is the Java class
 named by the symbol"
 (is (= (eval 'java.lang.Math) (class-for-name "java.lang.Math"))))

 (test-that
 "If a symbol is package-qualified, it is an error if there is
 no Class named by the symbol"
 (is (thrown? Compiler$CompilerException (eval 'java.lang.FooBar)))))

 (test-that
 "If a symbol is not qualified, the following applies, in this order:
 1. If it names a special form it is considered a special form,
 and must be utilized accordingly.
 2. A lookup is done in the current namespace to see if there is
 a mapping from the symbol to a class. If so, the symbol is
 considered to name a Java class object.

```

3. If in a local scope (i.e. in a function definition), a lookup is done to see if it names a local binding (e.g. a function argument or let-bound name). If so, the value is the value of the local binding.
4. A lookup is done in the current namespace to see if there is a mapping from the symbol to a var. If so, the value is the value of the binding of the var referred-to by the symbol.
5. It is an error."

```

; First
(doall (for [form '(def if do let quote var fn loop recur throw try
 monitor-enter monitor-exit)]
 (is (thrown? Compiler$CompilerException (eval form)))))

(let [if "foo"]
 (is (thrown? Compiler$CompilerException (eval 'if)))))

; Second
(is (= (eval 'Boolean) (class-for-name "java.lang.Boolean")))
(let [Boolean "foo"]
 (is (= (eval 'Boolean) (class-for-name "java.lang.Boolean"))))

; Third
(is (= (eval '(let [foo "bar"] foo)) "bar"))

; Fourth
(in-test-ns (is (= (eval 'foo) "abc")))
(is (thrown? Compiler$CompilerException
 (eval 'bar))) ; not in this namespace

; Fifth
(is (thrown? Compiler$CompilerException (eval 'foobar)))))

;;; Metadata tests ;;

(defstruct struct-with-symbols (with-meta 'k {:a "A"}))

(deftest Metadata

 (test-that
 "find returns key symbols and their metadata"
 (let [s (struct struct-with-symbols 1)]
 (is (= {:a "A"} (meta (first (find s 'k)))))))

 ;;; Collections tests ;;
 (def x 1)
 (def y 2)

 (deftest Collections

```

```
(in-test-ns
 (test-that
 "Vectors and Maps yield vectors and (hash) maps whose contents
 are the evaluated values of the objects they contain."
 (is (= (eval '[x y z]) [1 2 3]))
 (is (= (eval '{:x x :y y :z z}) {:x 1 :y 2 :z 3}))
 (is (instance? clojure.lang.IPersistentMap (eval '{:x x :y y})))))

(in-test-ns
 (test-that
 "Metadata maps yield maps whose contents are the evaluated values
 of the objects they contain. If a vector or map has metadata, the
 evaluated metadata map will become the metadata of the resulting
 value."
 (is (= (eval #^{:x x} '[x y]) #^{:x 1} [1 2]))))

(test-that
 "An empty list () evaluates to an empty list."
 (is (= (eval '()) ()))
 (is (empty? (eval ()))))
 (is (= (eval (list)) ())))

;aargh, fragile tests, please fix
#_(test-that
 "Non-empty lists are considered calls"
 (is (thrown? Compiler$CompilerException (eval '(1 2 3)))))

(deftest Macros)

(deftest Loading)
```

---

## 12.16 test/for.clj

— test/for.clj —

```
\getchunk{Clojure Copyright}

;; Tests for the Clojure 'for' macro
;;
;; by Chouser
;; Created Dec 2008

(ns clojure.test-clojure.for
 (:use clojure.test))
```

```
(deftest Docstring-Example
 (is (= (take 100 (for [x (range 100000000)
 y (range 1000000) :while (< y x)]
 [x y]))
 '([1 0] [2 0] [2 1] [3 0] [3 1] [3 2] [4 0] [4 1] [4 2] [4 3]
 [5 0] [5 1] [5 2] [5 3] [5 4]
 [6 0] [6 1] [6 2] [6 3] [6 4] [6 5]
 [7 0] [7 1] [7 2] [7 3] [7 4] [7 5] [7 6]
 [8 0] [8 1] [8 2] [8 3] [8 4] [8 5] [8 6] [8 7]
 [9 0] [9 1] [9 2] [9 3] [9 4] [9 5] [9 6] [9 7] [9 8]
 [10 0] [10 1] [10 2] [10 3] [10 4] [10 5] [10 6] [10 7]
 [10 8] [10 9]
 [11 0] [11 1] [11 2] [11 3] [11 4] [11 5] [11 6] [11 7]
 [11 8] [11 9] [11 10]
 [12 0] [12 1] [12 2] [12 3] [12 4] [12 5] [12 6] [12 7]
 [12 8] [12 9] [12 10] [12 11]
 [13 0] [13 1] [13 2] [13 3] [13 4] [13 5] [13 6] [13 7]
 [13 8] [13 9] [13 10] [13 11] [13 12]
 [14 0] [14 1] [14 2] [14 3] [14 4] [14 5] [14 6] [14 7]
 [14 8]))))

(defmacro deftest-both [txt & ises]
 `(do
 (deftest ~(symbol (str "For-" txt)) `@ises)
 (deftest ~(symbol (str "Doseq-" txt))
 `@(map (fn [[x-is x-= [x-for binds body] value]]
 (when (and (= x-is 'is) (= x-= '=) (= x-for 'for))
 '(is (= (let [acc# (atom [])]
 (doseq `binds (swap! acc# conj `body))
 @acc#)
 `value))))
 ises)))))

(deftest-both When
 (is (= (for [x (range 10) :when (odd? x)] x) '(1 3 5 7 9)))
 (is (= (for [x (range 4) y (range 4) :when (odd? y)] [x y])
 '([0 1] [0 3] [1 1] [1 3] [2 1] [2 3] [3 1] [3 3])))
 (is (= (for [x (range 4) y (range 4) :when (odd? x)] [x y])
 '([1 0] [1 1] [1 2] [1 3] [3 0] [3 1] [3 2] [3 3])))
 (is (= (for [x (range 4) :when (odd? x) y (range 4)] [x y])
 '([1 0] [1 1] [1 2] [1 3] [3 0] [3 1] [3 2] [3 3])))
 (is (= (for [x (range 5) y (range 5) :when (< x y)] [x y])
 '([0 1] [0 2] [0 3] [0 4] [1 2] [1 3] [1 4] [2 3] [2 4] [3 4]))))

(defn only
 "Returns a lazy seq of increasing ints starting at 0. Trying to get
 the nth+1 value of the seq throws an exception. This is meant to
 help detecting over-eagerness in lazy seq consumers."
 [n]
 (lazy-cat (range n))
```

```

 (throw (Exception. "consumer went too far in lazy seq")))

(deftest-both While
 (is (= (for [x (only 6) :while (< x 5)] x) '(0 1 2 3 4)))
 (is (= (for [x (range 4) y (only 4) :while (< y 3)] [x y])
 '([0 0] [0 1] [0 2] [1 0] [1 1] [1 2]
 [2 0] [2 1] [2 2] [3 0] [3 1] [3 2])))
 (is (= (for [x (range 4) y (range 4) :while (< x 3)] [x y])
 '([0 0] [0 1] [0 2] [0 3] [1 0] [1 1] [1 2] [1 3]
 [2 0] [2 1] [2 2] [2 3])))
 (is (= (for [x (only 4) :while (< x 3) y (range 4)] [x y])
 '([0 0] [0 1] [0 2] [0 3] [1 0] [1 1] [1 2] [1 3]
 [2 0] [2 1] [2 2] [2 3])))
 (is (= (for [x (range 4) y (range 4) :while (even? x)] [x y])
 '([0 0] [0 1] [0 2] [0 3] [2 0] [2 1] [2 2] [2 3])))
 (is (= (for [x (only 2) :while (even? x) y (range 4)] [x y])
 '([0 0] [0 1] [0 2] [0 3])))
 (is (= (for [x (range 4) y (only 4) :while (< y x)] [x y])
 '([1 0] [2 0] [2 1] [3 0] [3 1] [3 2]))))

(deftest-both While-and-When
 (is
 (= (for [x (only 6) :while (< x 5) y (range 4) :when (odd? y)] [x y])
 '([0 1] [0 3] [1 1] [1 3] [2 1] [2 3] [3 1] [3 3] [4 1] [4 3])))
 (is
 (= (for [x (range 4) :when (odd? x) y (only 6) :while (< y 5)] [x y])
 '([1 0] [1 1] [1 2] [1 3] [1 4] [3 0] [3 1] [3 2] [3 3] [3 4])))
 (is
 (= (for [x (only 6) :while (< x 5) y (range 4) :when (odd? (+ x y))])
 '[x y])
 '([0 1] [0 3] [1 0] [1 2] [2 1] [2 3] [3 0] [3 2] [4 1] [4 3])))
 (is
 (= (for [x (range 4) :when (odd? x) y (only 2) :while (odd? (+ x y))])
 '[x y])
 '([1 0] [3 0]))))

(deftest-both While-and-When-Same-Binding
 (is (= (for [x (only 6) :while (< x 5) :when (odd? x)] x) '(1 3)))
 (is (= (for [x (only 6)
 :while (< x 5) ; if :while is false, :when should not be evald
 :when (do (if (< x 5) (odd? x)))] x) '(1 3)))
 (is (= (for [a (range -2 5)
 :when (not= a 0) ; :when may guard :while
 :while (> (Math/abs (/ 1.0 a)) 1/3)] a) '(-2 -1 1 2)))))

(deftest-both Nesting
 (is (= (for [x '(a b) y (interpose x '(1 2)) z (list x y)] [x y z])
 '([a 1 a] [a 1 1] [a a a] [a a a] [a 2 a] [a 2 2]
 [b 1 b] [b 1 1] [b b b] [b b b] [b 2 b] [b 2 2])))
 (is (= (for [x ['a nil] y [x 'b]] [x y])
 '([a nil] [a b]))))

```

```

'([a a] [a b] [nil nil] [nil b)))))

(deftest-both Destructuring
 (is (= (for [{:syms [a b c]}
 (map #(zipmap '(a b c) (range % 5)) (range 3))
 x [a b c]]
 (Integer. (str a b c x)))
 '(120 121 122 1231 1232 1233 2342 2343 2344)))

(deftest-both Let
 (is (= (for [x (range 3) y (range 3) :let [z (+ x y)]
 :when (odd? z)] [x y z])
 '([0 1 1] [1 0 1] [1 2 3] [2 1 3])))
 (is (= (for [x (range 6) :let [y (rem x 2)]
 :when (even? y) z [8 9]] [x z])
 '([0 8] [0 9] [2 8] [2 9] [4 8] [4 9])))

; :while must skip all subsequent chunks as well as the remainder of
; the current chunk:
(deftest-both Chunked-While
 (is (= (for [x (range 100) :while (even? x)] x) '(0))))
```

---

## 12.17 test/genclass.clj

— test/genclass.clj —

```
\getchunk{Clojure Copyright}

(ns ^{:doc "Tests for clojure.core/gen-class"
 :author "Stuart Halloway, Daniel Solano Gmez"}
 clojure.test-clojure.genclass
 (:use clojure.test clojure.test-helper)
 (:import [clojure.test_clojure.genclass.examples ExampleClass
 ExampleAnnotationClass]
 [java.lang.annotation ElementType
 Retention
 RetentionPolicy
 Target])))

(deftest arg-support
 (let [example (ExampleClass.)
 o (Object.)]
 (is (= "foo with o, o" (.foo example o o)))
 (is (= "foo with o, i" (.foo example o (int 1))))
 (is (thrown? java.lang.UnsupportedOperationException
```

```

(.foo example o)))))

(deftest name-munging
 (testing "mapping from Java fields to Clojure vars"
 (is (= #'clojure.test-clojure.genclass.examples/-foo-Object-int
 (get-field ExampleClass 'foo_Object_int_var)))
 (is (= #'clojure.test-clojure.genclass.examples/-toString
 (get-field ExampleClass 'toString_var)))))

;todo - fix this, it depends on the order of things out of a hash-map
#_(deftest test-annotations
 (let [annot-class ExampleAnnotationClass
 foo-method
 (.getDeclaredMethod annot-class "foo" (into-array [String]))]
 (testing "Class annotations:"
 (is (= 2 (count (.getDeclaredAnnotations annot-class)))))
 (testing "@Deprecated"
 (let [deprecated (.getAnnotation annot-class Deprecated)]
 (is deprecated)))
 (testing "@Target([])"
 (let [resource (.getAnnotation annot-class Target)]
 (is (= 0 (count (.value resource)))))))
 (testing "Method annotations:"
 (testing "@Deprecated void foo(String):"
 (is (= 1 (count (.getDeclaredAnnotations foo-method)))))
 (is (.getAnnotation foo-method Deprecated))))
 (testing "Parameter annotations:"
 (let [param-annots (.getParameterAnnotations foo-method)]
 (is (= 1 (alength param-annots)))
 (let [first-param-annots (aget param-annots 0)]
 (is (= 2 (alength first-param-annots)))
 (testing "void foo(@Retention() String)"
 (let [retention (aget first-param-annots 0)]
 (is (instance? Retention retention))
 (= RetentionPolicy/SOURCE (.value retention))))
 (testing "void foo(@Target() String)"
 (let [target (aget first-param-annots 1)]
 (is (instance? Target target))
 (is (= [ElementType/TYPE ElementType/PARAMETER]
 (seq (.value target))))))))
 (deftest genclass-option-validation
 (is (fails-with-cause? IllegalArgumentException
 #"(Not a valid method name: has-hyphen"
 (@#'clojure.core/validate-generate-class-options
 {:methods '[[fine [] void] [has-hyphen [] void]]})))))
 ——————

```

## 12.18 test/java`interop.clj

— test/java`interop.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka

(ns clojure.test-clojure.java-interop
 (:use clojure.test))

; http://clojure.org/java_interop
; http://clojure.org/compilation

(deftest test-dot
 ; (.instanceMember instance args*)
 (are [x] (= x "FRED")
 (.toUpperCase "fred")
 (. "fred" toUpperCase)
 (. "fred" (toUpperCase)))

 (are [x] (= x true)
 (.startsWith "abcde" "ab")
 (. "abcde" startsWith "ab")
 (. "abcde" (startsWith "ab")))

 ; (.instanceMember Classname args*)
 (are [x] (= x "java.lang.String")
 (.getName String)
 (. (identity String) getName)
 (. (identity String) (getName)))

 ; (Classname/staticMethod args*)
 (are [x] (= x 7)
 (Math/abs -7)
 (. Math abs -7)
 (. Math (abs -7)))

 ; Classname/staticField
 (are [x] (= x 2147483647)
 Integer/MAX_VALUE
 (. Integer MAX_VALUE)))

(deftest test-double-dot
 (is (= (.. System (getProperties) (get "os.name"))
```

```

(. (. System (getProperties)) (get "os.name"))))

(deftest test-doto
 (let [m (doto (new java.util.HashMap)
 (.put "a" 1)
 (.put "b" 2))]
 (are [x y] (= x y)
 (class m) java.util.HashMap
 m {"a" 1 "b" 2})))

(deftest test-new
 ; Integer
 (are [expr cls value] (and (= (class expr) cls)
 (= expr value))
 (new java.lang.Integer 42) java.lang.Integer 42
 (java.lang.Integer. 123) java.lang.Integer 123)

 ; Date
 (are [x] (= (class x) java.util.Date)
 (new java.util.Date)
 (java.util.Date.)))

(deftest test-instance?
 ; evaluation
 (are [x y] (= x y)
 (instance? java.lang.Integer (+ 1 2)) false
 (instance? java.lang.Long (+ 1 2)) true)

 ; different types
 (are [type literal] (instance? literal type)
 1 java.lang.Long
 1.0 java.lang.Double
 1M java.math.BigDecimal
 \a java.lang.Character
 "a" java.lang.String)

 ; it is a Long, nothing else
 (are [x y] (= (instance? x 42) y)
 java.lang.Integer false
 java.lang.Long true
 java.lang.Character false
 java.lang.String false)))

; set!
; memfn

```

```

(deftest test-bean
 (let [b (bean java.awt.Color/black)]
 (are [x y] (= x y)
 (map? b) true
 (:red b) 0
 (:green b) 0
 (:blue b) 0
 (:RGB b) -16777216
 (:alpha b) 255
 (:transparency b) 1
 (:class b) java.awt.Color)))

; proxy, proxy-super

(deftest test-proxy-chain
 (testing "That the proxy functions can chain"
 (are [x y] (= x y)
 (-> (get-proxy-class Object)
 construct-proxy
 (init-proxy {})
 (update-proxy {"toString" (fn [] "chain chain chain")})
 str)
 "chain chain chain"

 (-> (proxy [Object] [] (toString [] "superfuzz bigmuff"))
 (update-proxy {"toString" (fn [] "chain chain chain")})
 str)
 "chain chain chain")))

(deftest test-bases
 (are [x y] (= x y)
 (bases java.lang.Math)
 (list java.lang.Object)
 (bases java.lang.Integer)
 (list java.lang.Number java.lang.Comparable)))

(deftest test-supers
 (are [x y] (= x y)
 (supers java.lang.Math)
 #{java.lang.Object}
 (supers java.lang.Integer)
 #{java.lang.Number java.lang.Object
 java.lang.Comparable java.io.Serializable}))

```

```

; Arrays: [alength] aget aset [make-array to-array into-array
; to-array-2d aclone] [float-array, int-array, etc]
; amap, areduce

(defmacro deftest-type-array [type-array type]
 `'(deftest ~(symbol (str "test-" type-array))
 ; correct type
 #_(is (= (class (first (~type-array [1 2]))) (class (~type 1)))))

 ; given size (and empty)
 (are [x] (and (= (alength (~type-array x)) x)
 (= (vec (~type-array x)) (repeat x 0)))
 0 1 5)

 ; copy of a sequence
 (are [x] (and (= (alength (~type-array x)) (count x))
 (= (vec (~type-array x)) x))
 [] []
 [1]
 [1 -2 3 0 5])

 ; given size and init-value
 (are [x] (and (= (alength (~type-array x 42)) x)
 (= (vec (~type-array x 42)) (repeat x 42)))
 0 1 5)

 ; given size and init-seq
 (are [x y z] (and (= (alength (~type-array x y)) x)
 (= (vec (~type-array x y)) z))
 0 [] []
 0 [1] []
 0 [1 2 3] []
 1 [] [0]
 1 [1] [1]
 1 [1 2 3] [1]
 5 [] [0 0 0 0 0]
 5 [1] [1 0 0 0 0]
 5 [1 2 3] [1 2 3 0 0]
 5 [1 2 3 4 5] [1 2 3 4 5]
 5 [1 2 3 4 5 6 7] [1 2 3 4 5])))

(deftest-type-array int-array int)
(deftest-type-array long-array long)
;todo, fix, test broken for float/double, should compare to 1.0 2.0 etc
#_(deftest-type-array float-array float)
#_(deftest-type-array double-array double)

; separate test for exceptions (doesn't work with above macro...)

```

```
(deftest test-type-array-exceptions
 (are [x] (thrown? NegativeArraySizeException x)
 (int-array -1)
 (long-array -1)
 (float-array -1)
 (double-array -1)))

(deftest test-make-array
 ; negative size
 (is (thrown? NegativeArraySizeException (make-array Integer -1)))

 ; one-dimensional
 (are [x] (= (alength (make-array Integer x)) x)
 0 1 5)

 (let [a (make-array Long 5)]
 (aset a 3 42)
 (are [x y] (= x y)
 (aget a 3) 42
 (class (aget a 3)) Long))

 ; multi-dimensional
 (let [a (make-array Long 3 2 4)]
 (aset a 0 1 2 987)
 (are [x y] (= x y)
 (alength a) 3
 (alength (first a)) 2
 (alength (first (first a))) 4

 (aget a 0 1 2) 987
 (class (aget a 0 1 2)) Long)))

(deftest test-to-array
 (let [v [1 "abc" :kw \c []]
 a (to-array v)]
 (are [x y] (= x y)
 ; length
 (alength a) (count v)

 ; content
 (vec a) v
 (class (aget a 0)) (class (nth v 0))
 (class (aget a 1)) (class (nth v 1))
 (class (aget a 2)) (class (nth v 2))
 (class (aget a 3)) (class (nth v 3))
 (class (aget a 4)) (class (nth v 4)))

 ; different kinds of collections
```

```

(are [x] (and (= (alength (to-array x)) (count x))
 (= (vec (to-array x)) (vec x)))
 ())
 '(1 2)
 []
 [1 2]
 (sorted-set)
 (sorted-set 1 2)

 (int-array 0)
 (int-array [1 2 3])

 (to-array [])
 (to-array [1 2 3]))

(deftest test-into-array
 ; compatible types only
 (is (thrown? IllegalArgumentException
 (into-array [1 "abc" :kw])))
 (is (thrown? IllegalArgumentException
 (into-array [1.2 4])))
 (is (thrown? IllegalArgumentException
 (into-array [(byte 2) (short 3)])))
 (is (thrown? IllegalArgumentException
 (into-array Byte/TYPE [1000000000000000])))

 ; simple case
 (let [v [1 2 3 4 5]
 a (into-array v)]
 (are [x y] (= x y)
 (alength a) (count v)
 (vec a) v
 (class (first a)) (class (first v)))

 (is (= \a (aget (into-array Character/TYPE [\a \b \c]) 0))))

 (let [types [Integer/TYPE
 Byte/TYPE
 Float/TYPE
 Short/TYPE
 Double/TYPE
 Long/TYPE]
 values [(byte 2) (short 3) (int 4) 5]]
 (for [t types]
 (let [a (into-array t values)]
 (is (== (aget a 0) 2))
 (is (== (aget a 1) 3))
 (is (== (aget a 2) 4))
 (is (== (aget a 3) 5)))))

```

```

; different kinds of collections
(are [x] (and (= (alength (into-array x)) (count x))
 (= (vec (into-array x)) (vec x))
 (= (alength (into-array Long/TYPE x)) (count x))
 (= (vec (into-array Long/TYPE x)) (vec x))))
()
'(1 2)
[]
[1 2]
(sorted-set)
(sorted-set 1 2)

(int-array 0)
(int-array [1 2 3])

(to-array [])
(to-array [1 2 3]))

(deftest test-to-array-2d
; needs to be a collection of collection(s)
(is (thrown? Exception (to-array-2d [1 2 3])))

; ragged array
(let [v [[1] [2 3] [4 5 6]]
 a (to-array-2d v)]
 (are [x y] (= x y)
 (alength a) (count v)
 (alength (aget a 0)) (count (nth v 0))
 (alength (aget a 1)) (count (nth v 1))
 (alength (aget a 2)) (count (nth v 2))

 (vec (aget a 0)) (nth v 0)
 (vec (aget a 1)) (nth v 1)
 (vec (aget a 2)) (nth v 2)))

; empty array
(let [a (to-array-2d [])]
 (are [x y] (= x y)
 (alength a) 0
 (vec a) [])))

(deftest test-alength
(are [x] (= (alength x) 0)
 (int-array 0)
 (long-array 0)
 (float-array 0)
 (double-array 0))

```

```

(boolean-array 0)
(byte-array 0)
(char-array 0)
(short-array 0)
(make-array Integer/TYPE 0)
(to-array [])
(into-array [])
(to-array-2d []))

(are [x] (= (alength x) 1)
 (int-array 1)
 (long-array 1)
 (float-array 1)
 (double-array 1)
 (boolean-array 1)
 (byte-array 1)
 (char-array 1)
 (short-array 1)
 (make-array Integer/TYPE 1)
 (to-array [1])
 (into-array [1])
 (to-array-2d [[1]])))

(are [x] (= (alength x) 3)
 (int-array 3)
 (long-array 3)
 (float-array 3)
 (double-array 3)
 (boolean-array 3)
 (byte-array 3)
 (char-array 3)
 (short-array 3)
 (make-array Integer/TYPE 3)
 (to-array [1 "a" :k])
 (into-array [1 2 3])
 (to-array-2d [[1] [2 3] [4 5 6]])))

(deftest test-aclone
; clone all arrays except 2D
(are [x] (and (= (alength (aclone x)) (alength x))
 (= (vec (aclone x)) (vec x)))
 (int-array 0)
 (long-array 0)
 (float-array 0)
 (double-array 0)
 (boolean-array 0)
 (byte-array 0)
 (char-array 0)
 (short-array 0))
)

```

```

(make-array Integer/TYPE 0)
(to-array [])
(into-array [])

(int-array [1 2 3])
(long-array [1 2 3])
(float-array [1 2 3])
(double-array [1 2 3])
(boolean-array [true false])
(byte-array [(byte 1) (byte 2)])
(char-array [\a \b \c])
(short-array [(short 1) (short 2)])
(make-array Integer/TYPE 3)
(to-array [1 "a" :k])
(into-array [1 2 3]))

; clone 2D
(are [x] (and (= (alength (aclone x)) (alength x))
 (= (map alength (aclone x)) (map alength x)))
 (= (map vec (aclone x)) (map vec x)))
 (to-array-2d []))
 (to-array-2d [[1] [2 3] [4 5 6]]))

; Type Hints, *warn-on-reflection*
; #^ints, #^floats, #^longs, #^doubles

; Coercions: [int, long, float, double, char, boolean, short, byte]
; num
; ints/longs/floats/doubles

(deftest test-boolean
 (are [x y] (and (instance? java.lang.Boolean (boolean x))
 (= (boolean x) y)))
 nil false
 false false
 true true

 0 true
 1 true
 () true
 [1] true

 "" true
 \space true
 :kw true)))

(deftest test-char
 ; int -> char

```

```
(is (instance? java.lang.Character (char 65)))

; char -> char
(is (instance? java.lang.Character (char \a)))
(is (= (char \a) \a))

;; Note: More coercions in numbers.clj
```

---

## 12.19 test/keywords.clj

— test/keywords.clj —

```
\getchunk{Clojure Copyright}

(ns clojure.test-clojure.keywords
 (:use clojure.test))

(let [this-ns (str (.name *ns*))]
 (deftest test-find-keyword
 (defn- find-keyword [sym]
 (let [absent-keyword-sym (gensym "absent-keyword-sym")]
 (are [result lookup] (= result (find-keyword lookup))
 :foo :foo
 ::foo 'foo
 :foo "foo"
 nil absent-keyword-sym
 nil (str absent-keyword-sym))
 (are [result lookup] (= result (find-keyword this-ns lookup))
 ::foo "foo"
 nil (str absent-keyword-sym)))))
```

---

## 12.20 test/logic.clj

— test/logic.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka
```

```

;;
;; Created 1/29/2009

(ns clojure.test-clojure.logic
 (:use clojure.test
 [clojure.test-helper :only (exception)]))

;; *** Tests **

(deftest test-if
 ; true/false/nil
 (are [x y] (= x y)
 (if true :t) :t
 (if true :t :f) :t
 (if true :t (exception)) :t

 (if false :t) nil
 (if false :t :f) :f
 (if false (exception) :f) :f

 (if nil :t) nil
 (if nil :t :f) :f
 (if nil (exception) :f) :f)

 ; zero/empty is true
 (are [x] (= (if x :t :f) :t)
 (byte 0)
 (short 0)
 (int 0)
 (long 0)
 (bigint 0)
 (float 0)
 (double 0)
 (bigdec 0)

 0/2
 ""
 #"""
 (symbol ""))

 ()

 []
 {}
 #{}
 (into-array []))

 ; anything except nil/false is true
 (are [x] (= (if x :t :f) :t)
 (byte 2))

```

```
(short 2)
(int 2)
(long 2)
(bigint 2)
(float 2)
(double 2)
(bigdec 2)

2/3
\a
"abc"
#"a*b"
'abc
:kw

'(1 2)
[1 2]
{:a 1 :b 2}
#{1 2}
(into-array [1 2])

(new java.util.Date)))

(deftest test-nil-punning
 (are [x y] (= (if x :no :yes) y)
 (first []) :yes
 (next [1]) :yes
 (rest [1]) :no

 (butlast [1]) :yes

 (seq nil) :yes
 (seq []) :yes

 (sequence nil) :no
 (sequence []) :no

 (lazy-seq nil) :no
 (lazy-seq []) :no

 (filter #(> % 10) [1 2 3]) :no
 (map identity []) :no
 (apply concat []) :no

 (concat) :no
 (concat []) :no

 (reverse nil) :no
 (reverse []) :no)
```

```

(sort nil) :no
(sort []) :no)

(deftest test-and
 (are [x y] (= x y)
 (and) true
 (and true) true
 (and nil) nil
 (and false) false

 (and true nil) nil
 (and true false) false

 (and 1 true :kw 'abc "abc") "abc"

 (and 1 true :kw nil 'abc "abc") nil
 (and 1 true :kw nil (exception) 'abc "abc") nil

 (and 1 true :kw 'abc "abc" false) false
 (and 1 true :kw 'abc "abc" false (exception)) false)))

(deftest test-or
 (are [x y] (= x y)
 (or) nil
 (or true) true
 (or nil) nil
 (or false) false

 (or nil false true) true
 (or nil false 1 2) 1
 (or nil false "abc" :kw) "abc"

 (or false nil) nil
 (or nil false) false
 (or nil nil nil false) false

 (or nil true false) true
 (or nil true (exception) false) true
 (or nil false "abc" (exception)) "abc"))

(deftest test-not
; (is (thrown? IllegalArgumentException (not)))
 (are [x] (= (not x) true)
 nil
 false)
 (are [x] (= (not x) false)

```

```
true

; numbers
0
0.0
42
1.2
0/2
2/3

; characters
\space
\tab
\a

; strings
""
"abc"

; regexes
#""
#"a*b"

; symbols
(symbol "")
'abc

; keywords
:kw

; collections/arrays
()
'(1 2)
[]
[1 2]
{}
{:a 1 :b 2}
#{}
#{1 2}
(into-array [])
(into-array [1 2])

; Java objects
(new java.util.Date))
```

---

## 12.21 test/macros.clj

— test/macros.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka

(ns clojure.test-clojure.macros
 (:use clojure.test))

; http://clojure.org/macros

; ->
; defmacro definline macroexpand-1 macroexpand
```

---

## 12.22 test/main.clj

— test/main.clj —

```
\getchunk{Clojure Copyright}

; Author: Stuart Halloway

(ns clojure.test-clojure.main
 (:use clojure.test
 [clojure.test-helper :only [platform-newlines]])
 (:require [clojure.main :as main]))

(deftest eval-opt
 (testing "evals and prints forms"
 (is (= (platform-newlines "2\n4\n")
 (with-out-str (#'clojure.main/eval-opt "(+ 1 1) (+ 2 2)")))))

 (testing "skips printing nils"
 (is (= (platform-newlines ":a\n:c\n")
 (with-out-str (#'clojure.main/eval-opt ":a nil :c")))))

 (testing "does not block access to *in* (#299)"
 (with-in-str "(+ 1 1)"
 (is (= (platform-newlines "(+ 1 1)\n")
 (main/*in*/)))))
```

```
(with-out-str (#'clojure.main/eval-opt "(read))))))

(defmacro with-err-str
 "Evaluates exprs in a context in which *err* is bound to a fresh
StringWriter. Returns the string created by any nested printing
calls."
 [& body]
 '(let [s# (new java.io.StringWriter)
 p# (new java.io.PrintWriter s#)]
 (binding [*err* p#]
 ^@body
 (str s#)))

(defn run-repl-and-return-err
 "Run repl, swallowing stdout and returning stderr."
 [in-str]
 (with-err-str
 (with-out-str
 (with-in-str in-str
 (main/repl)))))

;argh - test fragility, please fix
#_(deftest repl-exception-safety
 (testing "catches and prints exception on bad equals"
 (is (re-matches #"(java\.lang\.NullPointerException\r?\n"
 (run-repl-and-return-err
 "(proxy [Object] [] (equals [o] (.toString nil))))")))))
```

---

## 12.23 test/metadata.clj

— test/metadata.clj —

```
\getchunk{Clojure Copyright}

; Authors: Stuart Halloway, Frantisek Sodomka

(ns clojure.test-clojure.metadata
 (:use clojure.test
 [clojure.test-helper :only (eval-in-temp-ns)]))

(def public-namespaces
 '[clojure.core
 clojure.pprint
 clojure.inspector
 clojure.set]
```

```

clojure.stacktrace
clojure.test
clojure.walk
clojure.xml
clojure.zip
clojure.java.io
clojure.java/browse
clojure.java.javadoc
clojure.java.shell
clojure.string
clojure.data])

(doseq [ns public-namespaces]
 (require ns))

(def public-vars
 (mapcat #(vals (ns-publics %)) public-namespaces))

(def public-vars-with-docstrings
 (filter (comp :doc meta) public-vars))

(deftest public-vars-with-docstrings-have-added
 (is (= [] (remove (comp :added meta) public-vars-with-docstrings)))

(deftest interaction-of-def-with-metadata
 (testing "initial def sets metadata"
 (let [v (eval-in-temp-ns
 (def ^{:a 1} foo 0)
 #'foo)]
 (is (= 1 (-> v meta :a)))))
 #_(testing "subsequent declare doesn't overwrite metadata"
 (let [v (eval-in-temp-ns
 (def ^{:b 2} bar 0)
 (declare bar)
 #'bar)]
 (is (= 2 (-> v meta :b))))
 (testing "when compiled"
 (let [v (eval-in-temp-ns
 (def ^{:c 3} bar 0)
 (defn declare-bar []
 (declare bar))
 (declare-bar)
 #'bar)]
 (is (= 3 (-> v meta :c)))))
 (testing "subsequent def with init-expr *does* overwrite metadata"
 (let [v (eval-in-temp-ns
 (def ^{:d 4} quux 0)
 (def quux 1)
 #'quux)])
 (is (nil? (-> v meta :d)))))))

```

```
(testing "when compiled"
 (let [v (eval-in-temp-ns
 (def ^{:e 5} quux 0)
 (defn def-quux []
 (def quux 1))
 (def-quux)
 #'quux)]
 (is (nil? (-> v meta :e)))))))
```

---

## 12.24 test/multimethods.clj

— test/multimethods.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka, Robert Lachlan

(ns clojure.test-clojure.multimethods
 (:use clojure.test [clojure.test-helper :only (with-var-roots)])
 (:require [clojure.set :as set]))

; http://clojure.org/multimethods

; defmulti
; defmethod
; remove-method
; prefer-method
; methods
; prefers

(defmacro for-all
 [& args]
 `(dorun (for ~@args)))

(defn hierarchy-tags
 "Return all tags in a derivation hierarchy"
 [h]
 (set/select
 #(instance? clojure.lang.Named %)
 (reduce into #{} (map keys (vals h)))))

(defn transitive-closure
 "Return all objects reachable by calling f starting with o,
 not including o itself. f should return a collection."
 [o f]
```

```

(loop [results #{}
 more #{o}]
 (let [new-objects (set/difference more results)]
 (if (seq new-objects)
 (recur (set/union results more)
 (reduce into #{} (map f new-objects)))
 (disj results o)))))

(defn tag-descendants
 "Set of descendants which are tags (i.e. Named)."
 [& args]
 (set/select
 #(instance? clojure.lang.Named %)
 (or (apply descendants args) #{})))

(defn assert-valid-hierarchy
 [h]
 (let [tags (hierarchy-tags h)]
 (testing "ancestors are the transitive closure of parents"
 (for-all [tag tags]
 (is (= (transitive-closure tag #(parents h %))
 (or (ancestors h tag) #{}))))
 (testing "ancestors are transitive"
 (for-all [tag tags]
 (is (= (transitive-closure tag #(ancestors h %))
 (or (ancestors h tag) #{}))))
 (testing "tag descendants are transitive"
 (for-all [tag tags]
 (is (= (transitive-closure tag #(tag-descendants h %))
 (or (tag-descendants h tag) #{}))))
 (testing "a tag isa? all of its parents"
 (for-all [tag tags
 :let [parents (parents h tag)]
 parent parents]
 (is (isa? h tag parent))))
 (testing "a tag isa? all of its ancestors"
 (for-all [tag tags
 :let [ancestors (ancestors h tag)]
 ancestor ancestors]
 (is (isa? h tag ancestor))))
 (testing "all my descendants have me as an ancestor"
 (for-all [tag tags
 :let [descendants (descendants h tag)]
 descendant descendants]
 (is (isa? h descendant tag))))
 (testing "there are no cycles in parents"
 (for-all [tag tags]
 (is (not (contains?
 (transitive-closure tag #(parents h %)) tag))))))
 (testing "there are no cycles in descendants"

```

```

(for-all [tag tags]
 (is (not (contains? (descendants h tag) tag)))))

(def family
 (reduce #(apply derive (cons %1 %2)) (make-hierarchy)
 [[:parent-1 ::ancestor-1]
 [:parent-1 ::ancestor-2]
 [:parent-2 ::ancestor-2]
 [:child ::parent-2]
 [:child ::parent-1]]))

;tpd this should be a regex, but i don't know how to split a regex
;(deftest cycles-are-forbidden
; (testing "a tag cannot be its own parent"
; (is (thrown-with-msg? Throwable #"^(not= tag parent\b)"
; (derive family ::child ::child))))
; (testing "a tag cannot be its own ancestor"
; (is
; (thrown-with-msg? Throwable
; (str "Cyclic derivation: :clojure.test-clojure.multimethods/child "
; "has :clojure.test-clojure.multimethods/ancestor-1 as ancestor"))
; (derive family ::ancestor-1 ::child)))))

(deftest using-diamond-inheritance
 (let [diamond (reduce #(apply derive (cons %1 %2)) (make-hierarchy)
 [[:mammal ::animal]
 [:bird ::animal]
 [:griffin ::mammal]
 [:griffin ::bird]])
 bird-no-more (underive diamond ::griffin ::bird)]
 (assert-valid-hierarchy diamond)
 (assert-valid-hierarchy bird-no-more)
 (testing "a griffin is a mammal, indirectly through mammal and bird"
 (is (isa? diamond ::griffin ::animal)))
 (testing "a griffin is a bird"
 (is (isa? diamond ::griffin ::bird)))
 (testing "after underive, griffin is no longer a bird"
 (is (not (isa? bird-no-more ::griffin ::bird))))
 (testing "but it is still an animal, via mammal"
 (is (isa? bird-no-more ::griffin ::animal)))))

(deftest derivation-world-bridges-to-java-inheritance
 (let [h (derive (make-hierarchy) java.util.Map ::map)]
 (testing "a Java class can be isa? a tag"
 (is (isa? h java.util.Map ::map)))
 (testing "if a Java class isa? a tag, so are its subclasses..."
 (is (isa? h java.util.HashMap ::map)))
 (testing "...but not its superclasses!"
 (is (not (isa? h java.util.Collection ::map))))))

```

```
(deftest global-hierarchy-test
 (with-var-roots #'clojure.core/global-hierarchy (make-hierarchy)}
 (assert-valid-hierarchy @#'clojure.core/global-hierarchy)
 (testing "when you add some derivations..."
 (derive ::lion ::cat)
 (derive ::manx ::cat)
 (assert-valid-hierarchy @#'clojure.core/global-hierarchy))
 (testing "...isa? sees the derivations"
 (is (isa? ::lion ::cat))
 (is (not (isa? ::cat ::lion))))
 (testing "... you can traverse the derivations"
 (is (= #::manx ::lion) (descendants ::cat)))
 (is (= #::cat) (parents ::manx)))
 (is (= #::cat) (ancestors ::manx)))
 (testing "then, remove a derivation..."
 (underive ::manx ::cat))
 (testing "... traversals update accordingly"
 (is (= #::lion) (descendants ::cat)))
 (is (nil? (parents ::manx)))
 (is (nil? (ancestors ::manx)))))

#_(defmacro for-all
 "Better than the actual for-all, if only it worked."
 [& args]
 `(reduce
 #(and %1 %2)
 (map true? (for ~@args))))
```

---

## 12.25 test/ns'libs.clj

— test/ns'libs.clj —

```
\getchunk{Clojure Copyright}

; Authors: Frantisek Sodomka, Stuart Halloway

(ns clojure.test-clojure.ns-libs
 (:use clojure.test))

; http://clojure.org/namespaces

; in-ns ns create-ns
; alias import intern refer
; all-ns find-ns
```

```

; ns-name ns-aliases ns-imports ns-interns ns-map ns-publics ns-refers
; resolve ns-resolve namespace
; ns-unalias ns-unmap remove-ns

; http://clojure.org/libs

; require use
; loaded-libs

(deftest test-alias
(is (thrown-with-msg? Exception
 #"(No namespace: epicfail found" (alias 'bogus 'epicfail)))))

(deftest test-require
(is (thrown? Exception (require :foo)))
(is (thrown? Exception (require)))))

(deftest test-use
(is (thrown? Exception (use :foo)))
(is (thrown? Exception (use)))))

(deftest reimporting-deftypes
(let [inst1 (binding [*ns* *ns*]
 (eval '(do (ns exporter)
 (defrecord ReimportMe [a])
 (ns importer)
 (import exporter.ReimportMe)
 (ReimportMe. 1))))
inst2 (binding [*ns* *ns*]
 (eval '(do (ns exporter)
 (defrecord ReimportMe [a b])
 (ns importer)
 (import exporter.ReimportMe)
 (ReimportMe. 1 2))))]
(testing "you can reimport a changed class and see the changes"
 (is (= [:a] (keys inst1)))
 (is (= [:a :b] (keys inst2))))
;fragile tests, please fix
#_(testing "you cannot import same local name from a different namespace"
 (is (thrown? clojure.lang.Compiler$CompilerException
 #"(ReimportMe already refers to: class exporter.ReimportMe
 in namespace: importer"
 (binding [*ns* *ns*]
 (eval '(do (ns exporter-2)
 (defrecord ReimportMe [a b])
 (ns importer)
 (import exporter-2.ReimportMe)
 (ReimportMe. 1 2))))))))
```

```
(deftest naming-types
 (testing
 "you cannot use a name already referred from another namespace"
 (is (thrown? IllegalStateException
 #'String already refers to: class java.lang.String
 (definterface String)))
 (is (thrown? IllegalStateException
 #'StringBuffer already refers to: class java.lang.StringBuffer"
 (deftype StringBuffer [])))
 (is (thrown? IllegalStateException
 #'Integer already refers to: class java.lang.Integer"
 (defrecord Integer [])))))

(deftest resolution
 (let [s (gensym)]
 (are [result expr] (= result expr)
 #'clojure.core/first (ns-resolve 'clojure.core 'first)
 nil (ns-resolve 'clojure.core s)
 nil (ns-resolve 'clojure.core {'first :local-first} 'first)
 nil (ns-resolve 'clojure.core {'first :local-first} s)))))

(deftest refer-error-messages
 (let [temp-ns (gensym)]
 (binding [*ns* *ns*]
 (in-ns temp-ns)
 (eval '(def ^{:private true} hidden-var)))
 (testing "referring to something that does not exist"
 (is (thrown-with-msg? IllegalStateException
 #'nonexistent-var does not exist"
 (refer temp-ns :only '(nonexistent-var))))))
 (testing "referring to something non-public"
 (is (thrown-with-msg? IllegalStateException
 #'hidden-var is not public"
 (refer temp-ns :only '(hidden-var)))))))
```

---

## 12.26 test/numbers.clj

— test/numbers.clj —

```
\getchunk{Clojure Copyright}

; Author: Stephen C. Gilardi
;; scgilardi (gmail)
;; Created 30 October 2008
;;
```

```
(ns clojure.test-clojure.numbers
 (:use clojure.test
 clojure.template))

; TODO:
; ==
; and more...

;; *** Types ***

(deftest Coerced-BigDecimal
 (let [v (bigdec 3)]
 (are [x] (true? x)
 (instance? BigDecimal v)
 (number? v)
 (decimal? v)
 (not (float? v)))))

(deftest BigInteger-conversions
 (are [x] (biginteger x)
 Long/MAX_VALUE
 13178456923875639284562345789M
 13178456923875639284562345789N))

(deftest unchecked-cast-num-obj
 (do-template [prim-array cast]
 (are [n]
 (let [a (prim-array 1)]
 (aset a 0 (cast n)))
 (Byte. Byte/MAX_VALUE)
 (Short. Short/MAX_VALUE)
 (Integer. Integer/MAX_VALUE)
 (Long. Long/MAX_VALUE)
 (Float. Float/MAX_VALUE)
 (Double. Double/MAX_VALUE))
 byte-array
 unchecked-byte
 short-array
 unchecked-short
 char-array
 unchecked-char
 int-array
 unchecked-int
 long-array
 unchecked-long
 float-array))
```

```

unchecked-float
double-array
unchecked-double))

(deftest unchecked-cast-num-prim
 (do-template [prim-array cast]
 (are [n]
 (let [a (prim-array 1)]
 (aset a 0 (cast n)))
 Byte/MAX_VALUE
 Short/MAX_VALUE
 Integer/MAX_VALUE
 Long/MAX_VALUE
 Float/MAX_VALUE
 Double/MAX_VALUE)
 byte-array
 unchecked-byte
 short-array
 unchecked-short
 char-array
 unchecked-char
 int-array
 unchecked-int
 long-array
 unchecked-long
 float-array
 unchecked-float
 double-array
 unchecked-double))

(deftest unchecked-cast-char
 ; in keeping with the checked cast functions, char and Character
 ; can only be cast to int
 (is (unchecked-int (char 0xFFFF)))
 (is (let [c (char 0xFFFF)] (unchecked-int c))) ; force primitive char

(def expected-casts
 [
 [:input [-1 0 1
 Byte/MAX_VALUE Short/MAX_VALUE Integer/MAX_VALUE Long/MAX_VALUE
 Float/MAX_VALUE Double/MAX_VALUE]]
 [char [:error (char 0) (char 1)
 (char 127) (char 32767) :error :error
 :error :error]]
 [unchecked-char [(char 65535) (char 0) (char 1) (char 127)
 (char 32767) (char 65535) (char 65535)
 (char 65535) (char 65535)]]]
 [byte [-1 0 1 Byte/MAX_VALUE
 :error :error :error :error
 :error :error]]]

```

```

[unchecked-byte [-1 0 1 Byte/MAX_VALUE
 -1 -1 -1 -1]
[short [-1 0 1 Byte/MAX_VALUE
 Short/MAX_VALUE :error :error :error
 :error]]
[unchecked-short [-1 0 1 Byte/MAX_VALUE
 Short/MAX_VALUE -1 -1 -1]
[int [-1 0 1 Byte/MAX_VALUE
 Short/MAX_VALUE Integer/MAX_VALUE :error :error
 :error]]
[unchecked-int [-1 0 1 Byte/MAX_VALUE
 Short/MAX_VALUE Integer/MAX_VALUE -1 Integer/MAX_VALUE
 Integer/MAX_VALUE]]
[long [-1 0 1 Byte/MAX_VALUE
 Short/MAX_VALUE Integer/MAX_VALUE Long/MAX_VALUE
 Long/MAX_VALUE Long/MAX_VALUE]]
[unchecked-long [-1 0 1 Byte/MAX_VALUE
 Short/MAX_VALUE Integer/MAX_VALUE Long/MAX_VALUE
 Long/MAX_VALUE Long/MAX_VALUE]]
;; 2.14748365E9 if when float/double conversion is avoided...
[float [-1.0 0.0 1.0 127.0
 32767.0 2.147483648E9 9.223372036854776E18
 Float/MAX_VALUE :error]]
[unchecked-float [-1.0 0.0 1.0 127.0
 32767.0 2.147483648E9 9.223372036854776E18
 Float/MAX_VALUE Float/POSITIVE_INFINITY]]
[double [-1.0 0.0 1.0 127.0
 32767.0 2.147483647E9 9.223372036854776E18
 Double/MAX_VALUE]]
[unchecked-double [-1.0 0.0 1.0 127.0
 32767.0 2.147483647E9 9.223372036854776E18
 Double/MAX_VALUE]]))

(deftest test-expected-casts
 (let [[[inputs] & expectations] expected-casts]
 (doseq [[f vals] expectations]
 (let [wrapped (fn [x]
 (try
 (f x)
 (catch IllegalArgumentException e :error)))]
 (is (= vals (map wrapped inputs)))))))

;; *** Functions ***
(defonce DELTA 1e-12)

(deftest test-add
 (are [x y] (= x y)

```

```

(+)
(+ 1)
(+ 1 2) 3
(+ 1 2 3) 6

(+ -1) -1
(+ -1 -2) -3
(+ -1 +2 -3) -2

(+ 1 -1) 0
(+ -1 1) 0

(+ 2/3) 2/3
(+ 2/3 1) 5/3
(+ 2/3 1/3) 1)

(are [x y] (< (- x y) DELTA)
 (+ 1.2) 1.2
 (+ 1.1 2.4) 3.5
 (+ 1.1 2.2 3.3) 6.6)

; no overflow
(is (> (+ Integer/MAX_VALUE 10) Integer/MAX_VALUE))
; no string concatenation
(is (thrown? ClassCastException (+ "ab" "cd")))

(deftest test-subtract
 (is (thrown? IllegalArgumentException (-)))
 (are [x y] (= x y)
 (- 1) -1
 (- 1 2) -1
 (- 1 2 3) -4

 (- -2) 2
 (- 1 -2) 3
 (- 1 -2 -3) 6

 (- 1 1) 0
 (- -1 -1) 0

 (- 2/3) -2/3
 (- 2/3 1) -1/3
 (- 2/3 1/3) 1/3)

 (are [x y] (< (- x y) DELTA)
 (- 1.2) -1.2
 (- 2.2 1.1) 1.1
 (- 6.6 2.2 1.1) 3.3)
)

```

```

; no underflow
(is (< (- Integer/MIN_VALUE 10) Integer/MIN_VALUE)))

(deftest test-multiply
 (are [x y] (= x y)
 (* 1)
 (* 2) 2
 (* 2 3) 6
 (* 2 3 4) 24

 (* -2) -2
 (* 2 -3) -6
 (* 2 -3 -1) 6

 (* 1/2) 1/2
 (* 1/2 1/3) 1/6
 (* 1/2 1/3 -1/4) -1/24)

 (are [x y] (< (- x y) DELTA)
 (* 1.2) 1.2
 (* 2.0 1.2) 2.4
 (* 3.5 2.0 1.2) 8.4)

; no overflow
(is (> (* 3 (int (/ Integer/MAX_VALUE 2.0))) Integer/MAX_VALUE)))

(deftest test-ratios-simplify-to-ints-where-appropriate
 (testing "negative denominator (assembly #275)"
 (is (integer? (/ 1 -1/2)))
 (is (integer? (/ 0 -1/2)))))

(deftest test-divide
 (are [x y] (= x y)
 (/ 1) 1
 (/ 2) 1/2
 (/ 3 2) 3/2
 (/ 4 2) 2
 (/ 24 3 2) 4
 (/ 24 3 2 -1) -4

 (/ -1) -1
 (/ -2) -1/2
 (/ -3 -2) 3/2
 (/ -4 -2) 2
 (/ -4 2) -2)

 (are [x y] (< (- x y) DELTA)
 (/ 4.5 3) 1.5
 (/ 4.5 3.0 3.0) 0.5)

```

```

(is (thrown? ArithmeticException (/ 0)))
(is (thrown? ArithmeticException (/ 2 0)))
(is (thrown? IllegalArgumentException (/))))

;; mod
;; http://en.wikipedia.org/wiki/Modulo_operation
;; http://mathforum.org/library/drmath/view/52343.html
;;
;; is mod correct?
;; http://groups.google.com/group/clojure/
;; browse_frm/thread/2a0ee4d248f3d131#
;;
;; Issue 23: mod (modulo) operator
;; http://code.google.com/p/clojure/issues/detail?id=23

(deftest test-mod
 ; wrong number of args
 ; (is (thrown? IllegalArgumentException (mod)))
 ; (is (thrown? IllegalArgumentException (mod 1)))
 ; (is (thrown? IllegalArgumentException (mod 3 2 1)))

 ; divide by zero
 (is (thrown? ArithmeticException (mod 9 0)))
 (is (thrown? ArithmeticException (mod 0 0)))

 (are [x y] (= x y)
 (mod 4 2) 0
 (mod 3 2) 1
 (mod 6 4) 2
 (mod 0 5) 0

 (mod 2 1/2) 0
 (mod 2/3 1/2) 1/6
 (mod 1 2/3) 1/3

 (mod 4.0 2.0) 0.0
 (mod 4.5 2.0) 0.5

 ; |num| > |div|, num != k * div
 (mod 42 5) 2 ; (42 / 5) * 5 + (42 mod 5) = 8 * 5 + 2 = 42
 (mod 42 -5) -3 ; (42 / -5) * (-5) + (42 mod -5)
 ; = -9 * (-5) + (-3) = 42
 (mod -42 5) 3 ; (-42 / 5) * 5 + (-42 mod 5) = -9 * 5 + 3
 ; = -42
 (mod -42 -5) -2 ; (-42 / -5) * (-5) + (-42 mod -5)
 ; = 8 * (-5) + (-2) = -42

 ; |num| > |div|, num = k * div
)
)

```

```

(mod 9 3) 0 ; (9 / 3) * 3 + (9 mod 3) = 3 * 3 + 0 = 9
(mod 9 -3) 0
(mod -9 3) 0
(mod -9 -3) 0

; |num| < |div|
(mod 2 5) 2 ; (2 / 5) * 5 + (2 mod 5)
; = 0 * 5 + 2 = 2
(mod 2 -5) -3 ; (2 / -5) * (-5) + (2 mod -5)
; = (-1) * (-5) + (-3) = 2
(mod -2 5) 3 ; (-2 / 5) * 5 + (-2 mod 5)
; = (-1) * 5 + 3 = -2
(mod -2 -5) -2 ; (-2 / -5) * (-5) + (-2 mod -5)
; = 0 * (-5) + (-2) = -2

; num = 0, div != 0
(mod 0 3) 0 ; (0 / 3) * 3 + (0 mod 3) = 0 * 3 + 0 = 0
(mod 0 -3) 0
)

;

;; rem & quot
;; http://en.wikipedia.org/wiki/Remainder

(deftest test-rem
 ; wrong number of args
 ; (is (thrown? IllegalArgumentException (rem)))
 ; (is (thrown? IllegalArgumentException (rem 1)))
 ; (is (thrown? IllegalArgumentException (rem 3 2 1)))

 ; divide by zero
 (is (thrown? ArithmeticException (rem 9 0)))
 (is (thrown? ArithmeticException (rem 0 0)))

 (are [x y] (= x y)
 (rem 4 2) 0
 (rem 3 2) 1
 (rem 6 4) 2
 (rem 0 5) 0

 (rem 2 1/2) 0
 (rem 2/3 1/2) 1/6
 (rem 1 2/3) 1/3

 (rem 4.0 2.0) 0.0
 (rem 4.5 2.0) 0.5

 ; |num| > |div|, num != k * div
 (rem 42 5) 2 ; (8 * 5) + 2 == 42
 (rem 42 -5) 2 ; (-8 * -5) + 2 == 42
)
)
```

```

(rem -42 5) -2 ; (-8 * 5) + -2 == -42
(rem -42 -5) -2 ; (8 * -5) + -2 == -42

; |num| > |div|, num = k * div
(rem 9 3) 0
(rem 9 -3) 0
(rem -9 3) 0
(rem -9 -3) 0

; |num| < |div|
(rem 2 5) 2
(rem 2 -5) 2
(rem -2 5) -2
(rem -2 -5) -2

; num = 0, div != 0
(rem 0 3) 0
(rem 0 -3) 0
)
)

(deftest test-quot
 ; wrong number of args
 ; (is (thrown? IllegalArgumentException (quot)))
 ; (is (thrown? IllegalArgumentException (quot 1))))
 ; (is (thrown? IllegalArgumentException (quot 3 2 1)))

 ; divide by zero
 (is (thrown? ArithmeticException (quot 9 0)))
 (is (thrown? ArithmeticException (quot 0 0)))

 (are [x y] (= x y)
 (quot 4 2) 2
 (quot 3 2) 1
 (quot 6 4) 1
 (quot 0 5) 0

 (quot 2 1/2) 4
 (quot 2/3 1/2) 1
 (quot 1 2/3) 1

 (quot 4.0 2.0) 2.0
 (quot 4.5 2.0) 2.0

 ; |num| > |div|, num != k * div
 (quot 42 5) 8 ; (8 * 5) + 2 == 42
 (quot 42 -5) -8 ; (-8 * -5) + 2 == 42
 (quot -42 5) -8 ; (-8 * 5) + -2 == -42
 (quot -42 -5) 8 ; (8 * -5) + -2 == -42
)
)
```

```

; |num| > |div|, num = k * div
(quot 9 3) 3
(quot 9 -3) -3
(quot -9 3) -3
(quot -9 -3) 3

; |num| < |div|
(quot 2 5) 0
(quot 2 -5) 0
(quot -2 5) 0
(quot -2 -5) 0

; num = 0, div != 0
(quot 0 3) 0
(quot 0 -3) 0
)

)

;; *** Predicates ***

;; pos? zero? neg?

(deftest test-pos?-zero?-neg?
 (let [nums [[[byte 2) (byte 0) (byte -2)]
 [(short 3) (short 0) (short -3)]
 [(int 4) (int 0) (int -4)]
 [(long 5) (long 0) (long -5)]
 [(bigint 6) (bigint 0) (bigint -6)]
 [(float 7) (float 0) (float -7)]
 [(double 8) (double 0) (double -8)]
 [(bigdec 9) (bigdec 0) (bigdec -9)]
 [2/3 0 -2/3]]]
 pred-result [[pos? [true false false]]
 [zero? [false true false]]
 [neg? [false false true]]]]
 (doseq [pr pred-result]
 (doseq [n nums]
 (is (= (map (first pr) n) (second pr))
 (pr-str (first pr) n)))))

;; even? odd?

(deftest test-even?
 (are [x] (true? x)
 (even? -4)
 (not (even? -3))
 (even? 0)
 (not (even? 5)))

```

```

(even? 8)
(is (thrown? ArithmeticException (even? 1/2)))
(is (thrown? ArithmeticException (even? (double 10)))))

(deftest test-odd?
 (are [x] (true? x)
 (not (odd? -4))
 (odd? -3)
 (not (odd? 0))
 (odd? 5)
 (not (odd? 8)))
 (is (thrown? ArithmeticException (odd? 1/2)))
 (is (thrown? ArithmeticException (odd? (double 10)))))

(defn- expt
 "clojure.contrib.math/expt is a better and much faster impl,
 but this works. Math/pow overflows to Infinity."
 [x n] (apply *' (replicate n x)))

(deftest test-bit-shift-left
 (are [x y] (= x y)
 2r10 (bit-shift-left 2r1 1)
 2r100 (bit-shift-left 2r1 2)
 2r1000 (bit-shift-left 2r1 3)
 2r00101110 (bit-shift-left 2r00010111 1)
 2r00101110 (apply bit-shift-left [2r00010111 1])
 2r01 (bit-shift-left 2r10 -1)
 (expt 2 32) (bit-shift-left 1 32)
 (expt 2N 10000) (bit-shift-left 1N 10000)
))
)

(deftest test-bit-shift-right
 (are [x y] (= x y)
 2r0 (bit-shift-right 2r1 1)
 2r010 (bit-shift-right 2r100 1)
 2r001 (bit-shift-right 2r100 2)
 2r000 (bit-shift-right 2r100 3)
 2r0001011 (bit-shift-right 2r00010111 1)
 2r0001011 (apply bit-shift-right [2r00010111 1])
 2r100 (bit-shift-right 2r10 -1)
 1 (bit-shift-right (expt 2 32) 32)
 1N (bit-shift-right (expt 2N 10000) 10000)
))
)

;; arrays
(deftest test-array-types
 (are [x y z] (= (Class.forName x) (class y) (class z))
 "[Z" (boolean-array 1) (booleans (boolean-array 1 true)))
 "[B" (byte-array 1) (bytes (byte-array 1 (byte 1))))
```

```

"[C" (char-array 1) (chars (char-array 1 \a))
"[S" (short-array 1) (shorts (short-array 1 (short 1)))
"[F" (float-array 1) (floats (float-array 1 1))
"[D" (double-array 1) (doubles (double-array 1 1))
"[I" (int-array 1) (ints (int-array 1 1))
"[J" (long-array 1) (longs (long-array 1 1)))

(deftest test-ratios
 (is (= (denominator 1/2) 2))
 (is (= (numerator 1/2) 1))
 (is (= (bigint (/ 10000000000000000000000000000000 3)) 3333333333333333333))
 (is (= (long 10000000000000000000000000000000/3) 3333333333333333333)))

```

---

## 12.27 test/other-functions.clj

— test/other-functions.clj —

```

\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka

(ns clojure.test-clojure.other-functions
 (:use clojure.test))

; http://clojure.org/other_functions

; [= not= (tests in data_structures.clj and elsewhere)]

(deftest test-identity
 ; exactly 1 argument needed
 ; (is (thrown? IllegalArgumentException (identity)))
 ; (is (thrown? IllegalArgumentException (identity 1 2)))

 (are [x] (= (identity x) x)
 nil
 false true
 0 42
 0.0 3.14
 2/3
 0M 1M
 \c

```

```

"" "abc"
'sym
:kw
() '(1 2)
[] [1 2]
{} {:a 1 :b 2}
#{ } #{1 2})

; evaluation
(are [x y] (= (identity x) y)
 (+ 1 2) 3
 (> 5 0) true))

(deftest test-name
 (are [x y] (= x (name y))
 "foo" :foo
 "bar" 'bar
 "quux" "quux"))

(deftest test-fnil
 (let [f1 (fnil vector :a)
 f2 (fnil vector :a :b)
 f3 (fnil vector :a :b :c)]
 (are [result input]
 (= result [(apply f1 input) (apply f2 input) (apply f3 input)])
 [[1 2 3 4] [1 2 3 4] [1 2 3 4]] [1 2 3 4]
 [[[a 2 3 4] [:a 2 3 4] [:a 2 3 4]] [nil 2 3 4]
 [[[a nil 3 4] [:a :b 3 4] [:a :b 3 4]] [nil nil 3 4]
 [[[a nil nil 4] [:a :b nil 4] [:a :b :c 4]] [nil nil nil 4]
 [[[a nil nil nil] [:a :b nil nil]
 [:a :b :c nil]] [nil nil nil nil]])
 (are [x y] (= x y)
 ((fnil + 0) nil 42) 42
 ((fnil conj []) nil 42) [42]
 (reduce #(update-in %1 [%2] (fnil inc 0)) {}
 ["fun" "counting" "words" "fun"])
 {"words" 1, "counting" 1, "fun" 2}
 (reduce #(update-in %1 [(first %2)] (fnil conj [])) (second %2) {}
 [[:a 1] [:a 2] [:b 3]])
 {[:b 3], :a [1 2]}))

; time assert comment doc

; partial
; comp

(deftest test-comp
 (let [c0 (comp)]
 (are [x] (= (identity x) (c0 x)))

```

```

nil
42
[1 2 3]
#{}
:foo)
(are [x y] (= (identity x) (c0 y))
(+ 1 2 3) 6
(keyword "foo") :foo)))

; complement
; constantly

; Printing
; pr prn print println newline
; pr-str prn-str print-str println-str [with-out-str (vars.cljs)]

; Regex Support
; re-matcher re-find re-matches re-groups re-seq

```

---

## 12.28 test/parallel.clj

— test/parallel.clj —

```

\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka

(ns clojure.test-clojure.parallel
 (:use clojure.test))

;; !! Tests for the parallel library will be in a separate
;; file closure_parallel.clj !!

; future-call
; future
; pmap
; pcalls
; pvalues

;; pmap
;;
(deftest pmap-does-its-thing

```

```
; ; regression fixed in r1218; was OutOfMemoryError
(is (= '(1) (pmap inc [0])))
```

—

## 12.29 test/pprint.clj

— test/pprint.clj —

```
\getchunk{Clojure Copyright}

;; Author: Tom Faulhaber

(ns clojure.test-clojure pprint
 (:refer-clojure :exclude [format])
 (:use [clojure.test :only (deftest are run-tests)]
 [clojure.test-helper :only [platform-newlines]]
 clojure.test-clojure pprint.test-helper
 clojure pprint))

;tpd too broken for me at the moment (load "pprint/test_cl_format")
(load "pprint/test.pretty")
```

—

## 12.30 test/predicates.clj

— test/predicates.clj —

```
\getchunk{Clojure Copyright}

;; Author: Frantisek Sodomka

;;
;; Created 1/28/2009

(ns clojure.test-clojure predicates
 (:use clojure.test))

;; *** Type predicates ***

(def myvar 42)
```

```
(def sample-data {
 :nil nil

 :bool-true true
 :bool-false false

 :byte (byte 7)
 :short (short 7)
 :int (int 7)
 :long (long 7)
 :bigint (bigint 7)
 :float (float 7)
 :double (double 7)
 :bigdec (bigdec 7)

 :ratio 2/3

 :character \a
 :symbol 'abc
 :keyword :kw

 :empty-string ""
 :empty-regex #""
 :empty-list ()
 :empty-lazy-seq (lazy-seq nil)
 :empty-vector []
 :empty-map {}
 :empty-set #{ }
 :empty-array (into-array [])

 :string "abc"
 :regex #"a*b"
 :list '(1 2 3)
 :lazy-seq (lazy-seq [1 2 3])
 :vector [1 2 3]
 :map {:a 1 :b 2 :c 3}
 :set #{1 2 3}
 :array (into-array [1 2 3])

 :fn (fn [x] (* 2 x))

 :class java.util.Date
 :object (new java.util.Date)

 :var (var myvar)
 :delay (delay (+ 1 2))
})
```

```
(def type-preds {
 nil? [:nil]

 true? [:bool-true]
 false? [:bool-false]
 ; boolean?

 integer? [:byte :short :int :long :bigint]
 float? [:float :double]
 decimal? [:bigdec]
 ratio? [:ratio]
 rational? [:byte :short :int :long :bigint :ratio :bigdec]
 number? [:byte :short :int :long :bigint :ratio :bigdec
 :float :double]

 ; character?
 symbol? [:symbol]
 keyword? [:keyword]

 string? [:empty-string :string]
 ; regex?

 list? [:empty-list :list]
 vector? [:empty-vector :vector]
 map? [:empty-map :map]
 set? [:empty-set :set]

 coll? [:empty-list :list
 :empty-lazy-seq :lazy-seq
 :empty-vector :vector
 :empty-map :map
 :empty-set :set]

 seq? [:empty-list :list
 :empty-lazy-seq :lazy-seq]
 ; array?

 fn? [:fn]
 ifn? [:fn
 :empty-vector :vector :empty-map :map :empty-set :set
 :keyword :symbol :var]

 class? [:class]
 var? [:var]
 delay? [:delay]
})

;; Test all type predicates against all data types
;;
```

```
(defn- get-fn-name [f]
 (str
 (apply str (nthnext (first (.split (str f) "_"))
 (count "clojure.core$")))
 "?"))

(deftest test-type-preds
 (doseq [tp type-preds]
 (doseq [dt sample-data]
 (if (some #(= % (first dt)) (second tp))
 (is ((first tp) (second dt)))
 (pr-str (list (get-fn-name (first tp)) (second dt))))
 (is (not ((first tp) (second dt)))
 (pr-str
 (list 'not
 (list (get-fn-name (first tp)) (second dt)))))))))

;; Additional tests:
;; http://groups.google.com/group/clojure/browse_thread/
;; thread/537761a06edb4b06/bfd4f0705b746a38
;;
(deftest test-string?-more
 (are [x] (not (string? x))
 (new java.lang.StringBuilder "abc")
 (new java.lang.StringBuffer "xyz")))
```

—→

### 12.31 test/printer.clj

— test/printer.clj —

```
\getchunk{Clojure Copyright}

; Author: Stephen C. Gilardi

;; clojure.test-clojure.printer
;;
;; scgilardi (gmail)
;; Created 29 October 2008

(ns clojure.test-clojure.printer
 (:use clojure.test))

(deftest print-length-empty-seq
 (let [coll () val "()"]
```

```

(is (= val (binding [*print-length* 0] (print-str coll))))
(is (= val (binding [*print-length* 1] (print-str coll)))))

(deftest print-length-seq
 (let [coll (range 5)
 length-val '((0 "(...)")
 (1 "(0 ...)")
 (2 "(0 1 ...)")
 (3 "(0 1 2 ...)")
 (4 "(0 1 2 3 ...)")
 (5 "(0 1 2 3 4)")]
 (doseq [[length val] length-val]
 (binding [*print-length* length]
 (is (= val (print-str coll))))))

(deftest print-length-empty-vec
 (let [coll [] val "[]"]
 (is (= val (binding [*print-length* 0] (print-str coll))))
 (is (= val (binding [*print-length* 1] (print-str coll)))))

(deftest print-length-vec
 (let [coll [0 1 2 3 4]
 length-val '((0 "[...]")
 (1 "[0 ...]")
 (2 "[0 1 ...]")
 (3 "[0 1 2 ...]")
 (4 "[0 1 2 3 ...]")
 (5 "[0 1 2 3 4]"))
 (doseq [[length val] length-val]
 (binding [*print-length* length]
 (is (= val (print-str coll))))))

(deftest print-level-seq
 (let [coll '(0 (1 (2 (3 (4)))))
 level-val '((0 "#")
 (1 "(0 #)")
 (2 "(0 (1 #))")
 (3 "(0 (1 (2 #)))")
 (4 "(0 (1 (2 (3 #))))")
 (5 "(0 (1 (2 (3 (4)))))))")]
 (doseq [[level val] level-val]
 (binding [*print-level* level]
 (is (= val (print-str coll))))))

(deftest print-level-length-coll
 (let [coll '(if (member x y) (+ (first x) 3) (foo (a b c d "Baz")))
 level-length-val
 '((0 1 "#")
 (1 1 "(if ...)")
 (1 2 "(if # ...)")))
 (doseq [[level val] level-length-val]
 (binding [*print-level* level]
 (is (= val (print-str coll)))))))

```

```
(1 3 "(if # # ...)")
(1 4 "(if # # #)")
(2 1 "(if ...)")
(2 2 "(if (member x ...) ...)")
(2 3 "(if (member x y) (+ # 3) ...)")
(3 2 "(if (member x ...) ...)")
(3 3 "(if (member x y) (+ (first x) 3) ...)")
(3 4 "(if (member x y) (+ (first x) 3) (foo (a b c d ...))))")
(3 5 "(if (member x y) (+ (first x) 3) (foo (a b c d Baz))))"))
(doseq [[level length val] level-length-val]
 (binding [*print-level* level
 print-length length]
 (is (= val (print-str coll))))))

```

---

## 12.32 test/protocols.clj

— test/protocols.clj —

```
\getchunk{Clojure Copyright}

; Author: Stuart Halloway

(ns clojure.test-clojure.protocols
 (:use clojure.test clojure.test-clojure.protocols.examples)
 (:require [clojure.test-clojure.protocols.more-examples :as other]
 [clojure.set :as set]
 [clojure.test-helper])
 (:import [clojure.test_clojure.protocols.examples ExampleInterface]))

;; temporary hack until I decide how to cleanly reload protocol
;; this no longer works
(defn reload-example-protocols
 []
 (alter-var-root #'clojure.test-clojure.protocols.examples/ExampleProtocol
 assoc :impls {})
 (alter-var-root #'clojure.test-clojure.protocols.more-examples/SimpleProtocol
 assoc :impls {})
 (require :reload
 'clojure.test-clojure.protocols.examples
 'clojure.test-clojure.protocols.more-examples))

(defn method-names
 "return sorted list of method names on a class"
```

```

[c]
(->> (.getMethods c)
 (map #(.getName %))
 (sort)))

(defrecord EmptyRecord [])
(defrecord TestRecord [a b])
(defn r
 ([a b] (TestRecord. a b))
 ([a b meta ext] (TestRecord. a b meta ext)))
(defrecord MapEntry [k v]
 java.util.Map$Entry
 (getKey [_] k)
 (getValue [_] v))

;tpd i broke this by splitting the regex
(deftest protocols-test
 (testing "protocol fns have useful metadata"
 (let
 [common-meta
 {:ns (find-ns 'clojure.test-clojure.protocols.examples)
 :protocol #'ExampleProtocol}]
 (are [m f] (= (merge (quote m) common-meta)
 (meta (var f)))
 {:name foo :arglists ([a]) :doc "method with one arg"} foo
 {:name bar :arglists ([a b]) :doc "method with two args"} bar
 {:name baz :arglists ([a] [a b])
 :doc "method with multiple arities" :tag String} baz
 {:name with-quux :arglists ([a])
 :doc "method name with a hyphen"} with-quux)))
 ; (testing
 ; "protocol fns throw IllegalArgumentException if no impl matches"
 ; (is (thrown-with-msg?
 ; IllegalArgumentException
 ; (str "No implementation of method: :foo of protocol: "
 ; "#'clojure.test-clojure.protocols.examples/ExampleProtocol "
 ; "found for class: java.lang.Long")
 ; (foo 10))))
 ; (testing "#protocols generate a corresponding interface using -
 ; instead of - for method names"
 ; (is (= ["bar" "baz" "baz" "foo" "with_quux"]
 ; (method-names
 ; clojure.test_clojure.protocols.examples.ExampleProtocol))))
 ; (testing
 ; "#protocol will work with instances of its interface
 ; (use for interop, not in Clojure!)"
 ; (let [obj (proxy
 ; [clojure.test_clojure.protocols.examples.ExampleProtocol] []
 ; (foo [] "foo!"))]
 ; (is (= "foo!" (.foo obj)) "call through interface"))

```

```

; (is (= "foo!" (foo obj)) "call through protocol")))
(testing "you can implement just part of a protocol if you want"
 (let [obj (reify ExampleProtocol
 (baz [a b] "two-arg baz!")]
 (is (= "two-arg baz!" (baz obj nil)))
 (is (thrown? AbstractMethodError (baz obj)))))

; (testing "you can redefine a protocol with different methods"
; (eval '(defprotocol Elusive (old-method [x])))
; (eval '(defprotocol Elusive (new-method [x])))
; (is (= :new-method
; (eval
; '(new-method (reify Elusive (new-method [x] :new-method)))))))
; (is (fails-with-cause? IllegalArgumentException
; (str "No method of interface: user.Elusive found for function: "
; "old-method of protocol: Elusive (The protocol method may "
; "have been defined before and removed.)"))
; (eval
; '(old-method
; (reify Elusive (new-method [x] :new-method)))))))
)

(deftype ExtendTestWidget [name])
(deftype HasProtocolInline []
 ExampleProtocol
 (foo [this] :inline))
(deftest extend-test
 (testing "you can extend a protocol to a class"
 (extend String ExampleProtocol
 {:foo identity})
 (is (= "pow" (foo "pow"))))

 (testing "#'you can have two methods with the same name.
Just use namespaces!"
 (extend String other/SimpleProtocol
 {:foo (fn [s] (.toUpperCase s))})
 (is (= "POW" (other/foo "pow"))))

 (testing "you can extend deftype types"
 (extend
 ExtendTestWidget
 ExampleProtocol
 {:foo (fn [this] (str "widget " (.name this)))})
 (is (= "widget z" (foo (ExtendTestWidget. "z")))))

;tpd i broke this because I split the regex
;(deftest illegal-extending
; (testing "#'you cannot extend a protocol to a type that
; implements the protocol inline"
; (is (fails-with-cause? IllegalArgumentException
; #".*HasProtocolInline already directly implements interface"
; (eval
; '(extend clojure.test_clojure.protocols.HasProtocolInline
;
```

```

; clojure.test-clojure.protocols.examples/ExampleProtocol
; {:foo (fn [_] :extended)})))))
; (testing "you cannot extend to an interface"
; (is (fails-with-cause? IllegalArgumentException
; (str "interface "
; "clojure.test_clojure.protocols.examples.ExampleProtocol "
; "is not a protocol")
; (eval '(extend clojure.test_clojure.protocols.HasProtocolInline
; clojure.test_clojure.protocols.examples.ExampleProtocol
; {:foo (fn [_] :extended)}))))))

(deftype ExtendsTestWidget []
 ExampleProtocol)
#_(deftest extends?-test
 (reload-example-protocols)
 (testing
 "returns false if a type does not implement the protocol at all"
 (is (false? (extends? other/SimpleProtocol ExtendsTestWidget))))
 ;; semantics changed 4/15/2010
 (testing "returns true if a type implements the protocol directly"
 (is (true? (extends? ExampleProtocol ExtendsTestWidget))))
 (testing "returns true if a type explicitly extends protocol"
 (extend
 ExtendsTestWidget
 other/SimpleProtocol
 {:foo identity})
 (is (true? (extends? other/SimpleProtocol ExtendsTestWidget)))))

(deftype ExtendersTestWidget [])
#_(deftest extender-test
 (reload-example-protocols)
 (testing "a fresh protocol has no extenders"
 (is (nil? (extenders ExampleProtocol))))
 (testing "extending with no methods doesn't count!"
 (deftype Something [])
 (extend ::Something ExampleProtocol)
 (is (nil? (extenders ExampleProtocol)))))
 (testing
 "#extending a protocol (and including an impl)
 adds an entry to extenders"
 (extend ExtendersTestWidget ExampleProtocol {:foo identity})
 (is (= [ExtendersTestWidget] (extenders ExampleProtocol)))))

(deftype SatisfiesTestWidget []
 ExampleProtocol)
#_(deftest satisfies?-test
 (reload-example-protocols)
 (let [whatzit (SatisfiesTestWidget.)]
 (testing "#returns false if a type does not
 implement the protocol at all"

```

```

(is (false? (satisfies? other/SimpleProtocol whatzit))))
(testing "returns true if a type implements the protocol directly"
 (is (true? (satisfies? ExampleProtocol whatzit))))
(testing "returns true if a type explicitly extends protocol"
 (extend
 SatisfiesTestWidget
 other/SimpleProtocol
 {:foo identity})
 (is (true? (satisfies? other/SimpleProtocol whatzit)))))

(deftype ReExtendingTestWidget [])
#_(deftest re-extending-test
 (reload-example-protocols)
 (extend
 ReExtendingTestWidget
 ExampleProtocol
 {:foo (fn [_] "first foo")
 :baz (fn [_] "first baz")})
 (testing "#if you re-extend, the old implementation
 is replaced (not merged!)"
 (extend
 ReExtendingTestWidget
 ExampleProtocol
 {:baz (fn [_] "second baz")
 :bar (fn [_ _] "second bar")})
 (let [whatzit (ReExtendingTestWidget.)]
 (is (thrown? IllegalArgumentException (foo whatzit)))
 (is (= "second bar" (bar whatzit nil)))
 (is (= "second baz" (baz whatzit))))))

(defrecord DefrecordObjectMethodsWidgetA [a])
(defrecord DefrecordObjectMethodsWidgetB [a])
(deftest defrecord-object-methods-test
 (testing "= depends on fields and type"
 (is (true?
 (= (DefrecordObjectMethodsWidgetA. 1)
 (DefrecordObjectMethodsWidgetA. 1))))
 (is (false?
 (= (DefrecordObjectMethodsWidgetA. 1)
 (DefrecordObjectMethodsWidgetA. 2))))
 (is (false?
 (= (DefrecordObjectMethodsWidgetA. 1)
 (DefrecordObjectMethodsWidgetB. 1))))))

(deftest defrecord-acts-like-a-map
 (let [rec (r 1 2)]
 (is (.equals (r 1 3 {}) {:c 4}) (merge rec {:b 3 :c 4})))
 (is (.equals {:foo 1 :b 2} (set/rename-keys rec {:a :foo})))
 (is (.equals {:a 11 :b 2 :c 10} (merge-with + rec {:a 10 :c 10})))))

```

```
(deftest degenerate-defrecord-test
 (let [empty (EmptyRecord.)]
 (is (nil? (seq empty)))
 (is (not (.containsValue empty :a)))))

(deftest defrecord-interfaces-test
 (testing "java.util.Map"
 (let [rec (r 1 2)]
 (is (= 2 (.size rec)))
 (is (= 3 (.size (assoc rec :c 3))))
 (is (not (.isEmpty rec)))
 (is (.isEmpty (EmptyRecord.)))
 (is (.containsKey rec :a))
 (is (not (.containsKey rec :c)))
 (is (.containsValue rec 1))
 (is (not (.containsValue rec 3)))
 (is (= 1 (.get rec :a)))
 (is (thrown? UnsupportedOperationException (.put rec :a 1)))
 (is (thrown? UnsupportedOperationException (.remove rec :a)))
 (is (thrown? UnsupportedOperationException (.putAll rec {})))
 (is (thrown? UnsupportedOperationException (.clear rec)))
 (is (= #{:a :b} (.keySet rec)))
 (is (= #{1 2} (set (.values rec))))
 (is (= #{{:a 1} {:b 2}} (.entrySet rec)))

)))
 (testing "IPersistentCollection"
 (testing ".cons"
 (let [rec (r 1 2)]
 (are [x] (= rec (.cons rec x))
 nil {})
 (is (= (r 1 3) (.cons rec {:b 3})))
 (is (= (r 1 4) (.cons rec [:b 4])))
 (is (= (r 1 5) (.cons rec (MapEntry. :b 5)))))))

(defrecord RecordWithSpecificFieldNames [this that k m o])
(deftest defrecord-with-specific-field-names
 (let [rec (new RecordWithSpecificFieldNames 1 2 3 4 5)]
 (is (= rec rec))
 (is (= 1 (:this (with-meta rec {:foo :bar}))))
 (is (= 3 (get rec :k)))
 (is (= (seq rec) '(:this 1) [:that 2] [:k 3] [:m 4] [:o 5])))
 (is (= (dissoc rec :k) {:this 1, :that 2, :m 4, :o 5}))))

(deftest reify-test
 (testing "of an interface"
 (let [s :foo
 r (reify
 java.util.List
 (contains [_ o] (= s o)))]

```

```

(testing "implemented methods"
 (is (true? (.contains r :foo)))
 (is (false? (.contains r :bar))))
(testing "unimplemented methods"
 (is (thrown? AbstractMethodError (.add r :baz)))))

(testing "of two interfaces"
 (let [r (reify
 java.util.List
 (contains [_ o] (= :foo o))
 java.util.Collection
 (isEmpty [_] false)])
 (is (true? (.contains r :foo)))
 (is (false? (.contains r :bar)))
 (is (false? (.isEmpty r)))))

(testing "you can't define a method twice"
 (is (fails-with-cause?
 java.lang.ClassFormatError
 #"(Repetitive|Duplicate) method name"
 (eval '(reify
 java.util.List
 (size [_] 10)
 java.util.Collection
 (size [_] 20))))))

(testing "you can't define a method not on an interface/protocol/j.l.Object"
 (is (fails-with-cause?
 java.lang.IllegalArgumentException
 #"(Can't define method not in interfaces: foo"
 (eval '(reify java.util.List (foo []))))))

(testing "of a protocol"
 (let [r (reify
 ExampleProtocol
 (bar [this o] o)
 (baz [this] 1)
 (baz [this o] 2)])
 (= :foo (.bar r :foo))
 (= 1 (.baz r))
 (= 2 (.baz r nil)))))

(testing "destructuring in method def"
 (let [r (reify
 ExampleProtocol
 (bar [this [_ _ item]] item)])
 (= :c (.bar r [:a :b :c]))))

(testing "methods can recur"
 (let [r (reify
 java.util.List
 (get [_ index]
 (if (zero? index)
 :done
 (recur (dec index))))])

```

```

(is (= :done (.get r 0)))
(is (= :done (.get r 1))))
(testing "disambiguating with type hints"
 (testing "you must hint an overloaded method"
 (is (fails-with-cause?
 IllegalArgumentException
 #"Must hint overloaded method: hinted"
 (eval `(reify
 clojure.test_clojure.protocols.examples.ExampleInterface
 (hinted [_ o])))))
 (testing "hinting"
 (let [r (reify
 ExampleInterface
 (hinted [_ ^int i] (inc i))
 (hinted [_ ^String s] (str s s)))]
 (is (= 2 (.hinted r 1)))
 (is (= "xoxo" (.hinted r "xo")))))))

```

---

## 12.33 test/reader.clj

— test/reader.clj —

```

\getchunk{Clojure Copyright}

; Author: Stephen C. Gilardi

;;
;; Tests for the Clojure functions documented at the URL:
;;
;; http://clojure.org/Reader
;;
;; scgilardi (gmail)
;; Created 22 October 2008

(ns clojure.test-clojure.reader
 (:use clojure.test)
 (:import clojure.lang.BigInt))

;; Symbols

(deftest Symbols
 (is (= 'abc (symbol "abc")))
 (is (= '*+!-_? (symbol "*+!-_?")))
 (is (= 'abc:def:ghi (symbol "abc:def:ghi"))))

```

```

(is (= 'abc/def (symbol "abc" "def")))
(is (= 'abc.def/ghi (symbol "abc.def" "ghi")))
(is (= 'abc/def.ghi (symbol "abc" "def.ghi")))
(is (= 'abc:def/ghi:jkl.mno (symbol "abc:def" "ghi:jkl.mno")))
(is (instance? clojure.lang.Symbol 'alphabet))
)

;; Literals

(deftest Literals
; 'nil 'false 'true are reserved by Clojure and are not symbols
(is (= 'nil nil))
(is (= 'false false))
(is (= 'true true)))

;; Strings

(deftest Strings
(is (= "abcde" (str \a \b \c \d \e)))
(is (= "abc
def" (str \a \b \c \newline \space \space \d \e \f)))
)

;; Numbers

(deftest Numbers

; Read Integer
(is (instance? Long 2147483647))
(is (instance? Long +1))
(is (instance? Long 1))
(is (instance? Long +0))
(is (instance? Long 0))
(is (instance? Long -0))
(is (instance? Long -1))
(is (instance? Long -2147483648))

; Read Long
(is (instance? Long 2147483648))
(is (instance? Long -2147483649))
(is (instance? Long 9223372036854775807))
(is (instance? Long -9223372036854775808))

;; Numeric constants of different types don't wash out. Regression
;; fixed in r1157. Previously the compiler saw 0 and 0.0 as the same
;; constant and caused the sequence to be built of Doubles.
(let [x 0.0]
(let [sequence (loop [i 0 1 '()]
(if (< i 5)
(recur (inc i) (conj 1 i)))

```

```
1))]
(is (= [4 3 2 1 0] sequence))
(is (every? #(instance? Long %)
 sequence))))

; Read BigInteger
(is (instance? BigInt 9223372036854775808))
(is (instance? BigInt -9223372036854775809))
(is (instance? BigInt
 1000))
(is (instance? BigInt
 -100))

; Read Double
(is (instance? Double +1.0e+1))
(is (instance? Double +1.e+1))
(is (instance? Double +1e+1))

(is (instance? Double +1.0e1))
(is (instance? Double +1.e1))
(is (instance? Double +1e1))

(is (instance? Double +1.0e-1))
(is (instance? Double +1.e-1))
(is (instance? Double +1e-1))

(is (instance? Double 1.0e+1))
(is (instance? Double 1.e+1))
(is (instance? Double 1e+1))

(is (instance? Double 1.0e1))
(is (instance? Double 1.e1))
(is (instance? Double 1e1))

(is (instance? Double 1.0e-1))
(is (instance? Double 1.e-1))
(is (instance? Double 1e-1))

(is (instance? Double -1.0e+1))
(is (instance? Double -1.e+1))
(is (instance? Double -1e+1))

(is (instance? Double -1.0e1))
(is (instance? Double -1.e1))
(is (instance? Double -1e1))

(is (instance? Double -1.0e-1))
(is (instance? Double -1.e-1))
(is (instance? Double -1e-1))
```

```
(is (instance? Double +1.0))
(is (instance? Double +1.))

(is (instance? Double 1.0))
(is (instance? Double 1.))

(is (instance? Double +0.0))
(is (instance? Double +0.))

(is (instance? Double 0.0))
(is (instance? Double 0.))

(is (instance? Double -0.0))
(is (instance? Double -0.))

(is (instance? Double -1.0))
(is (instance? Double -1.))

; Read BigDecimal
(is (instance? BigDecimal 9223372036854775808M))
(is (instance? BigDecimal -9223372036854775809M))
(is (instance? BigDecimal 2147483647M))
(is (instance? BigDecimal +1M))
(is (instance? BigDecimal 1M))
(is (instance? BigDecimal +0M))
(is (instance? BigDecimal 0M))
(is (instance? BigDecimal -0M))
(is (instance? BigDecimal -1M))
(is (instance? BigDecimal -2147483648M))

(is (instance? BigDecimal +1.0e+1M))
(is (instance? BigDecimal +1.e+1M))
(is (instance? BigDecimal +1e+1M))

(is (instance? BigDecimal +1.0e1M))
(is (instance? BigDecimal +1.e1M))
(is (instance? BigDecimal +1e1M))

(is (instance? BigDecimal +1.0e-1M))
(is (instance? BigDecimal +1.e-1M))
(is (instance? BigDecimal +1e-1M))

(is (instance? BigDecimal 1.0e+1M))
(is (instance? BigDecimal 1.e+1M))
(is (instance? BigDecimal 1e+1M))

(is (instance? BigDecimal 1.0e1M))
(is (instance? BigDecimal 1.e1M))
(is (instance? BigDecimal 1e1M))
```

```

(is (instance? BigDecimal 1.0e-1M))
(is (instance? BigDecimal 1.e-1M))
(is (instance? BigDecimal 1e-1M))

(is (instance? BigDecimal -1.0e+1M))
(is (instance? BigDecimal -1.e+1M))
(is (instance? BigDecimal -1e+1M))

(is (instance? BigDecimal -1.0e1M))
(is (instance? BigDecimal -1.e1M))
(is (instance? BigDecimal -1e1M))

(is (instance? BigDecimal -1.0e-1M))
(is (instance? BigDecimal -1.e-1M))
(is (instance? BigDecimal -1e-1M))

(is (instance? BigDecimal +1.0M))
(is (instance? BigDecimal +1.M))

(is (instance? BigDecimal 1.0M))
(is (instance? BigDecimal 1.M))

(is (instance? BigDecimal +0.0M))
(is (instance? BigDecimal +0.M))

(is (instance? BigDecimal 0.0M))
(is (instance? BigDecimal 0.M))

(is (instance? BigDecimal -0.0M))
(is (instance? BigDecimal -0.M))

(is (instance? BigDecimal -1.0M))
(is (instance? BigDecimal -1.M))
)

;; Characters

(deftest t-Characters)

;; nil

(deftest t-nil)

;; Booleans

(deftest t-Booleans)

;; Keywords

(deftest t-Keywords)

```

```
(is (= :abc (keyword "abc")))
(is (= :abc (keyword 'abc)))
(is (= :*+!-_? (keyword "*+!-_?")))
(is (= :abc:def:ghi (keyword "abc:def:ghi")))
(is (= :abc/def (keyword "abc" "def")))
(is (= :abc/def (keyword 'abc/def)))
(is (= :abc.def/ghi (keyword "abc.def" "ghi")))
(is (= :abc/def.ghi (keyword "abc" "def.ghi")))
(is (= :abc:def/ghi:jk1.mno (keyword "abc:def" "ghi:jk1.mno")))
(is (instance? clojure.lang.Keyword :alphabet))
)

(deftest reading-keywords
 (are [x y] (= x (read-string y))
 :foo ":foo"
 :foo/bar ":foo/bar"
 :user/foo "::foo")
 (are [err msg form] (thrown-with-msg? err msg (read-string form))
 Exception #"Invalid token: foo:" "foo:"
 Exception #"Invalid token: :bar/" ":bar/"
 Exception #"Invalid token: ::does.not/exist" "::does.not/exist"))
;; Lists

(deftest t-Lists)

;; Vectors

(deftest t-Vectors)

;; Maps

(deftest t-Maps)

;; Sets

(deftest t-Sets)

;; Macro characters

;; Quote `

(deftest t-Quote)

;; Character `\\

(deftest t-Character)

;; Comment `;

(deftest t-Comment)
```

```
;; Deref (@)

(deftest t-Deref)

;; Dispatch (#)

;; #{} - see Sets above

;; Regex patterns (#"pattern")

(deftest t-Regex)

;; Metadata (^ or #^ (deprecated))

(deftest t-Metadata
 (is (= (meta '^^:static ^:awesome ^{:static false :bar :baz} sym)
 {:awesome true, :bar :baz, :static true})))

;; Var-quote (#')

(deftest t-Var-quote)

;; Anonymous function literal (#())

(deftest t-Anonymouns-function-literal)

;; Syntax-quote (`, note, the "backquote" character), Unquote (~) and
;; Unquote-splicing (~@)

(deftest t-Syntax-quote
 (are [x y] (= x y)
 `() () ; was NPE before SVN r1337
))

;; (read)
;; (read stream)
;; (read stream eof-is-error)
;; (read stream eof-is-error eof-value)
;; (read stream eof-is-error eof-value is-recursive)

(deftest t-read)
```

### 12.34 test/reflect.clj

— test/reflect.clj —

```
(ns clojure.test-clojure.reflect
 (:use clojure.data [clojure.reflect
 :as reflect] clojure.test clojure pprint)
 (:import [clojure.reflect AsmReflector JavaReflector])))

(defn nodiff
 [x y]
 (let [[x-only y-only common] (diff x y)]
 (when (or x-only y-only)
 (is false (with-out-str (pprint {:x-only x-only
 :y-only y-only
 :common common}))))))

(deftest compare-reflect-and-asm
 (let [cl (.getContextClassLoader (Thread/currentThread))
 asm-reflector (AsmReflector. cl)
 java-reflector (JavaReflector. cl)]
 (doseq [classname '[java.lang.Runnable
 java.lang.Object
 java.io.FileInputStream
 clojure.lang.Compiler
 clojure.lang.PersistentVector]]
 (nodiff (type-reflect classname :reflector asm-reflector)
 (type-reflect classname :reflector java-reflector)))))

(deftest field-descriptor->class-symbol-test
 (are [s d] (= s (@#'reflect/field-descriptor->class-symbol d))
 'clojure.asm.Type<><> "[Lclojure/asm/Type;"
 'int "I"
 'java.lang.Object "Ljava.lang.Object;"))

(deftest internal-name->class-symbol-test
 (are [s n] (= s (@#'reflect/internal-name->class-symbol n))
 'java.lang.Exception "java/lang/Exception"))

```

### 12.35 test/refs.clj

— test/refs.clj —

```
\getchunk{Clojure Copyright}
```

```
; Author: Frantisek Sodomka
```

```
(ns clojure.test-clojure.refs
 (:use clojure.test))

; http://clojure.org/refs

; ref
; deref, @-reader-macro
; dosync io!
; ensure ref-set alter commute
; set-validator get-validator
```

—————

## 12.36 test/repl.clj

— test/repl.clj —

```
(ns clojure.test-clojure.repl
 (:use clojure.test
 clojure.repl
 [clojure.test-helper :only [platform-newlines]]
 clojure.test-clojure.repl.example))

(deftest test-source
 (is (= "(defn foo []"
 (source-fn 'clojure.test-clojure.repl.example/foo)))
 (is (= (platform-newlines "(defn foo []\n")
 (with-out-str (source clojure.test-clojure.repl.example/foo))))
 (is (nil? (source-fn 'non-existent-fn)))))

(deftest test-dir
 (is (thrown? Exception (dir-fn 'non-existent-ns)))
 (is (= '[bar foo] (dir-fn 'clojure.test-clojure.repl.example)))
 (is (= (platform-newlines "bar\nfoo\n")
 (with-out-str (dir clojure.test-clojure.repl.example)))))

(deftest test-apropos
 (testing "with a regular expression"
 (is (= '[defmacro] (apropos #"^defmacro$")))
 (is (some #{'defmacro} (apropos #"def.acr.")))
 (is (= [] (apropos #"nothing-has-this-name"))))
```

```
(testing "with a string"
 (is (some #'defmacro) (apropos "defmacro")))
 (is (some #'defmacro) (apropos "efmac")))
 (is (= []) (apropos "nothing-has-this-name")))

(testing "with a symbol"
 (is (some #'defmacro) (apropos 'defmacro)))
 (is (some #'defmacro) (apropos 'efmac)))
 (is (= []) (apropos 'nothing-has-this-name))))
```

---

## 12.37 test/rt.clj

— test/rt.clj —

```
\getchunk{Clojure Copyright}

; Author: Stuart Halloway

(ns clojure.test-clojure.rt
 (:use clojure.test clojure.test-helper))

(defmacro with-err-print-writer
 "Evaluate with err pointing to a temporary PrintWriter, and
 return err contents as a string."
 [& body]
 '(let [s# (java.io.StringWriter.)
 p# (java.io.PrintWriter. s#)]
 (binding [*err* p#]
 ~@body
 (str s#)))))

(defmacro with-err-string-writer
 "Evaluate with err pointing to a temporary StringWriter, and
 return err contents as a string."
 [& body]
 '(let [s# (java.io.StringWriter.)]
 (binding [*err* s#]
 ~@body
 (str s#)))))

(defmacro should-print-err-message
 "Turn on all warning flags, and test that error message prints
 correctly for all semi-reasonable bindings of *err*."
 [msg-re form]
 '(binding [*warn-on-reflection* true]
```

```

(is (re-matches ~msg-re
 (with-err-string-writer (eval-in-temp-ns ~form))))
(is (re-matches ~msg-re
 (with-err-print-writer (eval-in-temp-ns ~form)))))

(defn bare-rt-print
 "Return string RT would print prior to print-initialize"
 [x]
 (with-out-str
 (try
 (push-thread-bindings {#'clojure.core/print-initialized false})
 (clojure.lang.RT/print x *out*)
 (finally
 (pop-thread-bindings)))))

(deftest rt-print-prior-to-print-initialize
 (testing "pattern literals"
 (is (= "#\"foo\"" (bare-rt-print #'foo)))))

;tpd broke these tests by splitting the regex
(deftest error-messages
; (testing "binding a core var that already refers to something"
; (should-print-err-message
; (str "WARNING: prefers already refers to: "
; "#'clojure.core/prefers in namespace: .?")
; (defn prefers [] (throw (RuntimeException. "rebound!")))))
; (testing "reflection cannot resolve field"
; (should-print-err-message
; (str "Reflection warning, NO_SOURCE_PATH: + - reference to field "
; "blah can't be resolved.?")
; (defn foo [x] (.blah x))))
; (testing "reflection cannot resolve instance method"
; (should-print-err-message
; (str "Reflection warning, NO_SOURCE_PATH:+ - call to zap can't "
; "be resolved.?")
; (defn foo [x] (.zap x 1))))
; (testing "reflection cannot resolve static method"
; (should-print-err-message
; (str "Reflection warning, NO_SOURCE_PATH:+ - call to valueOf "
; "can't be resolved.?")
; (defn foo [] (Integer/valueOf #"boom"))))
; (testing "reflection cannot resolve constructor"
; (should-print-err-message
; (str "Reflection warning, NO_SOURCE_PATH:+ - call to "
; "java.lang.String ctor can't be resolved.?")
; (defn foo [] (String. 1 2 3))))
;)

(def example-var)
(deftest binding-root-clears-macro-metadata

```

```
(alter-meta! #'example-var assoc :macro true)
(is (contains? (meta #'example-var) :macro))
(.bindRoot #'example-var 0)
(is (not (contains? (meta #'example-var) :macro)))))

(deftest last-var-wins-for-core
 (testing "you can replace a core name, with warning"
 (let [ns (temp-ns)
 replacement (gensym)]
 (with-err-string-writer (intern ns 'prefers replacement))
 (is (= replacement @('prefers (ns-publics ns)))))))
 (testing "you can replace a name you defined before"
 (let [ns (temp-ns)
 s (gensym)
 v1 (intern ns 'foo s)
 v2 (intern ns 'bar s)]
 (with-err-string-writer (.refer ns 'flatten v1))
 (.refer ns 'flatten v2)
 (is (= v2 (ns-resolve ns 'flatten))))))
 (testing "you cannot intern over an existing non-core name"
 (let [ns (temp-ns 'clojure.set)
 replacement (gensym)]
 (is (thrown? IllegalStateException
 (intern ns 'subset? replacement))))
 (is (nil? ('subset? (ns-publics ns)))))
 (is (= #'clojure.set/subset? ('subset? (ns-refers ns)))))))
 (testing "you cannot refer over an existing non-core name"
 (let [ns (temp-ns 'clojure.set)
 replacement (gensym)]
 (is (thrown? IllegalStateException
 (.refer ns 'subset? #'clojure.set/intersection)))
 (is (nil? ('subset? (ns-publics ns)))))
 (is (= #'clojure.set/subset? ('subset? (ns-refers ns)))))))

```

## 12.38 test/sequences.clj

— test/sequences.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka
; Contributors: Stuart Halloway

(ns clojure.test-clojure.sequences
 (:use clojure.test))
```

```

;; *** Tests ***

; TODO:
; apply, map, filter, remove
; and more...

(deftest test-reduce-from-chunked-into-unchunked
 (= [1 2 \a \b] (into [] (concat [1 2] "ab")))

(deftest test-reduce
 (let [int+ (fn [a b] (+ (int a) (int b)))
 arange (range 100) ;; enough to cross nodes
 avec (into [] arange)
 alist (into () arange)
 obj-array (into-array arange)
 int-array
 (into-array Integer/TYPE (map #(Integer. (int %)) arange))
 long-array (into-array Long/TYPE arange)
 float-array (into-array Float/TYPE arange)
 char-array (into-array Character/TYPE (map char arange))
 double-array (into-array Double/TYPE arange)
 byte-array (into-array Byte/TYPE (map byte arange))
 int-vec (into (vector-of :int) arange)
 long-vec (into (vector-of :long) arange)
 float-vec (into (vector-of :float) arange)
 char-vec (into (vector-of :char) (map char arange))
 double-vec (into (vector-of :double) arange)
 byte-vec (into (vector-of :byte) (map byte arange))
 all-true (into-array Boolean/TYPE (repeat 10 true))]
 (is (== 4950
 (reduce + arange)
 (reduce + avec)
 (reduce + alist)
 (reduce + obj-array)
 (reduce + int-array)
 (reduce + long-array)
 (reduce + float-array)
 (reduce int+ char-array)
 (reduce + double-array)
 (reduce int+ byte-array)
 (reduce + int-vec)
 (reduce + long-vec)
 (reduce + float-vec)
 (reduce int+ char-vec)
 (reduce + double-vec)
 (reduce int+ byte-vec)))
 (is (== 4951
 (reduce + 1 arange)
 (reduce + 1 avec)

```

```

(reduce + 1 alist)
(reduce + 1 obj-array)
(reduce + 1 int-array)
(reduce + 1 long-array)
(reduce + 1 float-array)
(reduce int+ 1 char-array)
(reduce + 1 double-array)
(reduce int+ 1 byte-array)
(reduce + 1 int-vec)
(reduce + 1 long-vec)
(reduce + 1 float-vec)
(reduce int+ 1 char-vec)
(reduce + 1 double-vec)
(reduce int+ 1 byte-vec)))
(is (= true
 (reduce #(and %1 %2) all-true)
 (reduce #(and %1 %2) true all-true)))))

(deftest test-equality
; lazy sequences
(are [x y] (= x y)
 ; fixed SVN 1288 - LazySeq and EmptyList equals/equiv
 ; http://groups.google.com/group/clojure/
 ; browse_frm/thread/286d807be9cae2a5#
 (map inc nil) ())
 (map inc ()) ())
 (map inc []) ())
 (map inc #{}) ())
 (map inc {}) ())

(deftest test-lazy-seq
(are [x] (seq? x)
 (lazy-seq nil)
 (lazy-seq [])
 (lazy-seq [1 2])))

(are [x y] (= x y)
 (lazy-seq nil) ())
 (lazy-seq [nil]) '(nil)

 (lazy-seq () ())
 (lazy-seq []) ())
 (lazy-seq #{}) ())
 (lazy-seq {}) ())
 (lazy-seq "") ())
 (lazy-seq (into-array [])) ()

 (lazy-seq (list 1 2)) '(1 2)
 (lazy-seq [1 2]) '(1 2)

```

```

(lazy-seq (sorted-set 1 2)) '(1 2)
(lazy-seq (sorted-map :a 1 :b 2)) '(:a 1 [:b 2])
(lazy-seq "abc") '(\a \b \c)
(lazy-seq (into-array [1 2])) '(1 2))

(deftest test-seq
 (is (not (seq? (seq []))))
 (is (seq? (seq [1 2])))

 (are [x y] (= x y)
 (seq nil) nil
 (seq [nil]) '(nil)

 (seq ()) nil
 (seq []) nil
 (seq #{}) nil
 (seq {}) nil
 (seq "") nil
 (seq (into-array [])) nil

 (seq (list 1 2)) '(1 2)
 (seq [1 2]) '(1 2)
 (seq (sorted-set 1 2)) '(1 2)
 (seq (sorted-map :a 1 :b 2)) '(:a 1 [:b 2])
 (seq "abc") '(\a \b \c)
 (seq (into-array [1 2])) '(1 2)))

(deftest test-cons
 (is (thrown? IllegalArgumentException (cons 1 2)))
 (are [x y] (= x y)
 (cons 1 nil) '(1)
 (cons nil nil) '(nil)

 (cons \a nil) '(\a)
 (cons \a "") '(\a)
 (cons \a "bc") '(\a \b \c)

 (cons 1 ()) '(1)
 (cons 1 '(2 3)) '(1 2 3)

 (cons 1 []) [1]
 (cons 1 [2 3]) [1 2 3]

 (cons 1 #{}) '(1)
 (cons 1 (sorted-set 2 3)) '(1 2 3)

 (cons 1 (into-array [])) '(1)
 (cons 1 (into-array [2 3])) '(1 2 3)))

```

```
(deftest test-empty
 (are [x y] (and (= (empty x) y)
 (= (class (empty x)) (class y)))
 nil nil

 () ())
 '(1 2) ())

 [] []
 [1 2] []

 {} {}
 {:a 1 :b 2} {}

 (sorted-map) (sorted-map)
 (sorted-map :a 1 :b 2) (sorted-map)

 #{} #{}
 #{1 2} #{}

 (sorted-set) (sorted-set)
 (sorted-set 1 2) (sorted-set)

 (seq ()) nil ; (seq ()) => nil
 (seq '(1 2)) ()

 (seq []) nil ; (seq []) => nil
 (seq [1 2]) ()

 (seq "") nil ; (seq "") => nil
 (seq "ab") ()

 (lazy-seq ()) ()
 (lazy-seq '(1 2)) ()

 (lazy-seq []) ()
 (lazy-seq [1 2]) ()

 ; non-coll, non-seq => nil
 42 nil
 1.2 nil
 "abc" nil))

;Tests that the comparator is preserved
;The first element should be the same in each set if preserved.
(deftest test-empty-sorted
 (let [inv-compare (comp - compare)]
 (are [x y] (= (first (into (empty x) x))
 (first (into (empty y) y))))
```

```

(first y)
(sorted-set 1 2 3) (sorted-set 1 2 3)
(sorted-set-by inv-compare 1 2 3) (sorted-set-by inv-compare 1 2 3)

(sorted-map 1 :a 2 :b 3 :c) (sorted-map 1 :a 2 :b 3 :c)
(sorted-map-by inv-compare 1 :a 2 :b 3 :c)
(sorted-map-by inv-compare 1 :a 2 :b 3 :c)))))

(deftest test-not-empty
; empty coll/seq => nil
(are [x] (= (not-empty x) nil)
()
[]
{}
#{}
(seq ())
(seq [])
(lazy-seq ())
(lazy-seq []))

; non-empty coll/seq => identity
(are [x] (and (= (not-empty x) x)
(= (class (not-empty x)) (class x)))
'(1 2)
[1 2]
{:a 1}
#{1 2}
(seq '(1 2))
(seq [1 2])
(lazy-seq '(1 2))
(lazy-seq [1 2])))

(deftest test-first
;(is (thrown? Exception (first)))
(is (thrown? IllegalArgumentException (first true)))
(is (thrown? IllegalArgumentException (first false)))
(is (thrown? IllegalArgumentException (first 1)))
;(is (thrown? IllegalArgumentException (first 1 2)))
(is (thrown? IllegalArgumentException (first \a)))
(is (thrown? IllegalArgumentException (first 's)))
(is (thrown? IllegalArgumentException (first :k)))
(are [x y] (= x y)
(first nil) nil

; string
(first "") nil
(first "a") \a
(first "abc") \a

```

```
; list
(first ()) nil
(first '(1)) 1
(first '(1 2 3)) 1

(first '(nil)) nil
(first '(1 nil)) 1
(first '(nil 2)) nil
(first '(())) ()
(first '(() nil)) ()
(first '(() 2 nil)) ()

; vector
(first []) nil
(first [1]) 1
(first [1 2 3]) 1

(first [nil]) nil
(first [1 nil]) 1
(first [nil 2]) nil
(first [[]]) []
(first [] nil) []
(first [] 2 nil) []

; set
(first #{}) nil
(first #{1}) 1
(first (sorted-set 1 2 3)) 1

(first #{nil}) nil
(first (sorted-set 1 nil)) nil
(first (sorted-set nil 2)) nil
(first #{#{}}) #{}
(first (sorted-set #{} nil)) nil
;(first (sorted-set #{} 2 nil)) nil

; map
(first {}) nil
(first (sorted-map :a 1)) '(:a 1)
(first (sorted-map :a 1 :b 2 :c 3)) '(:a 1)

; array
(first (into-array [])) nil
(first (into-array [1])) 1
(first (into-array [1 2 3])) 1
(first (to-array [nil])) nil
(first (to-array [1 nil])) 1
(first (to-array [nil 2])) nil)
```

```
(deftest test-next
 ; (is (thrown? IllegalArgumentException (next)))
 ; (is (thrown? IllegalArgumentException (next true)))
 ; (is (thrown? IllegalArgumentException (next false)))
 ; (is (thrown? IllegalArgumentException (next 1)))
 ;(is (thrown? IllegalArgumentException (next 1 2)))
 ;(is (thrown? IllegalArgumentException (next \a)))
 ;(is (thrown? IllegalArgumentException (next 's)))
 ;(is (thrown? IllegalArgumentException (next :k)))
 (are [x y] (= x y)
 (next nil) nil

 ; string
 (next "") nil
 (next "a") nil
 (next "abc") '(\b \c)

 ; list
 (next ()) nil
 (next '(1)) nil
 (next '(1 2 3)) '(2 3)

 (next '(nil)) nil
 (next '(1 nil)) '(nil)
 (next '(1 ())) '(())
 (next '(nil 2)) '(2)
 (next '(())) nil
 (next '(() nil)) '(nil)
 (next '(() 2 nil)) '(2 nil)

 ; vector
 (next []) nil
 (next [1]) nil
 (next [1 2 3]) [2 3]

 (next [nil]) nil
 (next [1 nil]) [nil]
 (next [1 []]) []
 (next [nil 2]) [2]
 (next [[]]) nil
 (next [[] nil]) [nil]
 (next [[] 2 nil]) [2 nil]

 ; set
 (next #{}) nil
 (next #{1}) nil
 (next (sorted-set 1 2 3)) '(2 3)

 (next #{nil}) nil
```

```

(next (sorted-set 1 nil)) '(1)
(next (sorted-set nil 2)) '(2)
(next #{#{}}) nil
(next (sorted-set #{} nil)) '(#{})
;(next (sorted-set #{} 2 nil)) #{}

; map
(next {}) nil
(next (sorted-map :a 1)) nil
(next (sorted-map :a 1 :b 2 :c 3)) '((:b 2) (:c 3))

; array
(next (into-array [])) nil
(next (into-array [1])) nil
(next (into-array [1 2 3])) '(2 3)

(next (to-array [nil])) nil
(next (to-array [1 nil])) '(nil)
;(next (to-array [1 (into-array [])])) (list (into-array []))
(next (to-array [nil 2])) '(2)
(next (to-array [(into-array [])])) nil
(next (to-array [(into-array []) nil])) '(nil)
(next (to-array [(into-array []) 2 nil])) '(2 nil))

(deftest test-last
 (are [x y] (= x y)
 (last nil) nil

 ; list
 (last ()) nil
 (last '(1)) 1
 (last '(1 2 3)) 3

 (last '(nil)) nil
 (last '(1 nil)) nil
 (last '(nil 2)) 2
 (last '()) ()
 (last '() nil) nil
 (last '() 2 nil) nil

 ; vector
 (last []) nil
 (last [1]) 1
 (last [1 2 3]) 3

 (last [nil]) nil
 (last [1 nil]) nil
 (last [nil 2]) 2
 (last [[]]) []
)

```

```

(last [] nil) nil
(last [] 2 nil) nil

; set
(last #{}) nil
(last #{1}) 1
(last (sorted-set 1 2 3)) 3

(last #{nil}) nil
(last (sorted-set 1 nil)) 1
(last (sorted-set nil 2)) 2
(last #{#{}}) #{}
(last (sorted-set #{} nil)) {}
;(last (sorted-set #{} 2 nil)) nil

; map
(last {}) nil
(last (sorted-map :a 1)) [:a 1]
(last (sorted-map :a 1 :b 2 :c 3)) [:c 3]

; string
(last "") nil
(last "a") \a
(last "abc") \c

; array
(last (into-array [])) nil
(last (into-array [1])) 1
(last (into-array [1 2 3])) 3
(last (to-array [nil])) nil
(last (to-array [1 nil])) nil
(last (to-array [nil 2])) 2)

;; (ffirst coll) = (first (first coll))
;;
(deftest test-ffirst
; (is (thrown? IllegalArgumentException (ffirst)))
 (are [x y] (= x y)
 (ffirst nil) nil

 (ffirst ()) nil
 (ffirst '((1 2) (3 4))) 1

 (ffirst []) nil
 (ffirst [[1 2] [3 4]]) 1

 (ffirst {}) nil
 (ffirst {:a 1}) :a

```

```

(ffirst #{}) nil
(ffirst #{{1 2}}) 1)

;; (fnext coll) = (first (next coll)) = (second coll)
;;
(deftest test-fnext
; (is (thrown? IllegalArgumentException (fnext)))
(are [x y] (= x y)
 (fnext nil) nil

 (fnext ()) nil
 (fnext '(1)) nil
 (fnext '(1 2 3 4)) 2

 (fnext []) nil
 (fnext [1]) nil
 (fnext [1 2 3 4]) 2

 (fnext {}) nil
 (fnext (sorted-map :a 1)) nil
 (fnext (sorted-map :a 1 :b 2)) [:b 2]

 (fnext #{}) nil
 (fnext #{1}) nil
 (fnext (sorted-set 1 2 3 4)) 2))

;; (nfirst coll) = (next (first coll))
;;
(deftest test-nfirst
; (is (thrown? IllegalArgumentException (nfirst)))
(are [x y] (= x y)
 (nfirst nil) nil

 (nfirst ()) nil
 (nfirst '((1 2 3) (4 5 6))) '(2 3)

 (nfirst []) nil
 (nfirst [[1 2 3] [4 5 6]]) '(2 3)

 (nfirst {}) nil
 (nfirst {:a 1}) '(1)

 (nfirst #{}) nil
 (nfirst #{{1 2}}) '(2)))

;; (nnfirst coll) = (next (next coll))
;;

```

```
(deftest test-nnext
; (is (thrown? IllegalArgumentException (nnext)))
 (are [x y] (= x y)
 (nnext nil) nil

 (nnext ()) nil
 (nnext '(1)) nil
 (nnext '(1 2)) nil
 (nnext '(1 2 3 4)) '(3 4)

 (nnext []) nil
 (nnext [1]) nil
 (nnext [1 2]) nil
 (nnext [1 2 3 4]) '(3 4)

 (nnext {}) nil
 (nnext (sorted-map :a 1)) nil
 (nnext (sorted-map :a 1 :b 2)) nil
 (nnext (sorted-map :a 1 :b 2 :c 3 :d 4)) '(:c 3) [:d 4])

 (nnext #{}) nil
 (nnext #{1}) nil
 (nnext (sorted-set 1 2)) nil
 (nnext (sorted-set 1 2 3 4)) '(3 4))

(deftest test-nth
; maps, sets are not supported
(is (thrown? UnsupportedOperationException (nth {} 0)))
(is (thrown? UnsupportedOperationException (nth {:a 1 :b 2} 0)))
(is (thrown? UnsupportedOperationException (nth #{} 0)))
(is (thrown? UnsupportedOperationException (nth #{1 2 3} 0)))

; out of bounds
(is (thrown? IndexOutOfBoundsException (nth '() 0)))
(is (thrown? IndexOutOfBoundsException (nth '(1 2 3) 5)))
(is (thrown? IndexOutOfBoundsException (nth '() -1)))
(is (thrown? IndexOutOfBoundsException (nth '(1 2 3) -1)))

(is (thrown? IndexOutOfBoundsException (nth [] 0)))
(is (thrown? IndexOutOfBoundsException (nth [1 2 3] 5)))
(is (thrown? IndexOutOfBoundsException (nth [] -1)))
(is (thrown? IndexOutOfBoundsException (nth [1 2 3] -1))) ; ???

(is (thrown? IndexOutOfBoundsException (nth (into-array []) 0)))
(is (thrown? IndexOutOfBoundsException (nth (into-array [1 2 3]) 5)))
(is (thrown? IndexOutOfBoundsException (nth (into-array []) -1)))
(is (thrown? IndexOutOfBoundsException (nth (into-array [1 2 3]) -1)))

(is (thrown? StringIndexOutOfBoundsException (nth "" 0)))
```

```

(is (thrown? StringIndexOutOfBoundsException (nth "abc" 5)))
(is (thrown? StringIndexOutOfBoundsException (nth "" -1)))
(is (thrown? StringIndexOutOfBoundsException (nth "abc" -1)))

(is (thrown? IndexOutOfBoundsException
 (nth (java.util.ArrayList. []) 0)))
(is (thrown? IndexOutOfBoundsException
 (nth (java.util.ArrayList. [1 2 3]) 5)))
(is (thrown? IndexOutOfBoundsException
 (nth (java.util.ArrayList. []) -1)) ; ???
(is (thrown? IndexOutOfBoundsException
 (nth (java.util.ArrayList. [1 2 3]) -1)) ; ???

(are [x y] (= x y)
 (nth '(1) 0) 1
 (nth '(1 2 3) 0) 1
 (nth '(1 2 3 4 5) 1) 2
 (nth '(1 2 3 4 5) 4) 5
 (nth '(1 2 3) 5 :not-found) :not-found

 (nth [1] 0) 1
 (nth [1 2 3] 0) 1
 (nth [1 2 3 4 5] 1) 2
 (nth [1 2 3 4 5] 4) 5
 (nth [1 2 3] 5 :not-found) :not-found

 (nth (into-array [1]) 0) 1
 (nth (into-array [1 2 3]) 0) 1
 (nth (into-array [1 2 3 4 5]) 1) 2
 (nth (into-array [1 2 3 4 5]) 4) 5
 (nth (into-array [1 2 3]) 5 :not-found) :not-found

 (nth "a" 0) \a
 (nth "abc" 0) \a
 (nth "abcde" 1) \b
 (nth "abcde" 4) \e
 (nth "abc" 5 :not-found) :not-found

 (nth (java.util.ArrayList. [1]) 0) 1
 (nth (java.util.ArrayList. [1 2 3]) 0) 1
 (nth (java.util.ArrayList. [1 2 3 4 5]) 1) 2
 (nth (java.util.ArrayList. [1 2 3 4 5]) 4) 5
 (nth (java.util.ArrayList. [1 2 3]) 5 :not-found) :not-found)

; regex Matchers
(let [m (re-matcher #"(a)(b)" "ababaa")]
 (re-find m) ; => ["ab" "a" "b"]
(are [x y] (= x y)
 (nth m 0) "ab"
 (nth m 1) "a"

```

```

(nth m 2) "b"
(nth m 3 :not-found) :not-found
(nth m -1 :not-found) :not-found)
(is (thrown? IndexOutOfBoundsException (nth m 3)))
(is (thrown? IndexOutOfBoundsException (nth m -1)))))

(let [m (re-matcher #"c" "ababaa")]
 (re-find m) ; => nil
 (are [x y] (= x y)
 (nth m 0 :not-found) :not-found
 (nth m 2 :not-found) :not-found
 (nth m -1 :not-found) :not-found)
 (is (thrown? IllegalStateException (nth m 0))))
 (is (thrown? IllegalStateException (nth m 2))))
 (is (thrown? IllegalStateException (nth m -1)))))

; distinct was broken for nil & false:
; fixed in rev 1278:
; http://code.google.com/p/clojure/source/detail?r=1278
;
(deftest test-distinct
 (are [x y] (= x y)
 (distinct ()) ())
 (distinct '(1)) '(1)
 (distinct '(1 2 3)) '(1 2 3)
 (distinct '(1 2 3 1 1 1)) '(1 2 3)
 (distinct '(1 1 1 2)) '(1 2)
 (distinct '(1 2 1 2)) '(1 2)

 (distinct []) ())
 (distinct [1]) '(1)
 (distinct [1 2 3]) '(1 2 3)
 (distinct [1 2 3 1 2 2 1 1]) '(1 2 3)
 (distinct [1 1 1 2]) '(1 2)
 (distinct [1 2 1 2]) '(1 2)

 (distinct "") ())
 (distinct "a") '(\a)
 (distinct "abc") '(\a \b \c)
 (distinct "cababc") '(\a \b \c)
 (distinct "aab") '(\a \b)
 (distinct "bab") '(\a \b))

 (are [x] (= (distinct [x x]) [x])
 nil
 false true
 0 42
 0.0 3.14
 2/3
)
)

```

```

OM 1M
\c
"" "abc"
'sym
:kw
() '(1 2)
[] [1 2]
{} {:a 1 :b 2}
#{()} #{[1 2]})))

(deftest test-interpose
(are [x y] (= x y)
 (interpose 0 []) ())
 (interpose 0 [1]) '(1)
 (interpose 0 [1 2]) '(1 0 2)
 (interpose 0 [1 2 3]) '(1 0 2 0 3)))

(deftest test-interleave
(are [x y] (= x y)
 (interleave [1 2] [3 4]) '(1 3 2 4)
 (interleave [1] [3 4]) '(1 3)
 (interleave [1 2] [3]) '(1 3)

 (interleave [] [3 4]) ()
 (interleave [1 2] []) ()
 (interleave [] []) ()))

(deftest test-zipmap
(are [x y] (= x y)
 (zipmap [:a :b] [1 2]) {:a 1 :b 2}
 (zipmap [:a] [1 2]) {:a 1}
 (zipmap [:a :b] [1]) {:a 1}

 (zipmap [] [1 2]) {})
 (zipmap [:a :b] []) {})
 (zipmap [] []) {}))

(deftest test-concat
(are [x y] (= x y)
 (concat) ())
 (concat []) ())
 (concat [1 2]) '(1 2)

```



```

(take 5 [1 2 3 4 5]) '(1 2 3 4 5)
(take 9 [1 2 3 4 5]) '(1 2 3 4 5)

(take 0 [1 2 3 4 5]) ()
(take -1 [1 2 3 4 5]) ()
(take -2 [1 2 3 4 5]) () ())

(deftest test-drop
 (are [x y] (= x y)
 (drop 1 [1 2 3 4 5]) '(2 3 4 5)
 (drop 3 [1 2 3 4 5]) '(4 5)
 (drop 5 [1 2 3 4 5]) ()
 (drop 9 [1 2 3 4 5]) ())

 (drop 0 [1 2 3 4 5]) '(1 2 3 4 5)
 (drop -1 [1 2 3 4 5]) '(1 2 3 4 5)
 (drop -2 [1 2 3 4 5]) '(1 2 3 4 5) ())

(deftest test-take-nth
 (are [x y] (= x y)
 (take-nth 1 [1 2 3 4 5]) '(1 2 3 4 5)
 (take-nth 2 [1 2 3 4 5]) '(1 3 5)
 (take-nth 3 [1 2 3 4 5]) '(1 4)
 (take-nth 4 [1 2 3 4 5]) '(1 5)
 (take-nth 5 [1 2 3 4 5]) '(1)
 (take-nth 9 [1 2 3 4 5]) '(1)

 ; infinite seq of 1s = (repeat 1)
 ;(take-nth 0 [1 2 3 4 5])
 ;(take-nth -1 [1 2 3 4 5])
 ;(take-nth -2 [1 2 3 4 5])
))

(deftest test-take-while
 (are [x y] (= x y)
 (take-while pos? []) ())
 (take-while pos? [1 2 3 4]) '(1 2 3 4)
 (take-while pos? [1 2 3 -1]) '(1 2 3)
 (take-while pos? [1 -1 2 3]) '(1)
 (take-while pos? [-1 1 2 3]) ()
 (take-while pos? [-1 -2 -3]) () ())

(deftest test-drop-while
 (are [x y] (= x y)
 (drop-while pos? []) ())
 (drop-while pos? [1 2 3 4]) ())

```

```

(drop-while pos? [1 2 3 -1]) '(-1)
(drop-while pos? [1 -1 2 3]) '(-1 2 3)
(drop-while pos? [-1 1 2 3]) '(-1 1 2 3)
(drop-while pos? [-1 -2 -3]) '(-1 -2 -3))

(deftest test-butlast
 (are [x y] (= x y)
 (butlast []) nil
 (butlast [1]) nil
 (butlast [1 2 3]) '(1 2)))

(deftest test-drop-last
 (are [x y] (= x y)
 ; as butlast
 (drop-last []) ()
 (drop-last [1]) ()
 (drop-last [1 2 3]) '(1 2)

 ; as butlast, but lazy
 (drop-last 1 []) ()
 (drop-last 1 [1]) ()
 (drop-last 1 [1 2 3]) '(1 2)

 (drop-last 2 []) ()
 (drop-last 2 [1]) ()
 (drop-last 2 [1 2 3]) '(1)

 (drop-last 5 []) ()
 (drop-last 5 [1]) ()
 (drop-last 5 [1 2 3]) ()

 (drop-last 0 []) ()
 (drop-last 0 [1]) '(1)
 (drop-last 0 [1 2 3]) '(1 2 3)

 (drop-last -1 []) ()
 (drop-last -1 [1]) '(1)
 (drop-last -1 [1 2 3]) '(1 2 3)

 (drop-last -2 []) ()
 (drop-last -2 [1]) '(1)
 (drop-last -2 [1 2 3]) '(1 2 3))

(deftest test-split-at
 (is (vector? (split-at 2 [])))
 (is (vector? (split-at 2 [1 2 3]))))

```

```

(are [x y] (= x y)
 (split-at 2 []) [() ()]
 (split-at 2 [1 2 3 4 5]) [(list 1 2) (list 3 4 5)]

 (split-at 5 [1 2 3]) [(list 1 2 3) ()]
 (split-at 0 [1 2 3]) [() (list 1 2 3)]
 (split-at -1 [1 2 3]) [() (list 1 2 3)]
 (split-at -5 [1 2 3]) [() (list 1 2 3)])))

(deftest test-split-with
 (is (vector? (split-with pos? [])))
 (is (vector? (split-with pos? [1 2 -1 0 3 4])))

 (are [x y] (= x y)
 (split-with pos? []) [() ()]
 (split-with pos? [1 2 -1 0 3 4]) [(list 1 2) (list -1 0 3 4)]

 (split-with pos? [-1 2 3 4 5]) [() (list -1 2 3 4 5)]
 (split-with number? [1 -2 "abc" \x])
 [(list 1 -2) (list "abc" \x)])))

(deftest test-repeat
 ;(is (thrown? IllegalArgumentException (repeat)))

 ; infinite sequence => use take
 (are [x y] (= x y)
 (take 0 (repeat 7)) ()
 (take 1 (repeat 7)) '(7)
 (take 2 (repeat 7)) '(7 7)
 (take 5 (repeat 7)) '(7 7 7 7 7))

 ; limited sequence
 (are [x y] (= x y)
 (repeat 0 7) ()
 (repeat 1 7) '(7)
 (repeat 2 7) '(7 7)
 (repeat 5 7) '(7 7 7 7 7)

 (repeat -1 7) ()
 (repeat -3 7) ())

 ; test different data types
 (are [x] (= (repeat 3 x) (list x x x))
 nil
 false true
 0 42
 0.0 3.14
 2/3
)
)

```

```

0M 1M
\c
"" "abc"
'sym
:kw
() '(1 2)
[] [1 2]
{} {:a 1 :b 2}
#{[]} #{[1 2] })
)

(deftest test-range
 (are [x y] (= x y)
 (range 0) () ; exclusive end!
 (range 1) '(0)
 (range 5) '(0 1 2 3 4)

 (range -1) ()
 (range -3) ()

 (range 2.5) '(0 1 2)
 (range 7/3) '(0 1 2)

 (range 0 3) '(0 1 2)
 (range 0 1) '(0)
 (range 0 0) ()
 (range 0 -3) ()

 (range 3 6) '(3 4 5)
 (range 3 4) '(3)
 (range 3 3) ()
 (range 3 1) ()
 (range 3 0) ()
 (range 3 -2) ()

 (range -2 5) '(-2 -1 0 1 2 3 4)
 (range -2 0) '(-2 -1)
 (range -2 -1) '(-2)
 (range -2 -2) ()
 (range -2 -5) ()

 (range 3 9 0) ()
 (range 3 9 1) '(3 4 5 6 7 8)
 (range 3 9 2) '(3 5 7)
 (range 3 9 3) '(3 6)
 (range 3 9 10) '(3)
 (range 3 9 -1) ()))
)

(deftest test-empty?

```

```

(are [x] (empty? x)
 nil
 ()
 (lazy-seq nil) ; => ()
 []
 {}
 #{}
 ""
 (into-array []))

(are [x] (not (empty? x))
 '(1 2)
 (lazy-seq [1 2])
 [1 2]
 {:a 1 :b 2}
 #{1 2}
 "abc"
 (into-array [1 2]))

(deftest test-every?
 ; always true for nil or empty coll/seq
 (are [x] (= (every? pos? x) true)
 nil
 () []
 {}
 (lazy-seq [])
 (into-array []))

 (are [x y] (= x y)
 true (every? pos? [1])
 true (every? pos? [1 2])
 true (every? pos? [1 2 3 4 5])

 false (every? pos? [-1])
 false (every? pos? [-1 -2])
 false (every? pos? [-1 -2 3])
 false (every? pos? [-1 2])
 false (every? pos? [1 -2])
 false (every? pos? [1 2 -3])
 false (every? pos? [1 2 -3 4])))

 (are [x y] (= x y)
 true (every? #{:a} [:a :a])
 ;! false (every? #{:a} [:a :b]) ; Issue 68: every? returns
 ; nil instead of false
 ;! false (every? #{:a} [:b :b])
 ;)
)

```

```
(deftest test-not-every?
 ; always false for nil or empty coll/seq
 (are [x] (= (not-every? pos? x) false)
 nil
 () []
 {} #{}
 (lazy-seq [])
 (into-array []))

 (are [x y] (= x y)
 false (not-every? pos? [1])
 false (not-every? pos? [1 2])
 false (not-every? pos? [1 2 3 4 5])

 true (not-every? pos? [-1])
 true (not-every? pos? [-1 -2])
 true (not-every? pos? [-1 -2 -3])
 true (not-every? pos? [-1 2])
 true (not-every? pos? [1 -2])
 true (not-every? pos? [1 2 -3])
 true (not-every? pos? [1 2 -3 4]))

 (are [x y] (= x y)
 false (not-every? #{:a} [:a :a])
 true (not-every? #{:a} [:a :b])
 true (not-every? #{:a} [:b :b]))))

(deftest test-not-any?
 ; always true for nil or empty coll/seq
 (are [x] (= (not-any? pos? x) true)
 nil
 () []
 {} #{}
 (lazy-seq [])
 (into-array []))

 (are [x y] (= x y)
 false (not-any? pos? [1])
 false (not-any? pos? [1 2])
 false (not-any? pos? [1 2 3 4 5])

 true (not-any? pos? [-1])
 true (not-any? pos? [-1 -2])

 false (not-any? pos? [-1 -2 -3])
 false (not-any? pos? [-1 2])
 false (not-any? pos? [1 -2])
 false (not-any? pos? [1 2 -3])
 false (not-any? pos? [1 2 -3 4]))

 (are [x y] (= x y)
```

```

false (not-any? #{:a} [:a :a])
false (not-any? #{:a} [:a :b])
true (not-any? #{:a} [:b :b]))

(deftest test-some
;; always nil for nil or empty coll/seq
(are [x] (= (some pos? x) nil)
 nil
 () []
 {} #{}
 (lazy-seq [])
 (into-array []))

(are [x y] (= x y)
 nil (some nil nil)

 true (some pos? [1])
 true (some pos? [1 2])

 nil (some pos? [-1])
 nil (some pos? [-1 -2])
 true (some pos? [-1 2])
 true (some pos? [1 -2])

 :a (some #{:a} [:a :a])
 :a (some #{:a} [:b :a])
 nil (some #{:a} [:b :b])

 :a (some #{:a} '(:a :b))
 :a (some #{:a} #{:a :b})
))

(deftest test-flatten-present
(are [expected nested-val] (= (flatten nested-val) expected)
 ;simple literals
 [] nil
 [] 1
 [] 'test
 [] :keyword
 [] 1/2
 [] #"[\r\n]"
 [] true
 [] false
 ;vectors
 [1 2 3 4 5] [[1 2] [3 4 [5]]]
 [1 2 3 4 5] [1 2 3 4 5]
 [#{:1 2} 3 4 5] [#{:1 2} 3 4 5]
 ;sets
 [] #{}
 [] #{#{1 2} 3 4 5}
)

```

```

[] #{1 2 3 4 5}
[] #_#{1 2} 3 4 5}
;lists
[] '()
[1 2 3 4 5] '(1 2 3 4 5)
;maps
[] {:a 1 :b 2}
[:a 1 :b 2] (seq {:a 1 :b 2})
[] {[::a :b] 1 :c 2}
[:a :b 1 :c 2] (seq {[::a :b] 1 :c 2})
[:a 1 2 :b 3] (seq {:a [1 2] :b 3})
;Strings
[] "12345"
[\"1 \\"2 \\"3 \\"4 \\"5] (seq "12345")
;fns
[] count
[count even? odd?] [count even? odd?])

(deftest test-group-by
 (is (= (group-by even? [1 2 3 4 5])
 {false [1 3 5], true [2 4]})))

(deftest test-partition-by
 (are [test-seq] (= (partition-by (comp even? count) test-seq)
 [[["a"] ["bb" "cccc" "dd"] ["eee" "f"] ["hh" "hh"]]
 ["a" "bb" "cccc" "dd" "eee" "f" "hh"]
 ("a" "bb" "cccc" "dd" "eee" "f" "hh"))
 (is (= (partition-by #{\a \e \i \o \u} "abcdefghijklm")
 [{"a" ["b" "c" "d"] ["e"] ["f" "g" "h"] ["i"] ["j" "k" "l" "m"]})))
))

(deftest test-frequencies
 (are [expected test-seq] (= (frequencies test-seq) expected)
 {\p 2, \s 4, \i 4, \m 1} "mississippi"
 {1 4 2 2 3 1} [1 1 1 1 2 2 3]
 {1 4 2 2 3 1} '(1 1 1 1 2 2 3)))

(deftest test-reductions
 (is (= (reductions + nil)
 [0]))
 (is (= (reductions + [1 2 3 4 5])
 [1 3 6 10 15]))
 (is (= (reductions + 10 [1 2 3 4 5])
 [10 11 13 16 20 25])))

(deftest test-rand-nth-invariants
 (let [elt (rand-nth [:a :b :c :d])]
 (is (#{:a :b :c :d} elt)))

(deftest test-partition-all
 (is (= (partition-all 4 [1 2 3 4 5 6 7 8 9])
))
)

```

```

[[1 2 3 4] [5 6 7 8] [9]]))
(is (= (partition-all 4 2 [1 2 3 4 5 6 7 8 9])
[[1 2 3 4] [3 4 5 6] [5 6 7 8] [7 8 9] [9]]))

(deftest test-shuffle-invariants
(is (= (count (shuffle [1 2 3 4])) 4))
(let [shuffled-seq (shuffle [1 2 3 4])]
(is (every? #{1 2 3 4} shuffled-seq))))
```

---

## 12.39 test/serialization.clj

— test/serialization.clj —

```

\getchunk{Clojure Copyright}

;; Author: Chas Emerick
;; cemerick@snowtide.com

(ns clojure.test-clojure.serialization
 (:use clojure.test)
 (:import (java.io ObjectOutputStream ObjectInputStream
 ByteArrayOutputStream ByteArrayInputStream)))

(defn- serialize
 "Serializes a single object, returning a byte array."
 [v]
 (with-open [bout (ByteArrayOutputStream.)]
 (ObjectOutputStream. bout)
 (.writeObject bout v)
 (.flush bout)
 (.toByteArray bout)))

(defn- deserialize
 "Deserializes and returns a single object from the given byte array."
 [bytes]
 (with-open [ois (ByteArrayInputStream. bytes ObjectInputStream.)]
 (.readObject ois)))

(defrecord SerializationRecord [a b c])
(defstruct SerializationStruct :a :b :c)

(defn- build-via-transient
 [coll]
 (persistent!
```

```

(reduce conj!
 (transient coll) (map vec (partition 2 (range 1000)))))

(defn- roundtrip
 [v]
 (let [rt (-> v serialize deserialize)
 rt-seq (-> v seq serialize deserialize)]
 (and (= v rt)
 (= (seq v) (seq rt))
 (= (seq v) rt-seq)))

(deftest sequable-serialization
 (are [val] (roundtrip val)
 ; lists and related
 (list)
 (apply list (range 10))
 (cons 0 nil)
 (clojure.lang.Cons. 0 nil)

 ; vectors
 []
 (into [] (range 10))
 (into [] (range 25))
 (into [] (range 100))
 (into [] (range 500))
 (into [] (range 1000))

 ; maps
 {}
 {:a 5 :b 0}
 (apply array-map (range 100))
 (apply hash-map (range 100))

 ; sets
 #{}
 #{'a' 'b' 'c'}
 (set (range 10))
 (set (range 25))
 (set (range 100))
 (set (range 500))
 (set (range 1000))
 (sorted-set)
 (sorted-set 'a 'b 'c)
 (apply sorted-set (reverse (range 10)))
 (apply sorted-set (reverse (range 25)))
 (apply sorted-set (reverse (range 100)))
 (apply sorted-set (reverse (range 500)))
 (apply sorted-set (reverse (range 1000)))

 ; queues

```

```

clojure.lang.PersistentQueue/EMPTY
(into clojure.lang.PersistentQueue/EMPTY (range 50))

; lazy seqs
(lazy-seq nil)
(lazy-seq (range 50))

; transient / persistent! round-trip
(build-via-transient [])
(build-via-transient {})
(build-via-transient #{})

; array-seqs
(seq (make-array Object 10))
(seq (make-array Boolean/TYPE 10))
(seq (make-array Byte/TYPE 10))
(seq (make-array Character/TYPE 10))
(seq (make-array Double/TYPE 10))
(seq (make-array Float/TYPE 10))
(seq (make-array Integer/TYPE 10))
(seq (make-array Long/TYPE 10))

; "records"
(SerializationRecord. 0 :foo (range 20))
(struct SerializationStruct 0 :foo (range 20))

; misc seqs
(seq "s11n")
(range 50)
(rseq (apply sorted-set (reverse (range 100)))))

(deftest misc-serialization
(are [v] (= v (-> v serialize deserialize)))
 25/3
 :keyword
 ::namespaced-keyword
 'symbol))

(deftest interned-serializations
(are [v] (identical? v (-> v serialize deserialize)))
 clojure.lang.RT/DEFAULT_COMPARATOR

 ; namespaces just get deserialized back into the same-named
 ; ns in the present runtime
 ; (they're referred to by defrecord instances)
 ns))

(deftest function-serialization
(let [capture 5]
(are [f] (= capture ((-> f serialize deserialize))))
```

```

(constantly 5)
(fn [] 5)
#(do 5)
(constantly capture)
(fn [] capture)
#(do capture)))))

(deftest check-unserializable-objects
 (are [t] (thrown? java.io.NotSerializableException (serialize t))
 ;;= transients
 (transient [])
 (transient {})
 (transient #{}))

 ;;= reference types
 (atom nil)
 (ref nil)
 (agent nil)
 #'+

 ;;= stateful seqs
 (enumeration-seq (java.util.Collections/enumeration (range 50)))
 (iterator-seq (.iterator (range 50))))
```

---

## 12.40 test/special.clj

— test/special.clj —

```

\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka

;;
;; Test special forms, macros and metadata
;;

(ns clojure.test-clojure.special
 (:use clojure.test))

; http://clojure.org/special_forms

; let, letfn
; quote
; var
; fn
```

---

## 12.41 test/string.clj

— test/string.clj —

```
(ns clojure.test-clojure.string
 (:require [clojure.string :as s])
 (:use clojure.test))

(deftest t-split
 (is (= ["a" "b"] (s/split "a-b" #"-")))
 (is (= ["a" "b-c"] (s/split "a-b-c" #"-" 2)))
 (is (vector? (s/split "abc" #"-"))))

(deftest t-reverse
 (is (= "tab" (s/reverse "bat"))))

(deftest t-replace
 (is (= "faabar" (s/replace "foobar" \o \a)))
 (is (= "barbarbar" (s/replace "foobarfoo" "foo" "bar")))
 (is (= "FOObarFOO" (s/replace "foobarfoo" #"foo" s/upper-case)))))

(deftest t-replace-first
 (is (= "barbarfoo" (s/replace-first "foobarfoo" "foo" "bar")))
 (is (= "barbarfoo" (s/replace-first "foobarfoo" #"foo" "bar")))
 (is (= "z.ology" (s/replace-first "zoology" \o \.)))
 (is (= "FOObarfoo" (s/replace-first "foobarfoo" #"foo" s/upper-case)))))

(deftest t-join
 (are [x coll] (= x (s/join coll))
 "" nil
 "" []
 "1" [1]
 "12" [1 2])
 (are [x sep coll] (= x (s/join sep coll))
 "1,2,3" \, [1 2 3]
 "" \, []
 "1" \, [1]
 "1 and-a 2 and-a 3" " and-a " [1 2 3]))

(deftest t-trim-newline
 (is (= "foo" (s/trim-newline "foo\n"))))
 (is (= "foo" (s/trim-newline "foo\r\n"))))
 (is (= "foo" (s/trim-newline "foo"))))
```

```
(is (= "" (s/trim-newline "")))))

(deftest t-capitalize
 (is (= "Foobar" (s/capitalize "foobar")))
 (is (= "Foobar" (s/capitalize "FOOBAR"))))

(deftest t-triml
 (is (= "foo " (s/triml " foo ")))
 (is (= "" (s/triml " ")))))

(deftest t-trimr
 (is (= " foo" (s/trimr " foo ")))
 (is (= "" (s/trimr " ")))))

(deftest t-trim
 (is (= "foo" (s/trim " foo \r\n"))))

(deftest t-upper-case
 (is (= "FOOBAR" (s/upper-case "Foobar"))))

(deftest t-lower-case
 (is (= "foobar" (s/lower-case "FooBar"))))

(deftest nil-handling
 (are [f args] (thrown? NullPointerException (apply f args))
 s/reverse [nil]
 s/replace [nil #'foo "bar"]
 s/replace-first [nil #'foo "bar"]
 s/capitalize [nil]
 s/upper-case [nil]
 s/lower-case [nil]
 s/split [nil #"-"]
 s/split [nil #"- 1]
 s/trim [nil]
 s/triml [nil]
 s/trimr [nil]
 s/trim-newline [nil])))

(deftest char-sequence-handling
 (are [result f args] (let [[^CharSequence s & more] args]
 (= result (apply f (StringBuffer. s) more)))
 "paz" s/reverse ["zap"]
 "foo:bar" s/replace ["foo-bar" \- \:]
 "ABC" s/replace ["abc" #"\\w" s/upper-case]
 "faa" s/replace ["foo" #"o" (StringBuffer. "a")]
 "baz::quux" s/replace-first ["baz--quux" #"--" "::"]
 "baz::quux" s/replace-first ["baz--quux" (StringBuffer. "--")
 (StringBuffer. "::")]
 "zim-zam" s/replace-first ["zim zam" #" " (StringBuffer. "-")]
 "Pow" s/capitalize ["POW"]
```

```

"BOOM" s/upper-case ["boom"]
"whimper" s/lower-case ["whimPER"]
["foo" "bar"] s/split ["foo-bar" #"-"]
"calvino" s/trim [" calvino "]
"calvino " s/triml [" calvino "]
" calvino" s/trimr [" calvino "]
"the end" s/trim-newline ["the end\r\n\r\n\r\n"]
true s/blank? []
["a" "b"] s/split-lines ["a\nb"]
"fa la la" s/escape ["fo lo lo" {\o \a}])))

(deftest t-escape
 (is (= "<foo>"
 (s/escape "<foo>" {\& "& " \< "<" \> ">"})))
 (is (= " \\\"foo\\\" "
 (s/escape " \"foo\" " {\\" \"\\\"})))
 (is (= "faabor"
 (s/escape "foobar" {\a \o, \o \a})))))

(deftest t-blank
 (is (s/blank? nil))
 (is (s/blank? ""))
 (is (s/blank? " "))
 (is (s/blank? " \t \n \r ")))
 (is (not (s/blank? " foo ")))))

(deftest t-split-lines
 (let [result (s/split-lines "one\ntwo\r\nthree")]
 (is (= ["one" "two" "three"] result))
 (is (vector? result)))
 (is (= (list "foo") (s/split-lines "foo")))))

```

---

## 12.42 test/test.clj

— test/test.clj —

```

\getchunk{Clojure Copyright}

;; test_clojure/test.clj: unit tests for test.clj

;; by Stuart Sierra
;; January 16, 2009

;; Thanks to Chas Emerick, Allen Rohner, and Stuart Halloway for

```

```

;; contributions and suggestions.

(ns clojure.test-clojure.test
 (:use clojure.test))

(deftest can-test-symbol
 (let [x true]
 (is x "Should pass"))
 (let [x false]
 (is x "Should fail")))

(deftest can-test-boolean
 (is true "Should pass")
 (is false "Should fail"))

(deftest can-test-nil
 (is nil "Should fail"))

(deftest can-test=-
 (is (= 2 (+ 1 1)) "Should pass")
 (is (= 3 (+ 2 2)) "Should fail"))

(deftest can-test-instance
 (is (instance? Long (+ 2 2)) "Should pass")
 (is (instance? Float (+ 1 1)) "Should fail"))

(deftest can-test-thrown
 (is (thrown? ArithmeticException (/ 1 0)) "Should pass")
 ;; No exception is thrown:
 (is (thrown? Exception (+ 1 1)) "Should fail")
 ;; Wrong class of exception is thrown:
 (is (thrown? ArithmeticException
 (throw (RuntimeException.))) "Should error"))

(deftest can-test-thrown-with-msg
 (is (thrown-with-msg? ArithmeticException
 #"Divide by zero" (/ 1 0)) "Should pass")
 ;; Wrong message string:
 (is (thrown-with-msg? ArithmeticException
 #"Something else" (/ 1 0)) "Should fail")
 ;; No exception is thrown:
 (is (thrown? Exception (+ 1 1)) "Should fail")
 ;; Wrong class of exception is thrown:
 (is (thrown-with-msg? IllegalArgumentException
 #"Divide by zero" (/ 1 0)) "Should error"))

(deftest can-catch-unexpected-exceptions
 (is (= 1 (throw (Exception.))) "Should error"))

```

```
(deftest can-test-method-call
 (is (.startsWith "abc" "a") "Should pass")
 (is (.startsWith "abc" "d") "Should fail"))

(deftest can-test-anonymous-fn
 (is (#(.startsWith % "a") "abc") "Should pass")
 (is (#(.startsWith % "d") "abc") "Should fail"))

(deftest can-test-regexps
 (is (re-matches #"\^ab.*\$" "abbabba") "Should pass")
 (is (re-matches #"\^cd.*\$" "abbabba") "Should fail")
 (is (re-find #"\ab" "abbabba") "Should pass")
 (is (re-find #"\cd" "abbabba") "Should fail"))

(deftest #^{:has-meta true} can-add-metadata-to-tests
 (is (:has-meta (meta #'can-add-metadata-to-tests)) "Should pass"))

;; still have to declare the symbol before testing unbound symbols
(declare does-not-exist)

#_(deftest can-test-unbound-symbol
 (is (= nil does-not-exist) "Should error"))

#_(deftest can-test-unbound-function
 (is (does-not-exist) "Should error"))

;; Here, we create an alternate version of test/report, that
;; compares the event with the message, then calls the original
;; 'report' with modified arguments.

(declare ^:dynamic original-report)

(defn custom-report [data]
 (let [event (:type data)
 msg (:message data)
 expected (:expected data)
 actual (:actual data)
 passed (cond
 (= event :fail) (= msg "Should fail")
 (= event :pass) (= msg "Should pass")
 (= event :error) (= msg "Should error")
 :else true)]
 (if passed
 (original-report {:type :pass, :message msg,
 :expected expected, :actual actual})
 (original-report {:type :fail, :message (str msg " but got " event)
 :expected expected, :actual actual})))))

;; test-ns-hook will be used by test/test-ns to run tests in this
```

```
;; namespace.
(defn test-ns-hook []
 (binding [original-report report
 report custom-report]
 (test-all-vars (find-ns 'clojure.test-clojure.test))))
```

---

## 12.43 test/test'fixtures.clj

— test/test'fixtures.clj —

```
\getchunk{Clojure Copyright}
;
;;; test_fixtures.clj: unit tests for fixtures in test.clj

;; by Stuart Sierra
;; March 28, 2009

(ns clojure.test-clojure.test-fixtures
 (:use clojure.test))

(declare ^:dynamic *a* ^:dynamic *b* ^:dynamic *c* ^:dynamic *d*)

(def ^:dynamic *n* 0)

(defn fixture-a [f]
 (binding [*a* 3] (f)))

(defn fixture-b [f]
 (binding [*b* 5] (f)))

(defn fixture-c [f]
 (binding [*c* 7] (f)))

(defn fixture-d [f]
 (binding [*d* 11] (f)))

(defn inc-n-fixture [f]
 (binding [*n* (inc *n*)] (f)))

(use-fixtures :once fixture-a fixture-b)

(use-fixtures :each fixture-c fixture-d inc-n-fixture)
(use-fixtures :each fixture-c fixture-d inc-n-fixture)

(deftest can-use-once-fixtures
```

```
(is (= 3 *a*))
(is (= 5 *b*))

(deftest can-use-each-fixtures
 (is (= 7 *c*))
 (is (= 11 *d*)))

(deftest use-fixtures-replaces
 (is (= *n* 1)))
```

—————

## 12.44 test/transients.clj

— test/transients.clj —

```
(ns clojure.test-clojure.transients
 (:use clojure.test))

(deftest popping-off
 (testing "across a node boundary"
 (are [n]
 (let [v (-> (range n) vec)]
 (= (subvec v 0 (- n 2)) (-> v transient pop! pop! persistent!)))
 33 (+ 32 (inc (* 32 32))) (+ 32 (inc (* 32 32 32)))))

 (testing "off the end"
 (is (thrown-with-msg? IllegalStateException #'"Can't pop empty vector"
 (-> [] transient pop!)))))
```

—————

## 12.45 test/vars.clj

— test/vars.clj —

```
\getchunk{Clojure Copyright}

; Author: Frantisek Sodomka, Stephen C. Gilardi

(ns clojure.test-clojure.vars
 (:use clojure.test))

; http://clojure.org/vars
```

```

; def
; defn defn- defonce

; declare intern binding find-var var

(def ^:dynamic a)
(deftest test-binding
 (are [x y] (= x y)
 (eval `(binding [a 4] a)) 4 ; regression in Clojure SVN r1370
))

; var-get var-set alter-var-root [var? (predicates.clj)]
; with-in-str with-out-str
; with-open

(deftest test-with-local-vars
 (let [factorial (fn [x]
 (with-local-vars [acc 1, cnt x]
 (while (> @cnt 0)
 (var-set acc (* @acc @cnt))
 (var-set cnt (dec @cnt)))
 @acc))])
 (is (= (factorial 5) 120)))

(deftest test-with-precision
 (are [x y] (= x y)
 (with-precision 4 (+ 3.5555555M 1)) 4.5556M
 (with-precision 6 (+ 3.5555555M 1)) 4.555556M
 (with-precision 6 :rounding CEILING (+ 3.5555555M 1)) 4.555556M
 (with-precision 6 :rounding FLOOR (+ 3.5555555M 1)) 4.55555M
 (with-precision 6 :rounding HALF_UP (+ 3.5555555M 1)) 4.555556M
 (with-precision 6 :rounding HALF_DOWN (+ 3.5555555M 1)) 4.555556M
 (with-precision 6 :rounding HALF_EVEN (+ 3.5555555M 1)) 4.555556M
 (with-precision 6 :rounding UP (+ 3.5555555M 1)) 4.555556M
 (with-precision 6 :rounding DOWN (+ 3.5555555M 1)) 4.555555M
 (with-precision 6 :rounding UNNECESSARY (+ 3.5555M 1)) 4.5555M))

(deftest test-settable-math-context
 (is (= (clojure.main/with-bindings
 (set! *math-context* (java.math.MathContext. 8))
 (+ 3.555555555555555M 1))
 4.5555556M)))

; set-validator get-validator

; doc find-doc test

(def stub-me :original)

```

```
(deftest test-with-redefs-fn
 (let [p (promise)]
 (with-redefs-fn #'stub-me :temp}
 (fn []
 (.start (Thread. #(deliver p stub-me)))
 @p)
 (is (= :temp @p))
 (is (= :original stub-me)))))

(deftest test-with-redefs
 (let [p (promise)]
 (with-redefs [stub-me :temp]
 (.start (Thread. #(deliver p stub-me)))
 @p)
 (is (= :temp @p))
 (is (= :original stub-me)))))

(deftest test-with-redefs-throw
 (let [p (promise)]
 (is (thrown? Exception
 (with-redefs [stub-me :temp]
 (deliver p stub-me)
 (throw (Exception. "simulated failure in with-redefs")))))
 (is (= :temp @p))
 (is (= :original stub-me))))
```

—————

## 12.46 test/vectors.clj

— test/vectors.clj —

```
\getchunk{Clojure Copyright}

; Author: Stuart Halloway, Daniel Solano Gmez

(ns clojure.test-clojure.vectors
 (:use clojure.test))

(deftest test-reversed-vec
 (let [r (range 6)
 v (into (vector-of :int) r)
 reversed (.rseq v)]
 (testing "returns the right impl"
 (is (= clojure.lang.APersistentVector$RSeq (class reversed))))
 (testing "RSeq methods"
```

```

(is (= [5 4 3 2 1 0] reversed))
(is (= 5 (.index reversed)))
(is (= 5 (.first reversed)))
(is (= [4 3 2 1 0] (.next reversed)))
(is (= [3 2 1 0] (.. reversed next next)))
(is (= 6 (.count reversed))))
(testing "clojure calling through"
 (is (= 5 (first reversed)))
 (is (= 5 (nth reversed 0)))))
(testing "empty reverses to nil"
 (is (nil? (.. v empty rseq)))))

(deftest test-vecseq
 (let [r (range 100)
 vs (into (vector-of :int) r)
 vs-1 (next vs)
 vs-32 (.chunkedNext (seq vs))]
 (testing "="
 (are [a b] (= a b)
 vs vs
 vs-1 vs-1
 vs-32 vs-32)
 (are [a b] (not= a b)
 vs vs-1
 vs-1 vs
 vs vs-32
 vs-32 vs)))
 (testing "IPersistentCollection.empty"
 (are [a]
 (identical? clojure.lang.PersistentList/EMPTY (.empty (seq a)))
 vs vs-1 vs-32))
 (testing "IPersistentCollection.cons"
 (are [result input] (= result (.cons input :foo))
 [:foo 1] (seq (into (vector-of :int) [1]))))
 (testing "IPersistentCollection.count"
 (are [ct s] (= ct (.count (seq s)))
 100 vs
 99 vs-1
 68 vs-32)
 ;; can't manufacture this scenario: ASeq defers to Counted, but
 ;; LazySeq doesn't, so Counted never gets checked on reified seq
 ;; below
 #_(testing "hops to counted when available"
 (is (= 200
 (.count (concat
 (seq vs)
 (reify clojure.lang.ISeq
 (seq [this] this)
 clojure.lang.Counted
 (count [_] 100)))))))
)
)
)
```

```

(testing "IPersistentCollection.equiv"
 (are [a b] (true? (.equiv a b))
 vs vs
 vs-1 vs-1
 vs-32 vs-32
 vs r)
 (are [a b] (false? (.equiv a b))
 vs vs-1
 vs-1 vs
 vs vs-32
 vs-32 vs
 vs nil)))))

(deftest test-vec-compare
 (let [nums (range 1 100)
 ; randomly replaces a single item with the given value
 rand-replace
 (fn[val]
 (let [r (rand-int 99)]
 (concat (take r nums) [val] (drop (inc r) nums))))
 ; all num sequences in map
 num-seqs {:standard nums
 :empty '()
 ; different lengths
 :longer (concat nums [100])
 :shorter (drop-last nums)
 ; greater by value
 :first-greater (concat [100] (next nums))
 :last-greater (concat (drop-last nums) [100])
 :rand-greater-1 (rand-replace 100)
 :rand-greater-2 (rand-replace 100)
 :rand-greater-3 (rand-replace 100)
 ; lesser by value
 :first-lesser (concat [0] (next nums))
 :last-lesser (concat (drop-last nums) [0])
 :rand-lesser-1 (rand-replace 0)
 :rand-lesser-2 (rand-replace 0)
 :rand-lesser-3 (rand-replace 0)}
 ; a way to create compare values based on num-seqs
 create-vals
 (fn [base-val]
 (zipmap (keys num-seqs)
 (map #(into base-val %1) (vals num-seqs))))
 ; Vecs made of int primitives
 int-vecs (create-vals (vector-of :int))
 ; Vecs made of long primitives
 long-vecs (create-vals (vector-of :long))
 ; standard boxing vectors
 regular-vecs (create-vals []))
 ; the standard int Vec for comparisons
)

```

```

 int-vec (:standard int-vecs)]
(testing "compare"
 (testing "identical"
 (is (= 0 (compare int-vec int-vec))))
 (testing "equivalent"
 (are [x y] (= 0 (compare x y))
 ; standard
 int-vec (:standard long-vecs)
 (:standard long-vecs) int-vec
 int-vec (:standard regular-vecs)
 (:standard regular-vecs) int-vec
 ; empty
 (:empty int-vecs) (:empty long-vecs)
 (:empty long-vecs) (:empty int-vecs)))
 (testing "lesser"
 (are [x] (= -1 (compare int-vec x))
 (:longer int-vecs)
 (:longer long-vecs)
 (:longer regular-vecs)
 (:first-greater int-vecs)
 (:first-greater long-vecs)
 (:first-greater regular-vecs)
 (:last-greater int-vecs)
 (:last-greater long-vecs)
 (:last-greater regular-vecs)
 (:rand-greater-1 int-vecs)
 (:rand-greater-1 long-vecs)
 (:rand-greater-1 regular-vecs)
 (:rand-greater-2 int-vecs)
 (:rand-greater-2 long-vecs)
 (:rand-greater-2 regular-vecs)
 (:rand-greater-3 int-vecs)
 (:rand-greater-3 long-vecs)
 (:rand-greater-3 regular-vecs))
 (are [x] (= -1 (compare x int-vec))
 nil
 (:empty int-vecs)
 (:empty long-vecs)
 (:empty regular-vecs)
 (:shorter int-vecs)
 (:shorter long-vecs)
 (:shorter regular-vecs)
 (:first-lesser int-vecs)
 (:first-lesser long-vecs)
 (:first-lesser regular-vecs)
 (:last-lesser int-vecs)
 (:last-lesser long-vecs)
 (:last-lesser regular-vecs)
 (:rand-lesser-1 int-vecs)
 (:rand-lesser-1 long-vecs)

```

```
(:rand-lesser-1 regular-vecs)
(:rand-lesser-2 int-vecs)
(:rand-lesser-2 long-vecs)
(:rand-lesser-2 regular-vecs)
(:rand-lesser-3 int-vecs)
(:rand-lesser-3 long-vecs)
(:rand-lesser-3 regular-vecs)))
(testing "greater"
 (are [x] (= 1 (compare int-vec x))
 nil
 (:empty int-vecs)
 (:empty long-vecs)
 (:empty regular-vecs)
 (:shorter int-vecs)
 (:shorter long-vecs)
 (:shorter regular-vecs)
 (:first-lesser int-vecs)
 (:first-lesser long-vecs)
 (:first-lesser regular-vecs)
 (:last-lesser int-vecs)
 (:last-lesser long-vecs)
 (:last-lesser regular-vecs)
 (:rand-lesser-1 int-vecs)
 (:rand-lesser-1 long-vecs)
 (:rand-lesser-1 regular-vecs)
 (:rand-lesser-2 int-vecs)
 (:rand-lesser-2 long-vecs)
 (:rand-lesser-2 regular-vecs)
 (:rand-lesser-3 int-vecs)
 (:rand-lesser-3 long-vecs)
 (:rand-lesser-3 regular-vecs)))
 (are [x] (= 1 (compare x int-vec))
 (:longer int-vecs)
 (:longer long-vecs)
 (:longer regular-vecs)
 (:first-greater int-vecs)
 (:first-greater long-vecs)
 (:first-greater regular-vecs)
 (:last-greater int-vecs)
 (:last-greater long-vecs)
 (:last-greater regular-vecs)
 (:rand-greater-1 int-vecs)
 (:rand-greater-1 long-vecs)
 (:rand-greater-1 regular-vecs)
 (:rand-greater-2 int-vecs)
 (:rand-greater-2 long-vecs)
 (:rand-greater-2 regular-vecs)
 (:rand-greater-3 int-vecs)
 (:rand-greater-3 long-vecs)
 (:rand-greater-3 regular-vecs))))
```

```

(testing "Comparable.compareTo"
 (testing "incompatible"
 (is (thrown? NullPointerException (.compareTo int-vec nil)))
 (are [x] (thrown? ClassCastException (.compareTo int-vec x))
 '()
 '{})
 (#{})
 (sorted-set)
 (sorted-map)
 nums
 1))
 (testing "identical"
 (is (= 0 (.compareTo int-vec int-vec))))
 (testing "equivalent"
 (are [x] (= 0 (.compareTo int-vec x))
 (:standard long-vecs)
 (:standard regular-vecs)))
 (testing "lesser"
 (are [x] (= -1 (.compareTo int-vec x))
 (:longer int-vecs)
 (:longer long-vecs)
 (:longer regular-vecs)
 (:first-greater int-vecs)
 (:first-greater long-vecs)
 (:first-greater regular-vecs)
 (:last-greater int-vecs)
 (:last-greater long-vecs)
 (:last-greater regular-vecs)
 (:rand-greater-1 int-vecs)
 (:rand-greater-1 long-vecs)
 (:rand-greater-1 regular-vecs)
 (:rand-greater-2 int-vecs)
 (:rand-greater-2 long-vecs)
 (:rand-greater-2 regular-vecs)
 (:rand-greater-3 int-vecs)
 (:rand-greater-3 long-vecs)
 (:rand-greater-3 regular-vecs)))
 (testing "greater"
 (are [x] (= 1 (.compareTo int-vec x))
 (:empty int-vecs)
 (:empty long-vecs)
 (:empty regular-vecs)
 (:shorter int-vecs)
 (:shorter long-vecs)
 (:shorter regular-vecs)
 (:first-lesser int-vecs)
 (:first-lesser long-vecs)
 (:first-lesser regular-vecs)
 (:last-lesser int-vecs)
 (:last-lesser long-vecs)))

```

```

(:last-lesser regular-vecs)
(:rand-lesser-1 int-vecs)
(:rand-lesser-1 long-vecs)
(:rand-lesser-1 regular-vecs)
(:rand-lesser-2 int-vecs)
(:rand-lesser-2 long-vecs)
(:rand-lesser-2 regular-vecs)
(:rand-lesser-3 int-vecs)
(:rand-lesser-3 long-vecs)
(:rand-lesser-3 regular-vecs)))))

(deftest test-vec-associative
 (let [empty-v (vector-of :long)
 v (into empty-v (range 1 6))]
 (testing "Associative.containsKey"
 (are [x] (.containsKey v x)
 0 1 2 3 4)
 (are [x] (not (.containsKey v x))
 -1 -100 nil [] "" #"#" #{} 5 100)
 (are [x] (not (.containsKey empty-v x))
 0 1))
 (testing "contains?"
 (are [x] (contains? v x)
 0 2 4)
 (are [x] (not (contains? v x))
 -1 -100 nil "" 5 100)
 (are [x] (not (contains? empty-v x))
 0 1))
 (testing "Associative.entryAt"
 (are [idx val] (= (clojure.lang.MapEntry. idx val)
 (.entryAt v idx)))
 0 1
 2 3
 4 5)
 (are [idx] (nil? (.entryAt v idx))
 -5 -1 5 10 nil "")
 (are [idx] (nil? (.entryAt empty-v idx))
 0 1))))
```

---

## 12.47 test/java`5.clj

— test/java`5.clj —

```
;; java 5 annotation tests
(in-ns 'clojure.test-clojure.annotations)
```

```

(import [java.lang.annotation Annotation Retention
 RetentionPolicy Target ElementType])
(definterface Foo (foo []))

(deftype #^{:Deprecated true
 Retention RetentionPolicy/RUNTIME}
 Bar [#^int a
 #^{:tag int
 Deprecated true
 Retention RetentionPolicy/RUNTIME} b]
 Foo (#^{:Deprecated true
 Retention RetentionPolicy/RUNTIME}
 foo [this] 42))

(defn annotation->map
 "Converts a Java annotation (which conceals data)
 into a map (which makes it usable). Not lazy.
 Works recursively. Returns non-annotations unscathed."
 [#^java.lang.annotation.Annotation o]
 (cond
 (instance? Annotation o)
 (let [type (.annotationType o)
 itfs (-> (into #{} (supers type))
 (disj java.lang.annotation.Annotation))
 data-methods (into #{} (mapcat #(.getDeclaredMethods %) itfs))]
 (into
 {:annotationType (.annotationType o)}
 (map
 (fn [m] [(keyword (.getName m))
 (annotation->map (.invoke m o nil))])
 data-methods)))
 (or (sequential? o) (.isArray (class o)))
 (map annotation->map o)
 :else o))

 (def expected-annotations
 #{{:annotationType java.lang.annotation.Retention,
 :value RetentionPolicy/RUNTIME}
 {:annotationType java.lang.Deprecated}})

 (deftest test-annotations-on-type
 (is (= expected-annotations
 (into #{} (map annotation->map (.getAnnotations Bar))))))

 (deftest test-annotations-on-field
 (is (= expected-annotations
 (into #{})))

```

```
(map annotation->map (.getAnnotations (.getField Bar "b")))))))

(deftest test-annotations-on-method
 (is (= expected-annotations
 (into #{}
 (map annotation->map
 (.getAnnotations (.getMethod Bar "foo" nil)))))))
```

---

## 12.48 test/java'6'and'later.clj

— test/java'6'and'later.clj —

```
;; java 6 annotation tests
(in-ns 'clojure.test-clojure.annotations)

(import [java.lang.annotation Annotation Retention
 RetentionPolicy Target ElementType]
 [javax.xml.ws WebServiceRef WebServiceRefs])
(definterface Foo (foo []))

(deftype #'^{:Deprecated true
 Retention RetentionPolicy/RUNTIME
 javax.annotation.processing.SupportedOptions
 ["foo" "bar" "baz"]
 javax.xml.ws.soap.Addressing {:enabled false :required true}
 WebServiceRefs
 [(WebServiceRef {:name "fred" :type String})
 (WebServiceRef {:name "ethel" :mappedName "lucy"})]}
 Bar [#^int a
 #^{:tag int
 ^{:Deprecated true
 Retention RetentionPolicy/RUNTIME
 javax.annotation.processing.SupportedOptions
 ["foo" "bar" "baz"]
 javax.xml.ws.soap.Addressing {:enabled false :required true}
 WebServiceRefs
 [(WebServiceRef {:name "fred" :type String})
 (WebServiceRef {:name "ethel" :mappedName "lucy"})]}
 b]
 Foo (#^{:Deprecated true
 Retention RetentionPolicy/RUNTIME
 javax.annotation.processing.SupportedOptions
 ["foo" "bar" "baz"]}
```

```

javax.xml.ws.soap.Addressing {:enabled false :required true}
WebServiceRefs
 [(WebServiceRef {:name "fred" :type String})
 (WebServiceRef {:name "ethel" :mappedName "lucy"})]
foo [this] 42))

(defn annotation->map
 "Converts a Java annotation (which conceals data)
 into a map (which makes it usable). Not lazy.
 Works recursively. Returns non-annotations unscathed."
 [#^java.lang.annotation.Annotation o]
 (cond
 (instance? Annotation o)
 (let [type (.annotationType o)
 itfs (-> (into #{type} (supers type))
 (disj java.lang.annotation.Annotation))
 data-methods (into #{} (mapcat #(.getDeclaredMethods %) itfs))]
 (into
 {:annotationType (.annotationType o)}
 (map
 (fn [m] [(keyword (.getName m))
 (annotation->map (.invoke m o nil))])
 data-methods)))
 (or (sequential? o) (.isArray (class o)))
 (map annotation->map o)
 :else o))

 (def expected-annotations
 #{{:annotationType java.lang.annotation.Retention,
 :value RetentionPolicy/RUNTIME}
 {:annotationType javax.xml.ws.WebServiceRefs,
 :value [{:annotationType javax.xml.ws.WebServiceRef,
 :name "fred", :mappedName "", :type java.lang.String,
 :wsdlLocation "", :value java.lang.Object}
 {:annotationType javax.xml.ws.WebServiceRef,
 :name "ethel", :mappedName "lucy",
 :type java.lang.Object,
 :wsdlLocation "", :value java.lang.Object}]}}
 {:annotationType javax.xml.ws.soap.Addressing,
 :enabled false, :required true}
 {:annotationType javax.annotation.processing.SupportedOptions,
 :value ["foo" "bar" "baz"]}
 {:annotationType java.lang.Deprecated}})

(deftest test-annotations-on-type
 (is (= expected-annotations
 (into #{} (map annotation->map (.getAnnotations Bar))))))

(deftest test-annotations-on-field

```

```
(is (= expected-annotations
 (into #{}
 (map annotation->map (.getAnnotations (.getField Bar "b")))))))

(deftest test-annotations-on-method
 (is (= expected-annotations
 (into #{}
 (map annotation->map
 (.getAnnotations (.getMethod Bar "foo" nil)))))))
```

---

## 12.49 test/examples.clj

— test/examples.clj —

```
\getchunk{Clojure Copyright}

(ns ^{:doc "Test classes that are AOT-compile for the tests in
 clojure.test-clojure.genclass."
 :author "Stuart Halloway, Daniel Solano Gmez"}
 clojure.test-clojure.genclass.examples)

(definterface ExampleInterface
 (foo [a])
 (foo [a b])
 (foo [a #^int b]))

(gen-class :name clojure.test_clojure.genclass.examples.ExampleClass
 :implements
 [clojure.test_clojure.genclass.examples.ExampleInterface])

;; -foo-Object unimplemented to test missing fn case

(defn -foo-Object-Object
 [_ o1 o2]
 "foo with o, o")

(defn -foo-Object-int
 [_ o i]
 "foo with o, i")

(gen-class :name
 ^{:Deprecated {}})
```

```

SuppressWarnings ["Warning1"] ; discarded
java.lang.annotation.Target []
clojure.test_clojure.genclass.examples.ExampleAnnotationClass
:prefix "annot-"
:methods [[^{:Deprecated {}}
 Override {} ;discarded
 foo [^{:java.lang.annotation.Retention
 java.lang.annotation.RetentionPolicy/SOURCE
 java.lang.annotation.Target
 [java.lang.annotation.ElementType/TYPE
 java.lang.annotation.ElementType/PARAMETER]}]
 String] void]])

```

---

## 12.50 test/io.clj

— test/io.clj —

```

\getchunk{Clojure Copyright}

(ns clojure.test-clojure.java.io
 (:use clojure.test clojure.java.io
 [clojure.test-helper :only [platform-newlines]])
 (:import (java.io File BufferedInputStream
 FileInputStream InputStreamReader InputStream
 FileOutputStream OutputStreamWriter OutputStream
 ByteArrayInputStream ByteArrayOutputStream)
 (java.net URL URI Socket ServerSocket)))

(defn temp-file
 [prefix suffix]
 (doto (File/createTempFile prefix suffix)
 (.deleteOnExit)))

(deftest test-spit-and-slurp
 (let [f (temp-file "clojure.java.io" "test")]
 (spit f "foobar")
 (is (= "foobar" (slurp f)))
 (spit f "foobar" :encoding "UTF-16")
 (is (= "foobar" (slurp f :encoding "UTF-16")))
 (testing "deprecated arity"
 (is (= (platform-newlines
 (with-out-str
 (is (= "foobar" (slurp f "UTF-16")))))))))
 "WARNING: (slurp f enc) is deprecated, use (slurp f :encoding enc).\n"
 (with-out-str
 (is (= "foobar" (slurp f "UTF-16"))))))))

```

```
(deftest test-streams-defaults
 (let [f (temp-file "clojure.java.io" "test-reader-writer")
 content "testing"]
 (try
 (is (thrown? Exception (reader (Object()))))
 (is (thrown? Exception (writer (Object()))))

 (are [write-to read-from]
 (= content (do
 (spit write-to content :encoding "UTF-8")
 (slurp read-from :encoding "UTF-8"))))
 f f
 (.getAbsolutePath f) (.getAbsolutePath f)
 (.toURL f) (.toURL f)
 (.toURI f) (.toURI f)
 (FileOutputStream. f) (FileInputStream. f)
 (OutputStreamWriter. (FileOutputStream. f) "UTF-8")
 (reader f :encoding "UTF-8")
 f (FileInputStream. f)
 (writer f :encoding "UTF-8")
 (InputStreamReader. (FileInputStream. f) "UTF-8"))

 (is (= content (slurp (.getBytes content "UTF-8"))))
 (is (= content (slurp (.toCharArray content)))))
 (finally
 (.delete f)))))

 (defn bytes-should-equal [byte-array-1 byte-array-2 msg]
 (is (= (clojure.java.io/byte-array-type (class byte-array-1)
 (class byte-array-2)) msg)
 (is (= (into [] byte-array-1) (into [] byte-array-2)) msg)))

 (defn data-fixture
 "in memory fixture data for tests"
 [encoding]
 (let [bs (.getBytes "hello" encoding)
 cs (.toCharArray "hello")
 i (ByteArrayInputStream. bs)
 r (InputStreamReader. i)
 o (ByteArrayOutputStream.)
 w (OutputStreamWriter. o)]
 {:bs bs
 :i i
 :r r
 :o o
 :s "hello"
 :cs cs
 :w w})))
```

```
(deftest test-copy
 (dorun
 (for [{:keys [in out flush] :as test}
 {:in :i :out :o}
 {:in :i :out :w}
 {:in :r :out :o}
 {:in :r :out :w}
 {:in :cs :out :o}
 {:in :cs :out :w}
 {:in :bs :out :o}
 {:in :bs :out :w}]

 opts
 [{} {:buffer-size 256}])
 (let [{:keys [s o] :as d} (data-fixture "UTF-8")]
 (apply copy (in d) (out d) (flatten (vec opts)))
 #_(when (= out :w) (.flush (:w d)))
 (.flush (out d))
 (bytes-should-equal (.getBytes s "UTF-8")
 (.toByteArray o)
 (str "combination " test opts)))))

(deftest test-copy-encodings
 (testing "from inputstream UTF-16 to writer UTF-8"
 (let [{:keys [i s w bs]} (data-fixture "UTF-16")]
 (copy i w :encoding "UTF-16")
 (.flush w)
 (bytes-should-equal (.getBytes s "UTF-8") (.toByteArray o) ""))
 (testing "from reader UTF-8 to output-stream UTF-16"
 (let [{:keys [r o s]} (data-fixture "UTF-8")]
 (copy r o :encoding "UTF-16")
 (bytes-should-equal (.getBytes s "UTF-16") (.toByteArray o) "")))

(deftest test-as-file
 (are [result input] (= result (as-file input))
 (File. "foo") "foo"
 (File. "bar") (File. "bar")
 (File. "baz") (URL. "file:baz")
 (File. "quux") (URI. "file:quux")
 nil nil))

(deftest test-file
 (are [result args] (= (File. result) (apply file args))
 "foo" ["foo"]
 "foo/bar" ["foo" "bar"]
 "foo/bar/baz" ["foo" "bar" "baz"]))
(deftest test-as-url
 (are [file-part input]
 (= (URL. (str "file:" file-part)) (as-url input))
 "foo" "file:foo"))
```

```

"baz" (URL. "file:baz")
"quux" (URI. "file:quux"))
(is (nil? (as-url nil)))))

(deftest test-delete-file
 (let [file (temp-file "test" "deletion")
 not-file (File. (str (java.util.UUID/randomUUID)))]
 (delete-file (.getAbsolutePath file))
 (is (not (.exists file)))
 (is (thrown? java.io.IOException (delete-file not-file)))
 (is (= :silently (delete-file not-file :silently)))))

(deftest test-as-relative-path
 (testing "strings"
 (is (= "foo" (as-relative-path "foo"))))
 (testing "absolute path strings are forbidden"
 (is (thrown? IllegalArgumentException
 (as-relative-path (.getAbsolutePath (File. "baz"))))))
 (testing "relative File paths"
 (is (= "bar" (as-relative-path (File. "bar")))))
 (testing "absolute File paths are forbidden"
 (is (thrown? IllegalArgumentException
 (as-relative-path
 (File. (.getAbsolutePath (File. "quux"))))))))

(defn stream-should-have [stream expected-bytes msg]
 (let [actual-bytes (byte-array (alength expected-bytes))]
 (.read stream actual-bytes)
 (is (= -1 (.read stream)) (str msg " : should be end of stream"))
 (is (= (seq expected-bytes) (seq actual-bytes))
 (str msg " : byte arrays should match"))))

(deftest test-input-stream
 (let [file (temp-file "test-input-stream" "txt")
 bytes (.getBytes "foobar")]
 (spit file "foobar")
 (doseq [[expr msg]
 [[file File]
 [(FileInputStream. file) FileInputStream]
 [(BufferedInputStream. (FileInputStream. file))
 BufferedInputStream]
 [(.. file toURI) URI]
 [(.. file toURI toURL) URL]
 [(.. file toURI toURL toString) "URL as String"]
 [(.. file toString) "File as String"]]]
 (with-open [s (input-stream expr)]
 (stream-should-have s bytes msg)))))

(deftest test-streams-buffering
 (let [data (.getBytes "")]

```

```

(is (instance? java.io.BufferedReader
 (reader data)))
(is (instance? java.io.BufferedWriter
 (writer (java.io.ByteArrayOutputStream.))))
(is (instance? java.io.BufferedInputStream
 (input-stream data)))
(is (instance? java.io.BufferedOutputStream
 (output-stream (java.io.ByteArrayOutputStream.)))))

(deftest test-resource
 (is (nil? (resource "non/existent/resource")))
 (is (instance? URL (resource "clojure/core.clj"))))
 (let [file (temp-file "test-resource" "txt")
 url (as-url (.getParentFile file))
 loader (java.net.URLClassLoader. (into-array [url]))]
 (is (nil? (resource "non/existent/resource" loader)))
 (is (instance? URL (resource (.getName file) loader)))))

(deftest test-make-parents
 (let [tmp (System/getProperty "java.io.tmpdir")]
 (delete-file
 (file tmp "test-make-parents" "child" "grandchild") :silently)
 (delete-file
 (file tmp "test-make-parents" "child") :silently)
 (delete-file
 (file tmp "test-make-parents") :silently)
 (make-parents tmp "test-make-parents" "child" "grandchild")
 (is (.isDirectory (file tmp "test-make-parents" "child")))
 (is (not
 (.isDirectory
 (file tmp "test-make-parents" "child" "grandchild"))))
 (delete-file (file tmp "test-make-parents" "child"))
 (delete-file (file tmp "test-make-parents"))))

(deftest test-socket-iofactory
 (let [port 65321
 server-socket (ServerSocket. port)
 client-socket (Socket. "localhost" port)]
 (try
 (is (instance? InputStream (input-stream client-socket)))
 (is (instance? OutputStream (output-stream client-socket)))
 (finally (.close server-socket)
 (.close client-socket)))))

 ——————

```

## 12.51 test/javadoc.clj

— test/javadoc.clj —

```
\getchunk{Clojure Copyright}

(ns clojure.test-clojure.java.javadoc
 (:use clojure.test
 [clojure.java.javadoc :as j])
 (:import (java.io File)))

(deftest javadoc-url-test
 (testing "for a core api"
 (binding [*feeling-lucky* false]
 (are [x y] (= x (#'j/javadoc-url y))
 nil "foo.Bar"
 (str *core-java-api* "java/lang/String.html"
 "java.lang.String")))
 (testing "for a remote javadoc"
 (binding [*remote-javadocs*
 (ref (sorted-map "java." "http://example.com/"))]
 (is (= "http://example.com/java/lang/Number.html"
 (#'j/javadoc-url "java.lang.Number"))))))
```

---

## 12.52 test/shell.clj

— test/shell.clj —

```
\getchunk{Clojure Copyright}

(ns clojure.test-clojure.java.shell
 (:use clojure.test
 [clojure.java.shell :as sh])
 (:import (java.io File)))

(def platform-enc (.name (java.nio.charset.Charset/defaultCharset)))
(def default-enc "UTF-8")

(deftest test-parse-args
 (are [x y] (= x y)
 [] {:in-enc default-enc
 :out-enc default-enc
 :dir nil}
```

```

:env nil}] (#'sh/parse-args [])
[["ls"] {:in-enc default-enc
 :out-enc default-enc
 :dir nil
 :env nil}] (#'sh/parse-args ["ls"])
[["ls" "-l"] {:in-enc default-enc
 :out-enc default-enc
 :dir nil
 :env nil}] (#'sh/parse-args ["ls" "-l"])
[["ls"] {:in-enc default-enc
 :out-enc "ISO-8859-1"
 :dir nil
 :env nil}]
 (#'sh/parse-args ["ls" :out-enc "ISO-8859-1"])
[] {:in-enc platform-enc
 :out-enc platform-enc
 :dir nil
 :env nil}]
(#'sh/parse-args [:in-enc platform-enc :out-enc platform-enc])))

(deftest test-with-sh-dir
 (are [x y] (= x y)
 nil *sh-dir*
 "foo" (with-sh-dir "foo" *sh-dir*)))

(deftest test-with-sh-env
 (are [x y] (= x y)
 nil *sh-env*
 {:KEY "VAL"} (with-sh-env {:KEY "VAL"} *sh-env*)))

(deftest test-as-env-strings
 (are [x y] (= x y)
 nil (#'sh/as-env-strings nil)
 ["FOO=BAR"]
 (seq (#'sh/as-env-strings {"FOO" "BAR"})))
 ["FOO_SYMBOL=BAR"]
 (seq (#'sh/as-env-strings {'FOO_SYMBOL "BAR"})))
 ["FOO_KEYWORD=BAR"]
 (seq (#'sh/as-env-strings {:FOO_KEYWORD "BAR"}))))
```

---

## 12.53 test/test'cl'format.clj

— test/test'cl'format.clj —

```
\getchunk{Clojure Copyright}

;; test_cl_format.clj -- part of the pretty printer for Clojure

;; Author: Tom Faulhaber
;; April 3, 2009

;; This test set tests the basic cl-format functionality

(in-ns 'clojure.test-clojure pprint)

(def format cl-format)

;; TODO tests for ~A, ~D, etc.
;; TODO add tests for ~F, etc.: 0.0, 9.9999 with rounding,
;; 9.9999E99 with rounding

(simple-tests d-tests
 (cl-format nil "~D" 0) "0"
 (cl-format nil "~D" 2e6) "2000000"
 (cl-format nil "~D" 2000000) "2000000"
 (cl-format nil "~:D" 2000000) "2,000,000"
 (cl-format nil "~D" 1/2) "1/2"
 (cl-format nil "~D" 'fred) "fred"
)

(simple-tests base-tests
 (cl-format nil "~{~2r~~ ~}~%" (range 10))
 "0 1 10 11 100 101 110 111 1000 1001\n"
 (with-out-str
 (dotimes [i 35]
 (binding [*print-base* (+ i 2)] ;print the decimal number 40
 (write 40) ;in each base from 2 to 36
 (if (zero? (mod i 10)) (prn) (cl-format true " "))))
 "101000
1111 220 130 104 55 50 44 40 37 34
31 2c 2a 28 26 24 22 20 1j 1i
1h 1g 1f 1e 1d 1c 1b 1a 19 18
17 16 15 14 "
 (with-out-str
 (doseq [pb [2 3 8 10 16]]
 (binding [*print-radix* true ;print the integer 10 and
 print-base pb] ;the ratio 1/10 in bases 2,
 (cl-format true "~~S ~S~%" 10 1/10))) ;3, 8, 10, 16
 "#b1010 #b1/1010\n#3r101 #3r1/101\n#o12
#o1/12\n10. #10r1/10\n#xa #x1/a\n")
)
)
)
```

```
(simple-tests cardinal-tests
 (cl-format nil "~R" 0) "zero"
 (cl-format nil "~R" 4) "four"
 (cl-format nil "~R" 15) "fifteen"
 (cl-format nil "~R" -15) "minus fifteen"
 (cl-format nil "~R" 25) "twenty-five"
 (cl-format nil "~R" 20) "twenty"
 (cl-format nil "~R" 200) "two hundred"
 (cl-format nil "~R" 203) "two hundred three"

 (cl-format nil "~R" 44879032)
 "forty-four million, eight hundred seventy-nine thousand, thirty-two"

 (cl-format nil "~R" -44879032)
 (str "minus forty-four million, eight hundred seventy-nine "
 "thousand, thirty-two")

 (cl-format nil "~R = ~:.*~:D" 44000032)
 "forty-four million, thirty-two = 44,000,032"

 (cl-format nil "~R = ~:.*~:D"
 448790329480948209384389429384029384029842098420989842094)
 (str
 "four hundred forty-eight septendecillion, seven hundred ninety "
 "sexdecillion, three hundred twenty-nine quindecillion, four hundred "
 "eighty quattuordecillion, nine hundred forty-eight tredecillion, "
 "two hundred nine duodecillion, three hundred eighty-four undecillion, "
 "three hundred eighty-nine decillion, four hundred twenty-nine "
 "nonillion, three hundred eighty-four octillion, twenty-nine "
 "septillion, three hundred eighty-four sextillion, twenty-nine "
 "quintillion, eight hundred forty-two quadrillion, ninety-eight "
 "trillion, four hundred twenty billion, nine hundred eighty-nine "
 "million, eight hundred forty-two thousand, ninety-four = 448,790,"
 "329,480,948,209,384,389,429,384,029,384,029,842,098,420,989,842,094")

 (cl-format nil "~R = ~:.*~:D"
 (+ (* 448790329480948209384389429384029384029842098420
 1000)
 989842094490320942058747587584758375847593475))
 (str "448,790,329,480,948,209,384,389,429,384,029,384,029,842,098,"
 "420,989,842,094,490,320,942,058,747,587,584,758,375,847,593,"
 "475 = 448,790,329,480,948,209,384,389,429,384,029,384,029,842,"
 "098,420,989,842,094,490,320,942,058,747,587,584,758,375,847,"
 "593,475")

 (cl-format nil "~R = ~:.*~:D" 2e6)
 "two million = 2,000,000"

 (cl-format nil "~R = ~:.*~:D" 200000200000)
 "two hundred billion, two hundred thousand = 200,000,200,000")
```

```
(simple-tests ordinal-tests
 (cl-format nil "~:R" 0) "zeroth"
 (cl-format nil "~:R" 4) "fourth"
 (cl-format nil "~:R" 15) "fifteenth"
 (cl-format nil "~:R" -15) "minus fifteenth"
 (cl-format nil "~:R" 25) "twenty-fifth"
 (cl-format nil "~:R" 20) "twentieth"
 (cl-format nil "~:R" 200) "two hundredth"
 (cl-format nil "~:R" 203) "two hundred third"

 (cl-format nil "~:R" 44879032)
 (str "forty-four million, eight hundred seventy-nine thousand,"
 " thirty-second"

 (cl-format nil "~:R" -44879032)
 (str "minus forty-four million, eight hundred seventy-nine "
 "thousand, thirty-second"

 (cl-format nil "~:R = ~:*~:D" 44000032)
 "forty-four million, thirty-second = 44,000,032"

 (cl-format nil "~:R = ~:*~:D"
 448790329480948209384389429384029842098420989842094)
 (str "four hundred forty-eight septendecillion, seven hundred ninety "
 "sexdecillion, three hundred twenty-nine quindecillion, four hundred "
 "eighty quattuordecillion, nine hundred forty-eight tredecillion, two "
 "hundred nine duodecillion, three hundred eighty-four undecillion, "
 "three hundred eighty-nine decillion, four hundred twenty-nine "
 "nonillion, three hundred eighty-four octillion, twenty-nine "
 "septillion, three hundred eighty-four sextillion, twenty-nine "
 "quintillion, eight hundred forty-two quadrillion, ninety-eight "
 "trillion, four hundred twenty billion, nine hundred eighty-nine "
 "million, eight hundred forty-two thousand, ninety-fourth = "
 "448,790,329,480,948,209,384,389,429,384,029,384,029,842,098,420,"
 "989,842,094")
 (cl-format nil "~:R = ~:*~:D"
 (+ (* 448790329480948209384389429384029842098420
 1000)
 989842094490320942058747587584758375847593475))
 (str "448,790,329,480,948,209,384,389,429,384,029,384,029,842,098,"
 "420,989,842,094,490,320,942,058,747,587,584,758,375,847,593,"
 "475th = 448,790,329,480,948,209,384,389,429,384,029,384,029,"
 "842,098,420,989,842,094,490,320,942,058,747,587,584,758,375,"
 "847,593,475"
 (cl-format nil "~:R = ~:*~:D"
 (+ (* 448790329480948209384389429384029842098420
 1000)
 989842094490320942058747587584758375847593471))
 "448,790,329,480,948,209,384,389,429,384,029,384,029,842,098,420,"
```

```

"989,842,094,490,320,942,058,747,587,584,758,375,847,593,471st = "
"448,790,329,480,948,209,384,389,429,384,029,384,029,842,098,420,"
"989,842,094,490,320,942,058,747,587,584,758,375,847,593,471"
(cl-format nil "~:R = ~*:~:D" 2e6)
"two millionth = 2,000,000")

(simple-tests ordinal1-tests
 (cl-format nil "~:R" 1) "first"
 (cl-format nil "~:R" 11) "eleventh"
 (cl-format nil "~:R" 21) "twenty-first"
 (cl-format nil "~:R" 20) "twentieth"
 (cl-format nil "~:R" 220) "two hundred twentieth"
 (cl-format nil "~:R" 200) "two hundredth"
 (cl-format nil "~:R" 999) "nine hundred ninety-ninth"
)

(simple-tests roman-tests
 (cl-format nil "~@R" 3) "III"
 (cl-format nil "~@R" 4) "IV"
 (cl-format nil "~@R" 9) "IX"
 (cl-format nil "~@R" 29) "XXIX"
 (cl-format nil "~@R" 429) "CDXXXIX"
 (cl-format nil "~@:R" 429) "CCCCXXVIIII"
 (cl-format nil "~@:R" 3429) "MMMCCCCXXVIIII"
 (cl-format nil "~@R" 3429) "MMMCDXXIX"
 (cl-format nil "~@R" 3479) "MMMCDLXXIX"
 (cl-format nil "~@R" 3409) "MMMCDIX"
 (cl-format nil "~@R" 300) "CCC"
 (cl-format nil "~@R ~D" 300 20) "CCC 20"
 (cl-format nil "~@R" 5000) "5,000"
 (cl-format nil "~@R ~D" 5000 20) "5,000 20"
 (cl-format nil "~@R" "the quick") "the quick")

(simple-tests c-tests
 (cl-format nil "~{~c~~, ~}~%" "hello") "h, e, l, l, o\n"
 (cl-format nil "~{:c~~, ~}~%" "hello") "h, e, l, l, o\n"
 (cl-format nil "~@C~%" \m) "\\m\n"
 (cl-format nil "~@C~%" (char 222)) "\\\n"
 (cl-format nil "~@C~%" (char 8)) "\\backspace\n")
;tpd the control-C won't pass latex.
;tpd I need some latex magic to fix it.
; (cl-format nil "~@C~%" (char 3)) "\\ctrl-\n")

(simple-tests e-tests
 (cl-format nil "*~E*" 0.0) "*0.0E+0*"
 (cl-format nil "*~6E*" 0.0) "*0.0E+0*"
 (cl-format nil "*~6,0E*" 0.0) "* 0.E+0*"
 (cl-format nil "*~7,2E*" 0.0) "*0.00E+0*"
 (cl-format nil "*~5E*" 0.0) "*0.E+0*"
 (cl-format nil "*~10,2,2,,?E*" 2.8E120) "*??????????*"

```

```
(cl-format nil "*~10,2E*" 9.99999) "* 1.00E+1*"
(cl-format nil "*~10,2E*" 9.99999E99) "* 1.00E+100*"
(cl-format nil "*~10,2,2E*" 9.99999E99) "* 1.00E+100*"
(cl-format nil "*~10,2,2,,?E*" 9.99999E99) "*????????????*"
)

(simple-tests $-tests
 (cl-format nil "~$" 22.3) "22.30"
 (cl-format nil "~$" 22.375) "22.38"
 (cl-format nil "~3,5$" 22.375) "00022.375"
 (cl-format nil "~3,5,8$" 22.375) "00022.375"
 (cl-format nil "~3,5,10$" 22.375) " 00022.375"
 (cl-format nil "~3,5,14@ $" 22.375) " +00022.375"
 (cl-format nil "~3,5,14@ $" 22.375) " +00022.375"
 (cl-format nil "~3,5,14@:$" 22.375) "+ 00022.375"
 (cl-format nil "~3,,14@:$" 0.375) "+ 0.375"
 (cl-format nil "~1,1,$" -12.0) "-12.0"
 (cl-format nil "~1,1$" 12.0) "12.0"
 (cl-format nil "~1,1$" 12.0) "12.0"
 (cl-format nil "~1,1@ $" 12.0) "+12.0"
 (cl-format nil "~1,1,8,'@:$" 12.0) "+ 12.0"
 (cl-format nil "~1,1,8,'@ $" 12.0) " +12.0"
 (cl-format nil "~1,1,8,':$" 12.0) " 12.0"
 (cl-format nil "~1,1,8,' $" 12.0) " 12.0"
 (cl-format nil "~1,1,8,'@:$" -12.0) "- 12.0"
 (cl-format nil "~1,1,8,'@ $" -12.0) " -12.0"
 (cl-format nil "~1,1,8,':$" -12.0) "- 12.0"
 (cl-format nil "~1,1,8,' $" -12.0) " -12.0"
 (cl-format nil "~1,1$" 0.001) "0.0"
 (cl-format nil "~2,1$" 0.001) "0.00"
 (cl-format nil "~1,1,6$" 0.001) " 0.0"
 (cl-format nil "~1,1,6$" 0.0015) " 0.0"
 (cl-format nil "~2,1,6$" 0.005) " 0.01"
 (cl-format nil "~2,1,6$" 0.01) " 0.01"
 (cl-format nil "~$" 0.099) "0.10"
 (cl-format nil "~1$" 0.099) "0.1"
 (cl-format nil "~1$" 0.1) "0.1"
 (cl-format nil "~1$" 0.99) "1.0"
 (cl-format nil "~1$" -0.99) "-1.0")

(simple-tests f-tests
 (cl-format nil "~,1f" -12.0) "-12.0"
 (cl-format nil "~,0f" 9.4) "9."
 (cl-format nil "~,0f" 9.5) "10."
 (cl-format nil "~,0f" -0.99) "-1."
 (cl-format nil "~,1f" -0.99) "-1.0"
 (cl-format nil "~,2f" -0.99) "-0.99"
 (cl-format nil "~,3f" -0.99) "-0.990"
 (cl-format nil "~,0f" 0.99) "1."
 (cl-format nil "~,1f" 0.99) "1.0"
```

```
(cl-format nil "~,2f" 0.99) "0.99"
(cl-format nil "~,3f" 0.99) "0.990"
(cl-format nil "~f" -1) "-1.0"
(cl-format nil "~2f" -1) "-1."
(cl-format nil "~3f" -1) "-1."
(cl-format nil "~4f" -1) "-1.0"
(cl-format nil "~8f" -1) "-1.0"
(cl-format nil "~1,1f" 0.1) ".1")

(simple-tests ampersand-tests
 (cl-format nil
 "The quick brown ~a jumped over ~d lazy dogs" 'elephant 5)
 "The quick brown elephant jumped over 5 lazy dogs"
 (cl-format nil
 "The quick brown ~&~a jumped over ~d lazy dogs" 'elephant 5)
 "The quick brown \nelephant jumped over 5 lazy dogs"
 (cl-format nil (platform-newlines
 "The quick brown ~&~a jumped\n~& over ~d lazy dogs") 'elephant 5)
 "The quick brown \nelephant jumped\n over 5 lazy dogs"
 (cl-format nil (platform-newlines
 "'~&The quick brown ~&~a jumped\n~& over ~d lazy dogs") 'elephant 5)
 "The quick brown \nelephant jumped\n over 5 lazy dogs"
 (cl-format nil (platform-newlines
 "'~3&The quick brown ~&~a jumped\n~& over ~d lazy dogs") 'elephant 5)
 "\n\nThe quick brown \nelephant jumped\n over 5 lazy dogs"
 (cl-format nil
 "'~@{~&The quick brown ~a jumped over ~d lazy dogs~}"'
 'elephant 5 'fox 10)
 (str "The quick brown elephant jumped over 5 lazy dogs\n"
 "The quick brown fox jumped over 10 lazy dogs")
 (cl-format nil "I ~[don't ~:;d~&o ~]have one~%" 0) "I don't have one\n"
 (cl-format nil "I ~[don't ~:;d~&o ~]have one~%" 1) "I d\nno have one\n")

(simple-tests t-tests
 (cl-format nil "'~@{~&~A~8,4T~:*~A~}"'
 'a 'aa 'aaa 'aaaa 'aaaaaa 'aaaaaaaa
 'aaaaaaaaa 'aaaaaaaaa 'aaaaaaaaaa)
 (str "a a\ntaa aa\ntaaa aaa\ntaaaa aaaa\ntaaaaa "
 "aaaaa\ntaaaaa aaaaa\ntaaaaaa aaaaaaa\ntaaaaaaa "
 "aaaaaaaa\ntaaaaaaaaa aaaaaaaaa\ntaaaaaaaaaa aaaaaaaaaaa")
 (cl-format nil "'~@{~&~A~,4T~:*~A~}"'
 'a 'aa 'aaa 'aaaa 'aaaaaa 'aaaaaaaa 'aaaaaaaaaa
 'aaaaaaaaaa)
 (str "a a\ntaa aa\ntaaa aaa\ntaaaa aaaa\ntaaaaa "
 "aaaaa\ntaaaaa aaaaaaa\ntaaaaaaa aaaaaaaa\ntaaaaaaa "
 "aaaaaaaa\ntaaaaaaaaa aaaaaaaaa\ntaaaaaaaaaa")
 (cl-format nil "'~@{~&~A~2,6@T~:*~A~}"'
 'a 'aa 'aaa 'aaaa 'aaaaaa 'aaaaaaaa 'aaaaaaaaaa)
 (str "a a\ntaa aa\ntaaa aaa\ntaaaa aaaa\ntaaaaa "
 "aaaaa\ntaaaaa aaaaaaa\ntaaaaaa aaaaaaa\ntaaaaaaa")
```

```

 " aaaaaaaaa\naaaaaaaaaa aaaaaaaaaa\naaaaaaaaaaaaa aaaaaaaaaaaa")
)

(simple-tests paren-tests
 (cl-format nil "~(PLEASE SPEAK QUIETLY IN HERE~)")
 "please speak quietly in here"
 (cl-format nil "~@(PLEASE SPEAK QUIETLY IN HERE~)")
 "Please speak quietly in here"
 (cl-format nil "~@:(but this Is importTant~)")
 "BUT THIS IS IMPORTANT"
 (cl-format nil "~:(the greAt gatsby~)!")
 "The Great Gatsby!"
 ;; Test cases from CLtL 18.3 - string-upcase, et al.
 (cl-format nil "~@:(~A~)" "Dr. Livingstone, I presume?")
 "DR. LIVINGSTONE, I PRESUME?"
 (cl-format nil "~(~A~)" "Dr. Livingstone, I presume?")
 "dr. livingstone, i presume?"
 (cl-format nil "~:(~A~)" " hello ")
 " Hello "
 (cl-format nil "~:(~A~)"
 "occludED cASEmenTs FOreSTALL iNADVertent DEFenestraTION")
 "Occluded Casements Forestall Inadvertent Defenestration"
 (cl-format nil "~:(~A~)" 'kludgy-hash-search) "Kludgy-Hash-Search"
 (cl-format nil "~:(~A~)" "DON'T!") "Don'T!" ;not "Don't!"
 (cl-format nil "~:(~A~)" "pipe 13a, foo16c") "Pipe 13a, Foo16c"
)

(simple-tests square-bracket-tests
 ;; Tests for format without modifiers
 (cl-format nil "I ~[don't ~]have one~%" 0) "I don't have one\n"
 (cl-format nil "I ~[don't ~]have one~%" 1) "I have one\n"
 (cl-format nil "I ~[don't ~;do ~]have one~%" 0) "I don't have one\n"
 (cl-format nil "I ~[don't ~;do ~]have one~%" 1) "I do have one\n"
 (cl-format nil "I ~[don't ~;do ~]have one~%" 2) "I have one\n"
 (cl-format nil "I ~[don't ~:;do ~]have one~%" 0) "I don't have one\n"
 (cl-format nil "I ~[don't ~:;do ~]have one~%" 1) "I do have one\n"
 (cl-format nil "I ~[don't ~:;do ~]have one~%" 2) "I do have one\n"
 (cl-format nil "I ~[don't ~:;do ~]have one~%" 700) "I do have one\n"

 ;; Tests for format with a colon
 (cl-format nil "I ~:[don't ~;do ~]have one~%" true) "I do have one\n"
 (cl-format nil "I ~:[don't ~;do ~]have one~%" 700) "I do have one\n"
 (cl-format nil "I ~:[don't ~;do ~]have one~%" '(a b)) "I do have one\n"
 (cl-format nil "I ~:[don't ~;do ~]have one~%" nil) "I don't have one\n"
 (cl-format nil "I ~:[don't ~;do ~]have one~%" false)
 "I don't have one\n"

 ;; Tests for format with an at sign
 (cl-format nil "We had ~D wins~@[(out of ~D tries)~].~%" 15 nil)
 "We had 15 wins.\n"
)

```

```
(cl-format nil "We had ~D wins~@[(out of ~D tries)~].~%" 15 17)
"We had 15 wins (out of 17 tries).\n"

;; Format tests with directives
(cl-format nil
"Max ~D: ~[Blue team ~D~;Red team ~D~:;No team ~A~].~%" 15, 0, 7)
"Max 15: Blue team 7.\n"
(cl-format nil
"Max ~D: ~[Blue team ~D~;Red team ~D~:;No team ~A~].~%" 15, 1, 12)
"Max 15: Red team 12.\n"
(cl-format nil
"Max ~D: ~[Blue team ~D~;Red team ~D~:;No team ~A~].~%" 15, -1, "(system failure)")
"Max 15: No team (system failure).\n"

;; Nested format tests
(cl-format nil
(str "Max ~D: ~[Blue team ~D~:[~; (complete success)~]~;""
 "Red team ~D~:;No team ~].~%")
 15, 0, 7, true)
"Max 15: Blue team 7 (complete success).\n"
(cl-format nil
(str "Max ~D: ~[Blue team ~D~:[~; (complete success)~]~;""
 "Red team ~D~:;No team ~].~%")
 15, 0, 7, false)
"Max 15: Blue team 7.\n"

;; Test the selector as part of the argument
(cl-format nil
"The answer is ~#[nothing~;~D~;~D out of ~D~:;something crazy~].")
"The answer is nothing."
(cl-format nil
"The answer is ~#[nothing~;~D~;~D out of ~D~:;something crazy~]. 4")
"The answer is 4."
(cl-format nil
"The answer is ~#[nothing~;~D~;~D out of ~D~:;something crazy~]."
 7 22)
"The answer is 7 out of 22."
(cl-format nil
"The answer is ~#[nothing~;~D~;~D out of ~D~:;something crazy~]."
 1 2 3 4)
"The answer is something crazy."
)

(simple-tests curly-brace-plain-tests
;; Iteration from sublist
(cl-format nil "Coordinates are~{ [~D,~D]~}~%""
 [0, 1, 1, 0, 3, 5, 2, 1])
"Coordinates are [0,1] [1,0] [3,5] [2,1]\n"
```

```
(cl-format nil "Coordinates are~2{ [~D,~D]~}~%"
 [0, 1, 1, 0, 3, 5, 2, 1])
"Coordinates are [0,1] [1,0]\n"

(cl-format nil "Coordinates are~{ ~#[none~; <~D>~:; [~D,~D]~]~}~%"
 [])
"Coordinates are\n"

(cl-format nil "Coordinates are~{ ~#[none~; <~D>~:; [~D,~D]~]~:}~%"
 [])
"Coordinates are none\n"

(cl-format nil "Coordinates are~{ ~#[none~; <~D>~:; [~D,~D]~]~:}~%"
 [2 3 1])
"Coordinates are [2,3] <1>\n"

(cl-format nil "Coordinates are~{~:}~%" "" [])
"Coordinates are\n"

(cl-format nil
 "Coordinates are~{~:}~%" " ~#[none~; <~D>~:; [~D,~D]~]" [2 3 1])
"Coordinates are [2,3] <1>\n"

(cl-format nil
 "Coordinates are~{~:}~%" " ~#[none~; <~D>~:; [~D,~D]~]" [])
"Coordinates are none\n"
)

(simple-tests curly-brace-colon-tests
;; Iteration from list of sublists
(cl-format nil "Coordinates are~:{ [~D,~D]~}~%"
 [[0, 1], [1, 0], [3, 5], [2, 1]])
"Coordinates are [0,1] [1,0] [3,5] [2,1]\n"

(cl-format nil "Coordinates are~:{ [~D,~D]~}~%"
 [[0, 1, 0], [1, 0, 12], [3, 5], [2, 1]])
"Coordinates are [0,1] [1,0] [3,5] [2,1]\n"

(cl-format nil "Coordinates are~2:{ [~D,~D]~}~%"
 [[0, 1], [1, 0], [3, 5], [2, 1]])
"Coordinates are [0,1] [1,0]\n"

(cl-format nil "Coordinates are~:{ ~#[none~; <~D>~:; [~D,~D]~]~:}~%"
 [])
"Coordinates are\n"

(cl-format nil "Coordinates are~:{ ~#[none~; <~D>~:; [~D,~D]~]~:}~%"
 [])
"Coordinates are none\n"
```

```
(cl-format nil "Coordinates are~:{ ~#[none~; <~D>~:; [~D,~D]~]~:}~%"
 [[2 3] [1]])
"Coordinates are [2,3] <1>\n"

(cl-format nil "Coordinates are~:{~:}~%" "" [])
"Coordinates are\n"

(cl-format nil "Coordinates are~:{~:}~%" " ~#[none~; <~D>~:; [~D,~D]~]"
 [[2 3] [1]])
"Coordinates are [2,3] <1>\n"

(cl-format nil "Coordinates are~:{~:}~%" " ~#[none~; <~D>~:; [~D,~D]~]"
 [])
"Coordinates are none\n"
)

(simple-tests curly-brace-at-tests
 ;;= Iteration from main list
 (cl-format nil "Coordinates are~@{ [~D,~D]~}~%"
 0, 1, 1, 0, 3, 5, 2, 1)
"Coordinates are [0,1] [1,0] [3,5] [2,1]\n"

(cl-format nil "Coordinates are~2@{ [~D,~D]~}~%"
 0, 1, 1, 0, 3, 5, 2, 1)
"Coordinates are [0,1] [1,0]\n"

(cl-format nil "Coordinates are~@{ ~#[none~; <~D>~:; [~D,~D]~]~:}~%"
"Coordinates are\n"

(cl-format nil "Coordinates are~@{ ~#[none~; <~D>~:; [~D,~D]~]~:}~%"
 2 3 1)
"Coordinates are [2,3] <1>\n"

(cl-format nil "Coordinates are~@{~:}~%" "")
"Coordinates are\n"

(cl-format nil "Coordinates are~@{~:}~%" " ~#[none~; <~D>~:; [~D,~D]~]"
 2 3 1)
"Coordinates are [2,3] <1>\n"

(cl-format nil "Coordinates are~@{~:}~%" " ~#[none~; <~D>~:; [~D,~D]~")
"Coordinates are none\n"
)

(simple-tests curly-brace-colon-at-tests
 ;;= Iteration from sublists on the main arg list
```

```
(cl-format nil "Coordinates are~@:{ [~D,~D]~}~%"
 [0, 1], [1, 0], [3, 5], [2, 1])
"Coordinates are [0,1] [1,0] [3,5] [2,1]\n"

(cl-format nil "Coordinates are~@:{ [~D,~D]~}~%"
 [0, 1, 0], [1, 0, 12], [3, 5], [2, 1])
"Coordinates are [0,1] [1,0] [3,5] [2,1]\n"

(cl-format nil "Coordinates are~20:{ [~D,~D]~}~%"
 [0, 1], [1, 0], [3, 5], [2, 1])
"Coordinates are [0,1] [1,0]\n"

(cl-format nil "Coordinates are~@:{ ~#[none~; <~D>~:; [~D,~D]~]~}~%")
"Coordinates are \n"

(cl-format nil "Coordinates are~@:{ ~#[none~; <~D>~:; [~D,~D]~]~:}~%")
"Coordinates are none\n"

(cl-format nil "Coordinates are~@:{ ~#[none~; <~D>~:; [~D,~D]~]~:}~%"
 [2 3] [1])
"Coordinates are [2,3] <1>\n"

(cl-format nil "Coordinates are~@:{~:}~%" "")
"Coordinates are \n"

(cl-format nil
 "Coordinates are~@:{~:}~%" " ~#[none~; <~D>~:; [~D,~D]~]" [2 3] [1])
"Coordinates are [2,3] <1>\n"
)

;; TODO tests for ~~ in ~[constructs and other brackets
;; TODO test ~:^ generates an error when used improperly
;; TODO test ~:^ works in ~@:{...~}
(let [aseq '(a quick brown fox jumped over the lazy dog)
 lseq (mapcat identity (for [x aseq] [x (.length (name x))]))]
(simple-tests up-tests
 (cl-format nil "~{~a~~, ~}" aseq)
 "a, quick, brown, fox, jumped, over, the, lazy, dog"
 (cl-format nil "~{~a~0~, ~}" aseq) "a"
 (cl-format nil "~{~a~#,3~, ~}" aseq)
 "a, quick, brown, fox, jumped, over"
 (cl-format nil "~{~a~v,3~, ~}" lseq) "a, quick, brown, fox"
 (cl-format nil "~{~a~3,v,4~, ~}" lseq) "a, quick, brown, fox")
)

(simple-tests angle-bracket-tests
```

```

(cl-format nil "~<foo~;bar~;baz~>")
"foobarbaz"
(cl-format nil "~20<foo~;bar~;baz~>")
"foo bar baz"
(cl-format nil "~,,2<foo~;bar~;baz~>")
"foo bar baz"
(cl-format nil "~20<~A~;~A~;~A~>" "foo" "bar" "baz")
"foo bar baz"
(cl-format nil "~20:<~A~;~A~;~A~>" "foo" "bar" "baz")
" foo bar baz"
(cl-format nil "~20@<~A~;~A~;~A~>" "foo" "bar" "baz")
"foo bar baz"
(cl-format nil "~20@:<~A~;~A~;~A~>" "foo" "bar" "baz")
" foo bar baz"
(cl-format nil "~10,,2<~A~;~A~;~A~>" "foo" "bar" "baz")
"foo bar baz"
(cl-format nil "~10,10,2<~A~;~A~;~A~>" "foo" "bar" "baz")
"foo bar baz"
(cl-format nil "~10,10<~A~;~A~;~A~>" "foo" "bar" "baz")
"foo barbaz"
(cl-format nil "~20<~A~;~~~A~;~~~A~>" "foo" "bar" "baz")
"foo bar baz"
(cl-format nil "~20<~A~;~~~A~;~~~A~>" "foo" "bar")
"foo bar"
(cl-format nil "~20@<~A~;~~~A~;~~~A~>" "foo")
"foo"
(cl-format nil "~20:<~A~;~~~A~;~~~A~>" "foo")
" foo"
)

(simple-tests angle-bracket-max-column-tests
 (cl-format nil
 "~~%; ~{~<~%; ~1,50:; ~A~>~}.~%"
 (into []
 (.split
 (str "This function computes the circular thermodynamic "
 "coefficient of the thrombulator angle for use in "
 "determining the reaction distance" "\s")))
 (str "\n;; This function computes the circular\n;; "
 "thermodynamic coefficient of the thrombulator\n;; "
 "angle for use in determining the reaction\n;; distance.\n")
 (cl-format true "~~%; ~{~<~%; ~:; ~A~>~}.~%"
 (into []
 (.split
 (str "This function computes the circular thermodynamic coefficient"
 " of the thrombulator angle for use in determining the reaction"
 " distance." "\s"))))

 (defn list-to-table [aseq column-width]
 (let [stream (get-prettier-writer (java.io.StringWriter.))]
 (binding [*out* stream]

```

```

(doseq [row aseq]
 (doseq [col row]
 (cl-format true "~4D~7,vT" col column-width))
 (prn)))
(.flush stream)
(.toString (:base @@(:base @@stream)))))

(simple-tests column-writer-test
 (list-to-table (map #(vector % (* % %) (* % % %)) (range 1 21)) 8)
 (str " 1 1 1 \n 2 4 8 \n 3 "
 " 9 27 \n 4 16 64 \n 5 25 125"
 " \n 6 36 216 \n 7 49 343 \n 8"
 " 64 512 \n 9 81 729 \n 10 100 "
 " 1000 \n 11 121 1331 \n 12 144 1728 \n"
 " 13 169 2197 \n 14 196 2744 \n 15 225"
 " 3375 \n 16 256 4096 \n 17 289 4913 "
 " \n 18 324 5832 \n 19 361 6859 \n 20 "
 "400 8000 \n"))
;;;;;;;;
;; The following tests are the various examples from the format
;; documentation in Common Lisp, the Language, 2nd edition, Chapter 22.3
;;;;;;;;
(defn expt [base pow] (reduce * (repeat pow base)))

(let [x 5, y "elephant", n 3]
 (simple-tests cltl-intro-tests
 (format nil "foo") "foo"
 (format nil "The answer is ~D." x) "The answer is 5."
 (format nil "The answer is ~3D." x) "The answer is 5."
 (format nil "The answer is ~3,'0D." x) "The answer is 005."
 (format nil "The answer is ~:D." (expt 47 x))
 "The answer is 229,345,007."
 (format nil "Look at the ~A!" y) "Look at the elephant!"
 (format nil "Type ~:C to ~A." (char 4) "delete all your files")
 "Type Control-D to delete all your files."
 (format nil "~D item~:P found." n) "3 items found."
 (format nil "~R dog~:[s are~; is~] here." n (= n 1))
 "three dogs are here."
 (format nil "~R dog~:*~[s are~; is~:;s are~] here." n)
 "three dogs are here."
 (format nil "Here ~[are~;is~:;are~] ~:*~R pupp~:@P." n)
 "Here are three puppies."))

(simple-tests cltl-B-tests
 ;; CLtL didn't have the colons here, but the spec requires them
 (format nil "~,,' ,4:B" 0xFACE) "1111 1010 1100 1110"
 (format nil "~,,' ,4:B" 0x1CE) "1 1100 1110"
 (format nil "~19,,,' ,4:B" 0xFACE) "1111 1010 1100 1110"
 ;; This one was a nice idea, but nothing in the spec supports it

```

```

;; working this way
;; (and SBCL doesn't work this way either)
;(format nil "~19,,,'4:B" 0x1CE) "0000 0001 1100 1110")
)

(simple-tests cltl-P-tests
 (format nil "~D tr~:@P/~D win~:P" 7 1) "7 tries/1 win"
 (format nil "~D tr~:@P/~D win~:P" 1 0) "1 try/0 wins"
 (format nil "~D tr~:@P/~D win~:P" 1 3) "1 try/3 wins")

(defn foo [x]
 (format nil "~6,2F|~6,2,1,*F|~6,2,,?F|~6F|~,2F|~F"
 x x x x x))

(simple-tests cltl-F-tests
 (foo 3.14159) " 3.14| 31.42| 3.14|3.1416|3.14|3.14159"
 (foo -3.14159) " -3.14|-31.42| -3.14|-3.142|-3.14|-3.14159"
 (foo 100.0) "100.00|*****|100.00| 100.0|100.00|100.0"
 (foo 1234.0) "1234.00|*****|??????|1234.0|1234.00|1234.0"
 (foo 0.006) " 0.01| 0.06| 0.01| 0.006|0.01|0.006")

(defn foo-e [x]
 (format nil
 "~9,2,1,,,*E|~10,3,2,2,'?,,'$E|~9,3,2,-2,'%@E|~9,2E"
 x x x x))

;; Clojure doesn't support float/double differences in representation
(simple-tests cltl-E-tests
 (foo-e 0.0314159) " 3.14E-2| 31.42$-03|+.003E+01| 3.14E-2"
 (foo-e 3.14159) " 3.14E+0| 31.42$-01|+.003E+03| 3.14E+0"
 (foo-e -3.14159) " -3.14E+0|-31.42$-01|-.003E+03| -3.14E+0"
 (foo-e 1100.0) " 1.10E+3| 11.00$+02|+.001E+06| 1.10E+3"
; In Clojure, this is identical to the above
; (foo-e 1100.0L) " 1.10L+3| 11.00$+02|+.001L+06| 1.10L+3"
 (foo-e 1.1E13) "*****| 11.00$+12|+.001E+16| 1.10E+13"
 (foo-e 1.1E120) "*****|?????????|%%%%%%%%|1.10E+120"
; Clojure doesn't support real numbers this large
; (foo-e 1.1L1200) "*****|?????????|%%%%%%%%|1.10L+1200"
)

(simple-tests cltl-E-scale-tests
 (map
 (fn [k] (format nil "Scale factor ~2D~*:|~13,6,2,VE|"
 (- k 5) 3.14159)) ;Prints 13 lines
 (range 13))
 '("Scale factor -5: | 0.000003E+06|"
 "Scale factor -4: | 0.000031E+05|"
 "Scale factor -3: | 0.000314E+04|"
 "Scale factor -2: | 0.003142E+03|"
 "Scale factor -1: | 0.031416E+02|"

```

```

"Scale factor 0: | 0.314159E+01|"
"Scale factor 1: | 3.141590E+00|"
"Scale factor 2: | 31.41590E-01|"
"Scale factor 3: | 314.1590E-02|"
"Scale factor 4: | 3141.590E-03|"
"Scale factor 5: | 31415.90E-04|"
"Scale factor 6: | 314159.0E-05|"
"Scale factor 7: | 3141590.E-06|"))

(defn foo-g [x]
 (format nil
 "~~9,2,1,,,*G|~9,3,2,3,'?,,,$G|~9,3,2,0,'%G|~9,2G"
 x x x x))

;; Clojure doesn't support float/double differences in representation
(simple-tests cltl-G-tests
 (foo-g 0.0314159) " 3.14E-2|314.2$-04|0.314E-01| 3.14E-2"
 (foo-g 0.314159) " 0.31 |0.314 |0.314 | 0.31 "
 (foo-g 3.14159) " 3.1 | 3.14 | 3.14 | 3.1 "
 (foo-g 31.4159) " 31. | 31.4 | 31.4 | 31. "
 (foo-g 314.159) " 3.14E+2| 314. | 314. | 3.14E+2"
 (foo-g 3141.59) " 3.14E+3|314.2$+01|0.314E+04| 3.14E+3"
; In Clojure, this is identical to the above
; (foo-g 3141.59L0) " 3.14L+3|314.2$+01|0.314L+04| 3.14L+3"
 (foo-g 3.14E12) "*****|314.0$+10|0.314E+13| 3.14E+12"
 (foo-g 3.14E120) "*****|?????????|%%%%%%%%|3.14E+120"
; Clojure doesn't support real numbers this large
; (foo-g 3.14L1200) "*****|?????????|%%%%%%%%|3.14L+1200"
)

(defn type-clash-error [fun nargs argnum right-type wrong-type]
 (format nil ;; CLtL has this format string slightly wrong
 "~~&Function ~S requires its ~:[~:R ~;~*~]~"
 "argument to be of type ~S,~%but it was called ~"
 "with an argument of type ~S.~%"
 fun (= nargs 1) argnum right-type wrong-type))

(simple-tests cltl-Newline-tests
 (type-clash-error 'aref nil 2 'integer 'vector)
 "Function aref requires its second argument to be of type integer,
 but it was called with an argument of type vector.\n"
 (type-clash-error 'car 1 1 'list 'short-float)
 "Function car requires its argument to be of type list,
 but it was called with an argument of type short-float.\n")

(simple-tests cltl-?-tests
 (format nil "? ~D" "<~A ~D>" '("Foo" 5) 7) "<Foo 5> 7"
 (format nil "? ~D" "<~A ~D>" '("Foo" 5 14) 7) "<Foo 5> 7"
 (format nil "?@? ~D" "<~A ~D>" "Foo" 5 7) "<Foo 5> 7"
 (format nil "?@? ~D" "<~A ~D>" "Foo" 5 14 7) "<Foo 5> 14")

```

```
(defn f [n] (format nil "~@(~R~) error~:P detected." n))

(simple-tests cltl-paren-tests
 (format nil "~@R ~(~R~)" 14 14) "XIV xiv"
 (f 0) "Zero errors detected."
 (f 1) "One error detected."
 (f 23) "Twenty-three errors detected.")

(let [*print-level* nil *print-length* 5]
 (simple-tests cltl-bracket-tests
 (format nil "~@[print level = ~D~]~@[print length = ~D~]"
 print-level *print-length*)
 " print length = 5"))

(let [foo "Items:~#[none~; ~S~; ~S and ~S~
 ~:;~@{~#[~; and~] ~
 ~S~~,~}~]."]
 (simple-tests cltl-bracket1-tests
 (format nil foo) "Items: none."
 (format nil foo 'foo) "Items: foo."
 (format nil foo 'foo 'bar) "Items: foo and bar."
 (format nil foo 'foo 'bar 'baz) "Items: foo, bar, and baz."
 (format nil foo 'foo 'bar 'baz 'quux)
 "Items: foo, bar, baz, and quux.))

(simple-tests cltl-curly-bracket-tests
 (format nil
 "The winners are:~{ ~S~}."
 '(fred harry jill))
 "The winners are: fred harry jill."

 (format nil "Pairs:~{ <~S,~S>~}." '(a 1 b 2 c 3))
 "Pairs: <a,1> <b,2> <c,3>."

 (format nil "Pairs:~:{ <~S,~S>~}." '((a 1) (b 2) (c 3)))
 "Pairs: <a,1> <b,2> <c,3>."

 (format nil "Pairs:~@{ <~S,~S>~}." 'a 1 'b 2 'c 3)
 "Pairs: <a,1> <b,2> <c,3>."

 (format nil "Pairs:~:@{ <~S,~S>~}." '(a 1) '(b 2) '(c 3))
 "Pairs: <a,1> <b,2> <c,3>.")

(simple-tests cltl-angle-bracket-tests
 (format nil "~10<foo~;bar~>") "foo bar"
 (format nil "~10:<foo~;bar~>") " foo bar"
 (format nil "~10:@<foo~;bar~>") " foo bar "
 (format nil "~10<foobar~>") " foobar"
 (format nil "~10:<foobar~>") " foobar"
```

```

(format nil "~10@<foobar~>") "foobar "
(format nil "~10:@<foobar~>") " foobar ")

(let [donestr "Done.~~ ~D warning~:P.~~ ~D error~:P."
 ;; The CLtL example is a little wrong here
 tellstr "~@{~@(~[~R~~ ~]~A~)~}."]

(simple-tests cltl-up-tests
 (format nil donestr) "Done."
 (format nil donestr 3) "Done. 3 warnings."
 (format nil donestr 1 5) "Done. 1 warning. 5 errors."
 (format nil tellstr 23) "Twenty-three."
 (format nil tellstr nil "losers") "Losers."
 (format nil tellstr 23 "losers") "Twenty-three losers."
 (format nil "~15<~S~;~~~S~;~~~S~>" 'foo)
 " foo"
 (format nil "~15<~S~;~~~S~;~~~S~>" 'foo 'bar)
 "foo bar"
 (format nil "~15<~S~;~~~S~;~~~S~>" 'foo 'bar 'baz)
 "foo bar baz"))

(simple-tests cltl-up-x3j13-tests
 (format nil
 "~:{/~S~~ ...~}"
 '((hot dog) (hamburger) (ice cream) (french fries)))
 "/hot .../hamburger/ice .../french ..."
 (format nil
 "~:{/~S~:~ ...~}"
 '((hot dog) (hamburger) (ice cream) (french fries)))
 "/hot .../hamburger .../ice .../french"

 (format nil
 "~:{/~S~#:~ ...~}" ;; This is wrong in CLtL
 '((hot dog) (hamburger) (ice cream) (french fries)))
 "/hot .../hamburger")

```

---

## 12.54 test1/test-helper.clj

— test1/test-helper.clj —

```

\getchunk{Clojure Copyright}
;; test_helper.clj -- part of the pretty printer for Clojure
;; Author: Tom Faulhaber

```

```
;; April 3, 2009

;; This is just a macro to make my tests a little cleaner

(ns clojure.test-clojure pprint test-helper
 (:use [clojure.test :only (deftest is])
 [clojure.test-helper :only [platform-newlines]]))

(defn- back-match [x y] (re-matches y x))

(defmacro simple-tests [name & test-pairs]
 `(deftest ~name
 `@(for [[x y] (partition 2 test-pairs)]
 (cond
 (instance? java.util.regex.Pattern y)
 `(is (#'clojure.test-clojure pprint test-helper/back-match
 ~x ~y))
 (instance? java.lang.String y)
 `'(is (= ~x (platform-newlines ~y)))
 :else `'(is (= ~x ~y))))))
```

---

## 12.55 test/test'pretty.clj

— test/test'pretty.clj —

```
\getchunk{Clojure Copyright}
;;; test.pretty.clj -- part of the pretty printer for Clojure

;; Author: Tom Faulhaber
;; April 3, 2009

(in-ns 'clojure.test-clojure pprint)

;;;;;;
;;;
;;; Unit tests for the pretty printer
;;;
;;;;;;

(simple-tests xp-fill-test
 (binding [*print-pprint-dispatch* simple-dispatch
 print-right-margin 38
```

```

print-miser-width nil]
(cl-format nil "(let ~:@{~:<~w ~_~w~:>~~ ~:_~}~:>_ ...)~%"
'((x 4) (*print-length* nil) (z 2) (list nil)))
"(let ((x 4) (*print-length* nil)\n (z 2) (list nil))\n ...)\n"

(binding [*print-pprint-dispatch* simple-dispatch
 print-right-margin 22]
 (cl-format nil "(let ~:@{~:<~w ~_~w~:>~~ ~:_~}~:>_ ...)~%"
'((x 4) (*print-length* nil) (z 2) (list nil)))
(str "(let ((x 4)\n (*print-length*\n nil)\n (z 2)\n (list nil))\n ...)\n"))

(simple-tests xp-miser-test
 (binding [*print-pprint-dispatch* simple-dispatch
 print-right-margin 10, *print-miser-width* 9]
 (cl-format nil "~:<LIST ~@_~W ~@_~W ~@_~W~:>" '(first second third))
 "(LIST\n first\n second\n third)"

 (binding [*print-pprint-dispatch* simple-dispatch
 print-right-margin 10, *print-miser-width* 8]
 (cl-format nil "~:<LIST ~@_~W ~@_~W ~@_~W~:>" '(first second third))
 "(LIST first second third)")

 (simple-tests mandatory-fill-test
 (cl-format nil
 "<pre>~%~<Usage: ~:I~@{*~a*~~~:@_~}~:>~%</pre>~%"
 ["hello" "gooodbye"])
 "<pre>
Usage: *hello*
gooodbye
</pre>
")
)

(simple-tests prefix-suffix-test
 (binding [*print-pprint-dispatch* simple-dispatch
 print-right-margin 10, *print-miser-width* 10]
 (cl-format nil "~<{~;LIST ~@_~W ~@_~W ~@_~W~;}~:>" '(first second third))
 "{LIST\n first\n second\n third}")

(simple-tests pprint-test
 (binding [*print-pprint-dispatch* simple-dispatch]
 (write
 '(defn foo [x y]
 (let [result (* x y)]
 (if (> result 400)
 (cl-format true "That number is too big")

```

```

 (cl-format true "The result of ~d x ~d is ~d" x y result)))))
:stream nil))
"(defn
foo
[x y]
(let
[result (* x y)]
(if
(> result 400)
(cl-format true \"That number is too big\")
(cl-format true \"The result of ~d x ~d is ~d\" x y result))))"

(with-pprint-dispatch code-dispatch
(write
'(defn foo [x y]
(let [result (* x y)]
(if (> result 400)
(cl-format true "That number is too big")
(cl-format true "The result of ~d x ~d is ~d" x y result))))
:stream nil))
"(defn foo [x y]
(let [result (* x y)]
(if (> result 400)
(cl-format true \"That number is too big\")
(cl-format true \"The result of ~d x ~d is ~d\" x y result))))"

(binding [*print-pprint-dispatch* simple-dispatch
 print-right-margin 15]
 (write '(fn (cons (car x) (cdr y))) :stream nil))
"(fn\n (cons\n (car x)\n (cdr y)))"

(with-pprint-dispatch code-dispatch
(binding [*print-right-margin* 52]
 (write
'(add-to-buffer this (make-buffer-blob (str (char c)) nil))
:stream nil)))
"(add-to-buffer\n this\n (make-buffer-blob (str (char c)) nil))"
)

(simple-tests pprint-reader-macro-test
 (with-pprint-dispatch code-dispatch
 (write (read-string "(map #(first %) [[1 2 3] [4 5 6] [7]])"))
 :stream nil))
"(map #(first %) [[1 2 3] [4 5 6] [7]])"

 (with-pprint-dispatch code-dispatch
 (write (read-string "@@(ref (ref 1)))")
 :stream nil))

```

```

"@@(ref (ref 1))"

(with-pprint-dispatch code-dispatch
 (write (read-string "'foo"))
 :stream nil))
"'foo"
)

(simple-tests code-block-tests
 (with-out-str
 (with-pprint-dispatch code-dispatch
 (pprint
 '(defn cl-format
 "An implementation of a Common Lisp compatible format function"
 [stream format-in & args]
 (let [compiled-format
 (if (string? format-in)
 (compile-format format-in)
 format-in)
 navigator (init-navigator args)]
 (execute-format stream compiled-format navigator)))))))
"(defn cl-format
 \"An implementation of a Common Lisp compatible format function\
[stream format-in & args]
(let [compiled-format (if (string? format-in)
 (compile-format format-in)
 format-in)
 navigator (init-navigator args)]
 (execute-format stream compiled-format navigator)))
"
""

(with-out-str
 (with-pprint-dispatch code-dispatch
 (pprint
 '(defn pprint-defn [writer alis]
 (if (next alis)
 (let [[defn-sym defn-name & stuff] alis
 [doc-str stuff] (if (string? (first stuff))
 [(first stuff) (next stuff)]
 [nil stuff])
 [attr-map stuff] (if (map? (first stuff))
 [(first stuff) (next stuff)]
 [nil stuff])]
 (pprint-logical-block writer :prefix "(" :suffix ")"
 (cl-format true "~w ~1I~@_~w" defn-sym defn-name)
 (if doc-str
 (cl-format true " ~_~w" doc-str))
 (if attr-map
 (cl-format true " ~_~w" attr-map)))
 ;; Note: the multi-defn case will work OK for

```

```

;; malformed defns too
(cond
 (vector? (first stuff))
 (single-defn stuff (or doc-str attr-map))
 :else
 (multi-defn stuff (or doc-str attr-map))))
(pprint-simple-code-list writer alis)))))

"(defn pprint-defn [writer alis]
(if (next alis)
 (let [[defn-sym defn-name & stuff] alis
 [doc-str stuff] (if (string? (first stuff))
 [(first stuff) (next stuff)]
 [nil stuff])
 [attr-map stuff] (if (map? (first stuff))
 [(first stuff) (next stuff)]
 [nil stuff])]
 (pprint-logical-block
 writer
 :prefix
 \"(\
 :suffix
 \")\
 (cl-format true \"~w ~I~@_~w\" defn-sym defn-name)
 (if doc-str (cl-format true \" ~_~w\" doc-str))
 (if attr-map (cl-format true \" ~_~w\" attr-map)))
 (cond
 (vector? (first stuff)) (single-defn
 stuff
 (or doc-str attr-map))
 :else (multi-defn stuff (or doc-str attr-map))))
 (pprint-simple-code-list writer alis)))
 ""))
)

(defn tst-pprint
 "A helper function to pprint to a string with a restricted right
margin"
 [right-margin obj]
 (binding [*print-right-margin* right-margin
 print-pretty true]
 (write obj :stream nil)))

;;; A bunch of predefined data to print
(def future-filled (future-call (fn [] 100)))
@future-filled
(def future-unfilled
 (future-call (fn [] (.acquire (java.util.concurrent.Semaphore. 0)))))
(def promise-filled (promise))
(deliver promise-filled '(first second third))
(def promise-unfilled (promise))

```

```
(def basic-agent (agent '(first second third)))
(defn failed-agent
 "must be a fn because you cannot await agents during load"
 [])
(let [a (agent "foo")]
 (send a +)
 (try (await-for 100 a) (catch RuntimeException re))
 a))
(def basic-atom (atom '(first second third)))
(def basic-ref (ref '(first second third)))
(def delay-forced (delay '(first second third)))
(force delay-forced)
(def delay-unforced (delay '(first second third)))
(defrecord pprint-test-rec [a b c])

(simple-tests pprint-datastructures-tests
(tst-pprint 20 future-filled)
 "#<Future@[0-9a-f]+: \r?\n 100>"
(tst-pprint 20 future-unfilled)
 "#<Future@[0-9a-f]+: \r?\n :pending>"
(tst-pprint 20 promise-filled)
 "#<Promise@[0-9a-f]+: \r?\n \\\(first\r?\n second\r?\n third\\)>"
;; This hangs currently, cause we can't figure out whether a promise
;; is filled (tst-pprint 20 promise-unfilled)
;; "#<Promise@[0-9a-f]+: \r?\n :pending>"
(tst-pprint 20 basic-agent)
 "#<Agent@[0-9a-f]+: \r?\n \\\(first\r?\n second\r?\n third\\)>"
(tst-pprint 20 (failed-agent))
 "#<Agent@[0-9a-f]+ FAILED: \r?\n \"foo\">"
(tst-pprint 20 basic-atom)
 "#<Atom@[0-9a-f]+: \r?\n \\\(first\r?\n second\r?\n third\\r?\\)>"
(tst-pprint 20 basic-ref)
 "#<Ref@[0-9a-f]+: \r?\n \\\(first\r?\n second\r?\n third\\)>"
(tst-pprint 20 delay-forced)
 "#<Delay@[0-9a-f]+: \r?\n \\\(first\r?\n second\r?\n third\\r?\\)>"
;; Currently no way not to force the delay
;; (tst-pprint 20 delay-unforced)
 "#<Delay@[0-9a-f]+: \n :pending>"
;tpd i broke this
; (tst-pprint 20 (pprint-test-rec 'first 'second 'third))
; "{:a first,\n :b second,\n :c third}"

;; basic java arrays: fails owing to assembla ticket #346
; ;(tst-pprint 10 (int-array (range 7)))
; ; "[0,\n 1,\n 2,\n 3,\n 4,\n 5,\n 6]"
; ;(tst-pprint 15
; ; (reduce conj clojure.lang.PersistentQueue/EMPTY (range 10)))
; ; "<-(0\n 1\n 2\n 3\n 4\n 5\n 6\n 7\n 8\n 9)-<"
)
```

```
;;; Some simple tests of dispatch

(defmulti
 test-dispatch
 "A test dispatch method"
 {:added "1.2" :arglists '[[object]]}
 #,(and (seq %) (not (string? %)))

(defmethod test-dispatch true [avec]
 (pprint-logical-block :prefix "[" :suffix "]"
 (loop [aseq (seq avec)]
 (when aseq
 (write-out (first aseq))
 (when (next aseq)
 (.write ^java.io.Writer *out* " ")
 (pprint-newline :linear)
 (recur (next aseq))))))
)

(defmethod test-dispatch false [aval] (pr aval))

(simple-tests dispatch-tests
 (with-pprint-dispatch test-dispatch
 (with-out-str
 (pprint '("hello" "there"))))
 "[\"hello\" \"there\"]\n")
)
```

—

## 12.56 test1/examples.clj

— test1/examples.clj —

```
(ns clojure.test-clojure.protocols.examples)

(defprotocol ExampleProtocol
 "example protocol used by clojure tests"

 (foo [a] "method with one arg")
 (bar [a b] "method with two args")
 (^String baz [a] [a b] "method with multiple arities")
 (with-quux [a] "method name with a hyphen"))

(definterface ExampleInterface
```

```
(hinted [^int i])
(hinted [^String s]))
```

—————

## 12.57 test/more-examples.clj

— test/more-examples.clj —

```
(ns clojure.test-clojure.protocols.more-examples)

(defprotocol SimpleProtocol
 "example protocol used by clojure tests. Note that
 foo collides with examples/ExampleProtocol."
 (foo [a] ""))

```

—————

## 12.58 test/example.clj

— test/example.clj —

```
(ns clojure.test-clojure.repl.example)

;; sample namespace for repl tests, don't add anything here
(defn foo [])
(defn bar [])
```

—————

## Appendix A

# External Java References

Click on a class name to go to the appropriate web page.

- AbstractMap
- Callable
- Collection
- Comparable
- Comparator
- ConcurrentMap
- DefaultHandler
- Enumeration
- IllegalArgumentException
- InvocationHandler
- Iterable
- Iterator
- List
- Map
- Map.Entry
- Number
- PushbackReader
- RandomAccess
- Runnable
- Serializable
- Set
- URLClassLoader



## Appendix B

# Copyright and Licenses

### — Clojure License —

Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

#### 1. DEFINITIONS

"Contribution" means:

- a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
- b) in the case of each subsequent Contributor:
  - i) changes to the Program, and
  - ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which:  
(i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

## 2. GRANT OF RIGHTS

- a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
- b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.
- c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.
- d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

## 3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

- a) it complies with the terms and conditions of this Agreement; and
- b) its license agreement:
  - i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
  - ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
  - iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
  - iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

- a) it must be made available under this Agreement; and
- b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

#### 4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising

from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

#### 5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

#### 6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### 7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

---

— Clojure Copyright —

```
; Copyright (c) Rich Hickey. All rights reserved. The use and
; distribution terms for this software are covered by the Eclipse
; Public License 1.0
; (http://opensource.org/licenses/eclipse-1.0.php) which can be
; found in the file epl-v10.html at the root of this distribution.
; By using this software in any fashion, you are agreeing to be
; bound by the terms of this license. You must not remove this
; notice, or any other, from this software.
```

---

---

— France Telecom Copyright —

```
/***
* ASM: a very small and fast Java bytecode manipulation framework
* Copyright (c) 2000-2005 INRIA, France Telecom
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
* documentation and/or other materials provided with the
* distribution.
* 3. Neither the name of the copyright holders nor the names of its
* contributors may be used to endorse or promote products derived
* from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
```

\*/

—

— Houser Copyright and License —

```
; Copyright (c) Chris Houser, Dec 2008. All rights reserved. The
; use and distribution terms for this software are covered by the
; Common Public License 1.0 (http://opensource.org/licenses/cpl.php)
; which can be found in the file CPL.TXT at the root of this
; distribution. By using this software in any fashion, you are
; agreeing to be bound by the terms of this license. You must not
; remove this notice, or any other, from this software.
```

—

— Lórange Copyright and License —

Copyright 2013 Jean Niklas L\’orange.  
This text is licensed under the Creative Commons  
Attribution-ShareAlike 3.0 Unported License, unless otherwise  
specified

—

— Kingsbury Copyright and License —

Copyright 2013 Kyle Kingsbury.  
Non-commercial re-use with attribution encouraged; all other rights reserved.

—

— Kingsbury Copyright and License —

Copyright 2013 Kai Wu  
Licensed under Common Public License 1.0

—



## Appendix C

# Building Clojure from this document

### C.1 The basic idea

Literate programs are TeXdocuments that contain executable source code. If you have a running system you can use the included clojure code to extract chunks at will. But what if you don't have clojure yet? There has to be a "bootstrap" mechanism and we supply that here.

First, you need to edit this file and clip out the "tangle" function written in C below. You compile this with GCC. This gives you a small program that will extract named chunks from this document.

Second, you extract the Makefile chunk. This Makefile will use the tangle function to create the clojure source tree, including the build.xml file which is used by the ant build program. Note that this should be rewritten to be part of the build.xml but that is not part of Clojure.

Third, you run the Makefile. It will extract all of the code into the proper places in the source tree and run the ant build process to create a running Clojure.

### C.2 The tangle function in C

— tangle.c —

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
```

```

#include <sys/mman.h>
#include <fcntl.h>

#define DEBUG 0

/* forward reference for the C compiler */
int getchunk(char *chunkname);

/* a memory mapped buffer copy of the file */
char *buffer;
int bufsize;

/* return the length of the next line */
int nextline(int i) {
 int j;
 if (i >= bufsize) return(-1);
 for (j=0; ((i+j < bufsize) && (buffer[i+j] != '\n')); j++);
 return(j);
}

/* output the line we need */
int printline(int i, int length) {
 int j;
 for (j=0; j<length; j++) { putchar(buffer[i+j]); }
 printf("\n");
}

/* handle begin{chunk}{chunkname} */
/* is this chunk name we are looking for? */
int foundchunk(int i, char *chunkname) {
 if ((strncmp(&buffer[i+14],chunkname,strlen(chunkname)) == 0) &&
 (buffer[i+13] == '{') &&
 (buffer[i+14+strlen(chunkname)] == '}')) return(1);
 return(0);
}

/* handle end{chunk} */
/* is it really an end? */
int foundEnd(int i) {
 if ((buffer[i] == '\\') &&
 (strncmp(&buffer[i+1],"end{chunk}",10) == 0)) {
 return(1);
 }
 return(0);
}

/* handle getchunk{chunkname} */
/* is this line a getchunk? */
int foundGetchunk(int i, int linelen) {
 int len;

```

```

if (strncmp(&buffer[i], "\\getchunk{", 10) == 0) {
 for(len=0; ((len < linelen) && (buffer[i+len] != '}')); len++);
 return(len-10);
}
return(0);
}

/* Somebody did a getchunk and we need a copy of the name */
/* malloc string storage for a copy of the getchunk name */
char *getChunkname(int k, int getlen) {
 char *result = (char *)malloc(getlen+1);
 strncpy(result,&buffer[k+10],getlen);
 result[getlen]='\0';
 return(result);
}

/* print lines in this chunk, possibly recursing into getchunk */
int printchunk(int i, int chunklinelen, char *chunkname) {
 int j;
 int k;
 int linelen;
 char *getname;
 int getlen = 0;
 for (k=i+chunklinelen+1; ((linelen=nextline(k)) != -1);) {
 if (DEBUG==2) {
 printf("">>>>"); printline(k,linelen); printf("<<<\n");
 }
 if ((getlen=foundGetchunk(k,linelen)) > 0) {
 getname = getChunkname(k,getlen);
 getchunk(getname);
 free(getname);
 k=k+getlen+12l;
 } else {
 if ((linelen >= 11) && (foundEnd(k) == 1)) {
 if (DEBUG) { printf("=====\\end{%s}\n",chunkname); }
 return(k+12);
 } else {
 if (DEBUG==2) {
 printf("===== printchunk else %d %d\n",k,linelen);
 }
 printline(k,linelen);
 k=k+linelen+1;
 }
 }
 }
 if (DEBUG==2) {
 printf("=====\\out{%s} %d\n",chunkname,k);
 }
 return(k);
}

```

```

/* find the named chunk and call printchunk on it */
int getchunk(char *chunkname) {
 int i;
 int j;
 int linelen;
 int chunklen = strlen(chunkname);
 for (i=0; ((linelen=nextline(i)) != -1);) {
 if (DEBUG==2) {
 printf("----"); printline(i,linelen); printf("----\n");
 }
 if ((linelen >= chunklen+15) && (foundchunk(i,chunkname) == 1)) {
 if (DEBUG) {
 fprintf(stderr,"=====\\getchunk(%s)\n",chunkname);
 }
 i=printchunk(i,linelen,chunkname);
 } else {
 i=i+linelen+1;
 }
 }
 if (DEBUG) {
 fprintf(stderr,"=====getchunk returned=%d\n",i);
 }
 return(i);
}

/* memory map the input file into the global buffer and get the chunk */
int main(int argc, char *argv[]) {
 int fd;
 struct stat filestat;
 if ((argc < 2) || (argc > 3)) {
 perror("Usage: tangle filename chunkname");
 exit(-1);
 }
 fd = open(argv[1],O_RDONLY);
 if (fd == -1) {
 perror("Error opening file for reading");
 exit(-2);
 }
 if (fstat(fd,&filestat) < 0) {
 perror("Error getting input file size");
 exit(-3);
 }
 bufsize = (int)filestat.st_size;
 buffer = mmap(0,filestat.st_size,PROT_READ,MAP_SHARED,fd,0);
 if (buffer == MAP_FAILED) {
 close(fd);
 perror("Error reading the file");
 exit(-4);
 }
 getchunk(argv[2]);
}

```

```
 close(fd);
 return(0);
}
```

---

### C.3 Makefile

— Makefile —

```
BOOK=clojure.pamphlet
WHERE=tpd
CLOJURE=${WHERE}/src/clj/clojure
CORE=${WHERE} // src/clj/clojure/core
JAVA=${WHERE}/src/clj/clojure/java
PPRINT=${WHERE}/src/clj/clojure/pprint
REFLECT=${WHERE}/src/clj/clojure/reflect
TEST=${WHERE}/src/clj/clojure/test
MAIN=${WHERE}/src/jvm/clojure
ASM=${WHERE}/src/jvm/clojure/asm
COMMONS=${WHERE}/src/jvm/clojure/asm/commons
LANG=${WHERE}/src/jvm/clojure/lang
TESTA=${WHERE}/test/clojure
TESTC=${WHERE}/test/clojure/test_clojure
TESTD=${WHERE}/test/clojure/test_clojure/annotations
TESTE=${WHERE}/test/clojure/test_clojure/genclass
TESTF=${WHERE}/test/clojure/test_clojure/java
TESTG=${WHERE}/test/clojure/test_clojure/pprint
TESTH=${WHERE}/test/clojure/test_clojure/protocols
TESTI=${WHERE}/test/clojure/test_clojure/repl

all:
rm -rf ${WHERE}
mkdir -p ${CLOJURE}
mkdir -p ${CORE}
mkdir -p ${JAVA}
mkdir -p ${PPRINT}
mkdir -p ${REFLECT}
mkdir -p ${TEST}
mkdir -p ${ASM}
mkdir -p ${COMMONS}
mkdir -p ${LANG}
mkdir -p ${TESTA}
mkdir -p ${TESTC}
mkdir -p ${TESTD}
mkdir -p ${TESTE}
```

```

mkdir -p ${TESTF}
mkdir -p ${TESTG}
mkdir -p ${TESTH}
mkdir -p ${TESTI}
tangle ${BOOK} build.xml >${WHERE}/build.xml
tangle ${BOOK} pom-template.xml >${WHERE}/pom-template.xml
tangle ${BOOK} protocols.clj >${CORE}/protocols.clj
tangle ${BOOK} core.clj >${CLOJURE}/core.clj
tangle ${BOOK} core_deftype.clj >${CLOJURE}/core_deftype.clj
tangle ${BOOK} core_print.clj >${CLOJURE}/core_print.clj
tangle ${BOOK} core_proxy.clj >${CLOJURE}/core_proxy.clj
tangle ${BOOK} data.clj >${CLOJURE}/data.clj
tangle ${BOOK} genclass.clj >${CLOJURE}/genclass.clj
tangle ${BOOK} gvec.clj >${CLOJURE}/gvec.clj
tangle ${BOOK} inspector.clj >${CLOJURE}/inspector.clj
tangle ${BOOK} browse.clj >${JAVA}/browse.clj
tangle ${BOOK} browse_ui.clj >${JAVA}/browse_ui.clj
tangle ${BOOK} io.clj >${JAVA}/io.clj
tangle ${BOOK} javadoc.clj >${JAVA}/javadoc.clj
tangle ${BOOK} shell.clj >${JAVA}/shell.clj
tangle ${BOOK} main.clj >${CLOJURE}/main.clj
tangle ${BOOK} parallel.clj >${CLOJURE}/parallel.clj
tangle ${BOOK} cl_format.clj >${PPRINT}/cl_format.clj
tangle ${BOOK} column_writer.clj >${PPRINT}/column_writer.clj
tangle ${BOOK} dispatch.clj >${PPRINT}/dispatch.clj
tangle ${BOOK} pprint_base.clj >${PPRINT}/pprint_base.clj
tangle ${BOOK} pretty_writer.clj >${PPRINT}/pretty_writer.clj
tangle ${BOOK} print_table.clj >${PPRINT}/print_table.clj
tangle ${BOOK} utilities.clj >${PPRINT}/utilities.clj
tangle ${BOOK} pprint.clj >${CLOJURE}/pprint.clj
tangle ${BOOK} java.clj >${REFLECT}/java.clj
tangle ${BOOK} reflect.clj >${CLOJURE}/reflect.clj
tangle ${BOOK} repl.clj >${CLOJURE}/repl.clj
tangle ${BOOK} set.clj >${CLOJURE}/set.clj
tangle ${BOOK} stacktrace.clj >${CLOJURE}/stacktrace.clj
tangle ${BOOK} string.clj >${CLOJURE}/string.clj
tangle ${BOOK} template.clj >${CLOJURE}/template.clj
tangle ${BOOK} junit.clj >${TEST}/junit.clj
tangle ${BOOK} tap.clj >${TEST}/tap.clj
tangle ${BOOK} test.clj >${CLOJURE}/test.clj
tangle ${BOOK} version.properties >${CLOJURE}/version.properties
tangle ${BOOK} walk.clj >${CLOJURE}/walk.clj
tangle ${BOOK} xml.clj >${CLOJURE}/xml.clj
tangle ${BOOK} zip.clj >${CLOJURE}/zip.clj
tangle ${BOOK} AnnotationVisitor.java \
 >${ASM}/AnnotationVisitor.java
tangle ${BOOK} AnnotationWriter.java \
 >${ASM}/AnnotationWriter.java
tangle ${BOOK} Attribute.java >${ASM}/Attribute.java
tangle ${BOOK} ByteVector.java >${ASM}/ByteVector.java

```

```

tangle ${BOOK} ClassAdapter.java >${ASM}/ClassAdapter.java
tangle ${BOOK} ClassReader.java >${ASM}/ClassReader.java
tangle ${BOOK} ClassVisitor.java >${ASM}/ClassVisitor.java
tangle ${BOOK} ClassWriter.java >${ASM}/ClassWriter.java
tangle ${BOOK} ClassWriter.java >${ASM}/ClassWriter.java
tangle ${BOOK} AdviceAdapter.java \
 >${COMMONS}/AdviceAdapter.java
tangle ${BOOK} AnalyzerAdapter.java \
 >${COMMONS}/AnalyzerAdapter.java
tangle ${BOOK} CodeSizeEvaluator.java \
 >${COMMONS}/CodeSizeEvaluator.java
tangle ${BOOK} EmptyVisitor.java \
 >${COMMONS}/EmptyVisitor.java
tangle ${BOOK} GeneratorAdapter.java \
 >${COMMONS}/GeneratorAdapter.java
tangle ${BOOK} LocalVariablesSorter.java \
 >${COMMONS}/LocalVariablesSorter.java
tangle ${BOOK} Method.java >${COMMONS}/Method.java
tangle ${BOOK} serialVersionUIDAdder.java \
 >${COMMONS}/SerialVersionUIDAdder.java
tangle ${BOOK} StaticInitMerger.java \
 >${COMMONS}/StaticInitMerger.java
tangle ${BOOK} TableSwitchGenerator.java \
 >${COMMONS}/TableSwitchGenerator.java
tangle ${BOOK} Edge.java >${ASM}/Edge.java
tangle ${BOOK} FieldVisitor.java >${ASM}/FieldVisitor.java
tangle ${BOOK} FieldWriter.java >${ASM}/FieldWriter.java
tangle ${BOOK} Frame.java >${ASM}/Frame.java
tangle ${BOOK} Handler.java >${ASM}/Handler.java
tangle ${BOOK} Item.java >${ASM}/Item.java
tangle ${BOOK} Label.java >${ASM}/Label.java
tangle ${BOOK} MethodAdapter.java >${ASM}/MethodAdapter.java
tangle ${BOOK} MethodVisitor.java >${ASM}/MethodVisitor.java
tangle ${BOOK} MethodWriter.java >${ASM}/MethodWriter.java
tangle ${BOOK} Opcodes.java >${ASM}/Opcodes.java
tangle ${BOOK} Type.java >${ASM}/Type.java
tangle ${BOOK} AFn.java >${LANG}/AFn.java
tangle ${BOOK} AFunction.java >${LANG}/AFunction.java
tangle ${BOOK} Agent.java >${LANG}/Agent.java
tangle ${BOOK} AMapEntry.java >${LANG}/AMapEntry.java
tangle ${BOOK} APersistentMap.java >${LANG}/APersistentMap.java
tangle ${BOOK} APersistentSet.java >${LANG}/APersistentSet.java
tangle ${BOOK} APersistentVector.java \
 >${LANG}/APersistentVector.java
tangle ${BOOK} AReference.java >${LANG}/AReference.java
tangle ${BOOK} ARef.java >${LANG}/ARef.java
tangle ${BOOK} ArityException.java >${LANG}/ArityException.java
tangle ${BOOK} ArrayChunk.java >${LANG}/ArrayChunk.java
tangle ${BOOK} ArraySeq.java >${LANG}/ArraySeq.java
tangle ${BOOK} ASeq.java >${LANG}/ASeq.java

```

```

tangle ${BOOK} Associative.java >${LANG}/Associative.java
tangle ${BOOK} Atom.java >${LANG}/Atom.java
tangle ${BOOK} ATransientMap.java >${LANG}/ATransientMap.java
tangle ${BOOK} ATransientSet.java >${LANG}/ATransientSet.java
tangle ${BOOK} BigInt.java >${LANG}/BigInt.java
tangle ${BOOK} Binding.java >${LANG}/Binding.java
tangle ${BOOK} Box.java >${LANG}/Box.java
tangle ${BOOK} ChunkBuffer.java >${LANG}/ChunkBuffer.java
tangle ${BOOK} ChunkedCons.java >${LANG}/ChunkedCons.java
tangle ${BOOK} Compile.java >${LANG}/Compile.java
tangle ${BOOK} Compiler.java >${LANG}/Compiler.java
tangle ${BOOK} Cons.java >${LANG}/Cons.java
tangle ${BOOK} Counted.java >${LANG}/Counted.java
tangle ${BOOK} Delay.java >${LANG}/Delay.java
tangle ${BOOK} DynamicClassLoader.java \
 >${LANG}/DynamicClassLoader.java
tangle ${BOOK} EnumerationSeq.java >${LANG}/EnumerationSeq.java
tangle ${BOOK} Fn.java >${LANG}/Fn.java
tangle ${BOOK} IChunkedSeq.java >${LANG}/IChunkedSeq.java
tangle ${BOOK} IChunk.java >${LANG}/IChunk.java
tangle ${BOOK} IDeref.java >${LANG}/IDeref.java
tangle ${BOOK} IEditableCollection.java \
 >${LANG}/IEditableCollection.java
tangle ${BOOK} IFn.java >${LANG}/IFn.java
tangle ${BOOK} IKeywordLookup.java >${LANG}/IKeywordLookup.java
tangle ${BOOK} ILookup.java >${LANG}/ILookup.java
tangle ${BOOK} ILookup.java >${LANG}/ILookup.java
tangle ${BOOK} ILookupSite.java >${LANG}/ILookupSite.java
tangle ${BOOK} ILookupThunk.java >${LANG}/ILookupThunk.java
tangle ${BOOK} IMapEntry.java >${LANG}/IMapEntry.java
tangle ${BOOK} IMeta.java >${LANG}/IMeta.java
tangle ${BOOK} Indexed.java >${LANG}/Indexed.java
tangle ${BOOK} IndexedSeq.java >${LANG}/IndexedSeq.java
tangle ${BOOK} IObj.java >${LANG}/IObj.java
tangle ${BOOK} IPersistentCollection.java \
 >${LANG}/IPersistentCollection.java
tangle ${BOOK} IPersistentList.java \ >${LANG}/IPersistentList.java
tangle ${BOOK} IPersistentMap.java >${LANG}/IPersistentMap.java
tangle ${BOOK} IPersistentSet.java >${LANG}/IPersistentSet.java
tangle ${BOOK} IPersistentStack.java \
 >${LANG}/IPersistentStack.java
tangle ${BOOK} IPersistentVector.java \
 >${LANG}/IPersistentVector.java
tangle ${BOOK} IPromiseImpl.java >${LANG}/IPromiseImpl.java
tangle ${BOOK} IProxy.java >${LANG}/IProxy.java
tangle ${BOOK} IReduce.java >${LANG}/IReduce.java
tangle ${BOOK} IReference.java >${LANG}/IReference.java
tangle ${BOOK} IRef.java >${LANG}/IRef.java
tangle ${BOOK} ISeq.java >${LANG}/ISeq.java

```

```

tangle ${BOOK} IteratorSeq.java >${LANG}/IteratorSeq.java
tangle ${BOOK} ITransientAssociative.java \
 >${LANG}/ITransientAssociative.java
tangle ${BOOK} ITransientCollection.java \
 >${LANG}/ITransientCollection.java
tangle ${BOOK} ITransientMap.java >${LANG}/ITransientMap.java
tangle ${BOOK} ITransientSet.java >${LANG}/ITransientSet.java
tangle ${BOOK} ITransientVector.java \
 >${LANG}/ITransientVector.java
tangle ${BOOK} Keyword.java >${LANG}/Keyword.java
tangle ${BOOK} KeywordLookupSite.java \
 >${LANG}/KeywordLookupSite.java
tangle ${BOOK} LazilyPersistentVector.java \
 >${LANG}/LazilyPersistentVector.java
tangle ${BOOK} LazySeq.java >${LANG}/LazySeq.java
tangle ${BOOK} LineNumberingPushbackReader.java \
 >${LANG}/LineNumberingPushbackReader.java
tangle ${BOOK} LineNumberingPushbackReader.java \
 >${LANG}/LineNumberingPushbackReader.java
tangle ${BOOK} LispReader.java >${LANG}/LispReader.java
tangle ${BOOK} LockingTransaction.java \
 >${LANG}/LockingTransaction.java
tangle ${BOOK} MapEntry.java >${LANG}/MapEntry.java
tangle ${BOOK} MapEquivalence.java >${LANG}/MapEquivalence.java
tangle ${BOOK} MethodImplCache.java \
 >${LANG}/MethodImplCache.java
tangle ${BOOK} MultiFn.java >${LANG}/MultiFn.java
tangle ${BOOK} Named.java >${LANG}/Named.java
tangle ${BOOK} Namespace.java >${LANG}/Namespace.java
tangle ${BOOK} Numbers.java >${LANG}/Numbers.java
tangle ${BOOK} Obj.java >${LANG}/Obj.java
tangle ${BOOK} PersistentArrayMap.java \
 >${LANG}/PersistentArrayMap.java
tangle ${BOOK} PersistentHashMap.java \
 >${LANG}/PersistentHashMap.java
tangle ${BOOK} PersistentHashSet.java \
 >${LANG}/PersistentHashSet.java
tangle ${BOOK} PersistentList.java >${LANG}/PersistentList.java
tangle ${BOOK} PersistentQueue.java \
 >${LANG}/PersistentQueue.java
tangle ${BOOK} PersistentStructMap.java \
 >${LANG}/PersistentStructMap.java
tangle ${BOOK} PersistentTreeMap.java \
 >${LANG}/PersistentTreeMap.java
tangle ${BOOK} PersistentTreeSet.java \
 >${LANG}/PersistentTreeSet.java
tangle ${BOOK} PersistentVector.java \
 >${LANG}/PersistentVector.java
tangle ${BOOK} ProxyHandler.java >${LANG}/ProxyHandler.java
tangle ${BOOK} Range.java >${LANG}/Range.java

```

```
tangle ${BOOK} Ratio.java >${LANG}/Ratio.java
tangle ${BOOK} Ref.java >${LANG}/Ref.java
tangle ${BOOK} Reflector.java >${LANG}/Reflector.java
tangle ${BOOK} Repl.java >${LANG}/Repl.java
tangle ${BOOK} RestFn.java >${LANG}/RestFn.java
tangle ${BOOK} Reversible.java >${LANG}/Reversible.java
tangle ${BOOK} RT.java >${LANG}/RT.java
tangle ${BOOK} Script.java >${LANG}/Script.java
tangle ${BOOK} Seqable.java >${LANG}/Seqable.java
tangle ${BOOK} SeqEnumeration.java >${LANG}/SeqEnumeration.java
tangle ${BOOK} SeqIterator.java >${LANG}/SeqIterator.java
tangle ${BOOK} Sequential.java >${LANG}/Sequential.java
tangle ${BOOK} Settable.java >${LANG}/Settable.java
tangle ${BOOK} Sorted.java >${LANG}/Sorted.java
tangle ${BOOK} StringSeq.java >${LANG}/StringSeq.java
tangle ${BOOK} Symbol.java >${LANG}/Symbol.java
tangle ${BOOK} TransactionalHashMap.java \
>${LANG}/TransactionalHashMap.java

tangle ${BOOK} Util.java >${LANG}/Util.java
tangle ${BOOK} Var.java >${LANG}/Var.java
tangle ${BOOK} XMLHandler.java >${LANG}/XMLHandler.java
tangle ${BOOK} main.java >${MAIN}/main.java
tangle ${BOOK} test/test_clojure.clj >${TESTA}/test_clojure.clj
tangle ${BOOK} test/test_helper.clj >${TESTA}/test_helper.clj
tangle ${BOOK} test/agents.clj >${TESTC}/agents.clj
tangle ${BOOK} test/annotations.clj >${TESTC}/annotations.clj
tangle ${BOOK} test/atoms.clj >${TESTC}/atoms.clj
tangle ${BOOK} test/clojure_set.clj >${TESTC}/clojure_set.clj
tangle ${BOOK} test/clojure_xml.clj >${TESTC}/clojure_xml.clj
tangle ${BOOK} test/clojure_zip.clj >${TESTC}/clojure_zip.clj
tangle ${BOOK} test/compilation.clj >${TESTC}/compilation.clj
tangle ${BOOK} test/control.clj >${TESTC}/control.clj
tangle ${BOOK} test/data.clj >${TESTC}/data.clj
tangle ${BOOK} test/data_structures.clj \
>${TESTC}/data_structures.clj

tangle ${BOOK} test/def.clj >${TESTC}/def.clj
tangle ${BOOK} test/errors.clj >${TESTC}/errors.clj
tangle ${BOOK} test/evaluation.clj >${TESTC}/evaluation.clj
tangle ${BOOK} test/for.clj >${TESTC}/for.clj
tangle ${BOOK} test/genclass.clj >${TESTC}/genclass.clj
tangle ${BOOK} test/java_interop.clj >${TESTC}/java_interop.clj
tangle ${BOOK} test/keywords.clj >${TESTC}/keywords.clj
tangle ${BOOK} test/logic.clj >${TESTC}/logic.clj
tangle ${BOOK} test/macros.clj >${TESTC}/macros.clj
tangle ${BOOK} test/main.clj >${TESTC}/main.clj
tangle ${BOOK} test/metadata.clj >${TESTC}/metadata.clj
tangle ${BOOK} test/multimethods.clj >${TESTC}/multimethods.clj
tangle ${BOOK} test/ns_libs.clj >${TESTC}/ns_libs.clj
tangle ${BOOK} test/numbers.clj >${TESTC}/numbers.clj
tangle ${BOOK} test/other_functions.clj \
```

```

>${TESTC}/other_functions.clj
tangle ${BOOK} test/parallel.clj >${TESTC}/parallel.clj
tangle ${BOOK} test/pprint.clj >${TESTC}/pprint.clj
tangle ${BOOK} test/predicates.clj >${TESTC}/predicates.clj
tangle ${BOOK} test/printer.clj >${TESTC}/printer.clj
tangle ${BOOK} test/protocols.clj >${TESTC}/protocols.clj
tangle ${BOOK} test/reader.clj >${TESTC}/reader.clj
tangle ${BOOK} test/reflect.clj >${TESTC}/reflect.clj
tangle ${BOOK} test/refs.clj >${TESTC}/refs.clj
tangle ${BOOK} test/repl.clj >${TESTC}/repl.clj
tangle ${BOOK} test/rt.clj >${TESTC}/rt.clj
tangle ${BOOK} test/sequences.clj >${TESTC}/sequences.clj
tangle ${BOOK} test/serialization.clj >${TESTC}/serialization.clj
tangle ${BOOK} test/special.clj >${TESTC}/special.clj
tangle ${BOOK} test/string.clj >${TESTC}/string.clj
tangle ${BOOK} test/test.clj >${TESTC}/test.clj
tangle ${BOOK} test/test_fixtures.clj >${TESTC}/test_fixtures.clj
tangle ${BOOK} test/transients.clj >${TESTC}/transients.clj
tangle ${BOOK} test/vars.clj >${TESTC}/vars.clj
tangle ${BOOK} test/vectors.clj >${TESTC}/vectors.clj
tangle ${BOOK} test/java_5.clj >${TESTD}/java_5.clj
tangle ${BOOK} test/java_6_and_later.clj \
 >${TESTD}/java_6_and_later.clj
tangle ${BOOK} test/examples.clj >${TESTE}/examples.clj
tangle ${BOOK} test/io.clj >${TESTF}/io.clj
tangle ${BOOK} test/javadoc.clj >${TESTF}/javadoc.clj
tangle ${BOOK} test/shell.clj >${TESTF}/shell.clj
tangle ${BOOK} test/test_cl_format.clj \
 >${TESTG}/test_cl_format.clj
tangle ${BOOK} test1/test_helper.clj >${TESTG}/test_helper.clj
tangle ${BOOK} test/test_pretty.clj >${TESTG}/test_pretty.clj
tangle ${BOOK} test1/examples.clj >${TESTH}/examples.clj
tangle ${BOOK} test/more_examples.clj >${TESTH}/more_examples.clj
tangle ${BOOK} test/example.clj >${TESTI}/example.clj
(cd ${WHERE}; ant)
latex clojure.pamphlet
makeindex clojure.idx
latex clojure.pamphlet
dvipdf clojure.dvi
xpdf clojure.pdf &
java -cp tpd/clojure.jar clojure.main

clean:
rm -f *.ilg
rm -f *.ind
rm -f *.out
rm -f *.aux
rm -f *.dvi
rm -f *.idx
rm -f *.log

```

```
rm -f *
rm -f *.ps
rm -f *.tex
rm -f *.toc
rm -f *.pdf
```

---

## C.4 The tangle function in Clojure

This is the tangle function written in Clojure. It would be useful to modify the reader to directly extract source code from literate documents. That would make literate documents a native file format for Clojure.

### C.4.1 Author and License

Timothy Daly ([daly@axiom-developer.org](mailto:daly@axiom-developer.org))  
License: Public Domain

### C.4.2 Abstract and Use Cases

Don Knuth has defined literate programming as a combination of documentation and source code in a single file. The TeX language is documented this way in books. Knuth defined two functions

- tangle -*i* extract the source code from a literate file
- weave -*i* extract the latex from a literate file

This seems unnecessarily complex. Latex is a full programming language and is capable of defining “environments” that can handle code directly in Latex. Here we define the correct environment macros. Thus, the “weave” function is not needed.

If this “tangle” function were added to Clojure then Clojure could read literate files in Latex format and extract the code. We create the necessary “tangle” function here.

This program will extract the source code from a literate file.

A literate lisp file contains a mixture of latex and lisp sources code. The file is intended to be in standard latex format. In order to delimit code chunks we define a latex “chunk” environment.

Latex format files defines a newenvironment so that code chunks can be delimited by `\begin{chunk}{name}` ... `\end{chunk}` blocks. This is supported by the following latex code.

So a trivial example of a literate latex file might look like

```
this is a file that is in a literate
form it has a chunk called
\begin{chunk}{first chunk}
THIS IS THE FIRST CHUNK
\end{chunk}
and this is a second chunk
\begin{chunk}{second chunk}
THIS IS THE SECOND CHUNK
\end{chunk}
and this is more in the first chunk
\begin{chunk}{first chunk}
\getchunk{second chunk}
THIS IS MORE IN THE FIRST CHUNK
\end{chunk}
\begin{chunk}{all}
\getchunk{first chunk}
\getchunk{second chunk}
\end{chunk}
and that's it
```

From a file called “testcase” that contains the above text we want to extract the chunk names “second chunk”. We do this with:

```
(tangle "testcase" "second chunk")
```

which yields:

```
THIS IS THE SECOND CHUNK
```

From the same file we might extract the chunk named “first chunk”. Notice that this has the second chunk embedded recursively inside. So we execute:

```
(tangle "testcase" "first chunk")
```

which yields:

```
THIS IS THE FIRST CHUNK
THIS IS THE SECOND CHUNK
THIS IS MORE IN THE FIRST CHUNK
```

There is a third chunk called “all” which will extract both chunks:

```
(tangle "testcase" "all")
```

which yields

```
THIS IS THE FIRST CHUNK
THIS IS THE SECOND CHUNK
THIS IS MORE IN THE FIRST CHUNK
THIS IS THE SECOND CHUNK
```

The tangle function takes a third argument which is the name of an output file. Thus, you can write the same results to a file with:

```
(tangle "testcase" "all" "outputfile")
```

It is also worth noting that all chunks with the same name will be merged into one chunk so it is possible to split chunks in multiple parts and have them extracted as one. That is,

```
\begin{chunk}{a partial chunk}
part 1 of the partial chunk
\end{chunk}
not part of the chunk
\begin{chunk}{a partial chunk}
part 2 of the partial chunk
\end{chunk}
```

These will be combined on output as a single chunk. Thus

```
(tangle "testmerge" "a partial chunk")
```

will yield

```
part 1 of the partial chunk
part 2 of the partial chunk
```

### C.4.3 The Latex Support Code

The verbatim package quotes everything within its grasp and is used to hide and quote the source code during latex formatting. The verbatim environment is built in but the package form lets us use it in our chunk environment and it lets us change the font.

```
\usepackage{verbatim}
```

Make the verbatim font smaller Note that we have to temporarily change the '`@`' to be just a character because the `@font` name uses it as a character

```
\chardef\atcode=\catcode`@
\catcode`\@=11
\renewcommand{\verb@font}{\ttfamily\small}
\catcode`\@=\atcode
```

This declares a new environment named “chunk” which has one argument that is the name of the chunk. All code needs to live between the `\begin{chunk}{name}` and the `\end{chunk}`. The “name” is used to define the chunk. Reuse of the same chunk name later concatenates the chunks.

For those of you who can’t read latex this says: Make a new environment named chunk with one argument. The first block is the code for the `\begin{chunk}{name}`. The second block is the code for the `\end{chunk}`. The % is the latex comment character.

We have two alternate markers, a lightweight one using dashes and a heavyweight one using the `\begin` and `\end` syntax. You can choose either one by changing the comment char in column 1

```
\newenvironment{chunk}[1]{%
 we need the chunkname as an argument
{\\ }\\newline\\noindent% make sure we are in column 1
%{\small \$\\backslash{}$begin\\{chunk\\}\\\\{{\\bf #1}\\}}% alternate begin mark
\\hbox{\\hskip 2.0cm}{\\bf --- #1 ---}}% mark the beginning
\\verbatim% say exactly what we see
{\\endverbatim% process \\end{chunk}
\\par{}% we add a newline
\\noindent{}% start in column 1
\\hbox{\\hskip 2.0cm}{\\bf -----}}% mark the end
%$\\backslash{}$end\\{chunk\\}% alternate end mark (commented)
\\par% and a newline
\\normalsize\\noindent{}% and return to the document
```

This declares the place where we want to expand a chunk. Technically we don’t need this because a getchunk must always be properly nested within a chunk and will be verbatim.

```
\providecommand{\getchunk}[1]{%
\\noindent%
{\small \$\\backslash{}$begin\\{chunk\\}\\\\{{\\bf #1}\\}}% mark the reference
```

#### C.4.4 Imports

##### — Clojure tangle —

```
(import [java.io BufferedReader FileReader BufferedWriter FileWriter])
(import [java.lang String])
```

### C.4.5 The Tangle Command

The tangle command does all of the work of extracting code.

In latex form the code blocks are delimited by

```
\begin{chunk}{name}
...
(code for name)...
\end{chunk}
```

and referenced by `\getchunk{name}` which gets replaced by the code

There are several ways to invoke the tangle function.

The first argument is always the file from which to extract code

The second argument is the name of the chunk to extract

```
(tangle "clweb.pamphlet" "name")
```

The standard chunk name is “\*” but any name can be used.

The third argument is the name of an output file:

```
(tangle "clweb.pamphlet" "clweb.chunk" "clweb.spadfile")
```

### C.4.6 The say function

This function will either write to the output file, or if null, to the console

— Clojure tangle —

```
(defn say [where what]
 (if where
 (do (.write where what) (.write where "\n"))
 (println what)))
```

---

### C.4.7 The read-file function

Here we return a lazy sequence that will fetch lines as we need them from the file.

— Clojure tangle —

```
(defn read-file [streamname]
 ^{:doc "Implement read-sequence in GCL"}
 (let [stream (BufferedReader. (FileReader. streamname))]
 (line-seq stream)))
```

---

### C.4.8 The ischunk function

There is a built-in assumption (in the ischunk functions) that the chunks occur on separate lines and that the indentation of the chunk reference has no meaning.

The ischunk function recognizes chunk names in latex convention

There are 3 cases to recognize:

```
\begin{chunk}{thechunkname} ==> 'define thechunkname
\end{chunk} ==> 'end nil
\getchunk{thechunkname} ==> 'refer thechunkname
```

The regex pattern #"<sup>^</sup>\begin{chunk}\{.\*\}\$" matches  
`\begin{chunk}{anything here}`

The regex pattern #"<sup>^</sup>\end{chunk}" matches  
`\end{chunk}`

The regex pattern #"<sup>^</sup>\getchunk\{.\*\}\$" matches  
`\getchunk{anything here}`

#### — Clojure tangle —

```
(defn ischunk [line]
 ^{:doc "Find chunks delimited by latex syntax"}
 (let [begin #"^\begin{chunk}\{.*\}$"
 end #"^\end{chunk}"
 get #"^\getchunk\{.*\}$"
 trimmed (.trim line)]
 (cond
 (re-find begin trimmed)
 (list 'define (apply str (butlast (drop 14 trimmed))))
 (re-find end trimmed)
 (list 'end nil)
 (re-find get trimmed)
 (list 'refer trimmed)
 :else
 (list nil trimmed))))
```

---

### C.4.9 The haschunks function

The hashchunks function gathers the chunks and puts them in the hash table. If we find the chunk syntax and it is a

- define ==> parse the chunkname and start gathering lines onto a stack
- end ==> push the completed list of lines into a stack of chunks already in the hash table
- otherwise ==> if we are gathering, push the line onto the stack

A hash table entry is a list of lists such as

```
((("6" "5") ("4" "3") ("2" "1")))
```

each of the sublists is a set of lines in reverse (stack) order each sublist is a single chunk of lines. There is a new sublist for each reuse of the same chunkname

Calls to ischunk can have 4 results (define, end, refer, nil) where

- define ==> we found a \begin{chunk}\{...\}
- end ==> we found a \end{chunk}
- refer ==> we found a \getchunk\{...\}
- nil ==> ordinary text or program text

The variable gather is initially false, implying that we are not gathering code. The variable gather is true if we are gathering a chunk.

#### — Clojure tangle —

```
(defn hashchunks [lines]
 ^{:doc "Gather all of the chunks and put them into a hash table"}
 (loop [line lines
 gather false
 hash (hash-map)
 chunkname ""]
 (if (not (empty? line))
 (let [[key value] (ischunk (first line))]
 (condp = key
 'define
 (recur (rest line) true hash value)
 'end
 (recur (rest line) false hash chunkname)
 'refer
 (if gather
 (recur (rest line) gather
 (assoc hash chunkname
 (conj (get hash chunkname) value)) chunkname)
 value)))
 (recur (rest line) gather hash chunkname))))
```

```

(recur (rest line) gather hash chunkname))
nil
(if gather
 (recur (rest line) gather
 (assoc hash chunkname
 (conj (get hash chunkname) value)) chunkname)
 (recur (rest line) gather hash chunkname)))
(hash)))

```

---

### C.4.10 The expand function

The expand function will recursively expand chunks in the hash table.

Latex chunk names are just the chunkname itself e.g. chunkname

A hash table key is the chunk name and the value is a reverse list of all of the text in that chunk. To process the chunk we reverse the main list and for each sublist we reverse the sublist and process the lines

If a chunk name reference is encountered in a line we call expand recursively to expand the inner chunkname.

#### — Clojure tangle —

```

(defn expand [chunkname where table]
^{:doc recursively expand latex getchunk tags}
(let [chunk (reverse (get table chunkname))]
(when chunk
(loop [lines chunk]
(when (not (empty? lines))
(let [line (first lines)]
(let [[key value] (ischunk line)]
(if (= key 'refer)
(do
(expand (apply str (butlast (drop 10 value))) where table)
(recur (rest lines)))
(do (say where line)
(recur (rest lines)))))))))))

```

---

### C.4.11 The tangle function

We expand all of the lines in the file that are surrounded by the requested chunk name. These chunk names are looked up in the hash table built by hashchunks,

given the input filename. then we recursively expand the “topchunk” to the output stream

— Clojure tangle —

```
(defn tangle
 ^{:doc "Extract the source code from a pamphlet file,
 optional file output"}
 ([filename topchunk] (tangle filename topchunk nil))
 ([filename topchunk file]
 (if (string? file)
 (with-open [where (BufferedWriter. (FileWriter. file))]
 (expand topchunk where (hashchunks (read-file filename)))))
 (expand topchunk nil (hashchunks (read-file filename))))))
```

---

# Bibliography

[Bag00] Bagwell, Phil

“Fast and space efficient trie searches”

Technical Report EPFL (2000)

<http://lampwww.epfl.ch/papers/triesearches.pdf.gz>

[Bag01] Bagwell, Phil

“Ideal Hash Trees”

Es Grands Champs (2001)

[Dal10] Daly, Timothy,

“Knuth’s literate programming “tangle” function in Clojure”

<groups.google.com/group/clojure>

thread /browse\_thread/thread/664a1d305f32ab90

[Dev12] Devlin, Sean

“Full Disclosure”

<vimeo.com/channels/fulldisclosure>

[DSST89] Driscoll, James; Sarnak, Neil; Sleator, Daniel; Tarjan, Robert

“Making Data Structures Persistent”

Journal of Computer and System Sciences 38, 86-124 (1989)

[Hal11] Halloway, Stuart

“Clojure Java Interop”

<www.infoq.com/presentations/Clojure-Java-Interop>

[Har05] Harper, Robert

“Programming in Standard ML”

<www.cs.cmu.edu/~rwh/smlbook/online.pdf>

[Hic10] Hickey, Rich

“Persistent Data Structures and Managed References”

[King13] Kingsbury, Kyle

“Clojure from the ground up”

- <aphyr.com/posts/301-clojure-from-the-ground-up-welcome>  
<aphyr.com/posts/302-clojure-from-the-ground-up-basic-types>
- [Knu84] Knuth, Donald  
 “Literate Programming (1984)”  
 Literate Programming CSLI, p99
- [L13] Lórange, Jean Niklas  
 “Understand Clojure’s Persistent Vectors”  
<hypirion.com/musings/understanding-persistent-vector-pt-1>  
<hypirion.com/musings/understanding-persistent-vector-pt-2>
- [Oka98] Okasaki, Chris  
 “Purely Functional Data Structures”  
 Cambridge University Press (1998) ISBN 0-521-66350-4
- [Ora11] Oracle  
 “java.lang.String class”  
<download.oracle.com/javase/1.4.2/docs/api/java/lang/String.html>
- [Que94] Queinnec, Christian  
 “Lisp in Small Pieces”  
 Cambridge University Press (1994) ISBN 0-521-56247-3
- [Ste09] Steele, Guy  
 “Organizing Functional Code for Execution”  
<vimeo.com/6624203>
- [Ste11] Steele, Guy  
 “How to Think about Parallel Programming, Not!”  
 International Conference on Functional Programming (ICFP) Edinburgh  
 (2009)  
<www.infoq.com/presentations/Thinking-Parallel-Programming>
- [Web11] Web  
 “Clojure’s unconventional symbols”  
<arcanesentiment.blogspot.com/2011/01/clojures-unconventional-symbols.html>
- [Wiki] Wikipedia  
 “Red-black tree”  
[en.wikipedia.org/wiki/Red-black\\_tree](en.wikipedia.org/wiki/Red-black_tree)
- [Wiki1] Wikipedia  
 “The ML programming language”  
[en.wikipedia.org/wiki/ML\\_\(programming\\_language\)](en.wikipedia.org/wiki/ML_(programming_language))
- [1] Wu, Kai  
 “An example of literate programming in Clojure”  
<limist.com/coding/an-example-of-literate-programming-in-clojure-using-emacsorg.html>

# Index

- AbstractMap, 1723
  - extended by 1145, 1145
  - Class, 1723
- AdviceAdapter, 413
  - extends 453, 413
  - implements 387, 413
  - Class, 413
- AFn, 509
  - extended by 519, 519
  - extended by 530, 530
  - extended by 538, 538
  - extended by 541, 541
  - extended by 579, 579
  - extended by 581, 581
  - extended by 82, 82
  - extended by 81, 81
  - extended by 74, 74
  - extended by 89, 89
  - extended by 83, 83
  - extended by 87, 87
  - extended by 86, 86
  - extended by 80, 80
  - extended by 81, 81
  - extended by 75, 75
  - extended by 852, 852
  - extended by 85, 85
  - extended by 87, 87
  - extended by 72, 72
  - extended by 1143, 1143
  - extended by 76, 76
  - extended by 80, 80
  - extended by 79, 79
  - extended by 89, 89
  - extended by 85, 85
  - extended by 80, 80
  - extended by 74, 74
  - implements 774, 509
  - Class, 509
- AFunction, 519
  - extended by 1055, 1055
  - extends 509, 519
  - implements 1723, 519
  - implements 772, 519

- implements 800, 519  
implements 1723, 519  
Class, 519
- Agent, 520
  - extends 553, 520
  - Class, 520
- AMapEntry, 527
  - extended by 850, 850
  - extended by 41, 41
  - extends 541, 527
  - implements 798, 527
  - Class, 527
- AnalyzerAdapter, 426
  - extends 314, 426
  - Class, 426
- AnnotationVisitor, 163
  - implemented by 165, 165
  - implemented by 449, 449
  - Interface, 163
- AnnotationWriter, 165
  - implements 163, 165
  - Class, 165
- APersistentMap, 530
  - extended by 995, 995
  - extended by 1000, 1000
  - extends 509, 530
  - implements 801, 530
  - implements 1723, 530
  - implements 850, 530
  - implements 1723, 530
  - implements 1723, 530
  - Class, 530
- APersistentSet, 538
  - extended by 980, 980
  - extended by 1012, 1012
  - extends 509, 538
  - implements 1723, 538
  - implements 802, 538
  - implements 1723, 538
  - implements 1723, 538
  - Class, 538
- APersistentVector, 541
  - extended by 527, 527
  - extended by 1014, 1014
  - extends 509, 541

- implements 1723, 541
- implements 802, 541
- implements 1723, 541
- Class, 541
- ARef, 553
  - extended by 520, 520
  - extended by 577, 577
  - extended by 1034, 1034
  - extended by 1152, 1152
  - extends 552, 553
  - implements 805, 553
  - Class, 553
- AReference, 552
  - extended by 553, 553
  - extended by 861, 861
  - implements 804, 552
  - Class, 552
- ArgReader, 71
  - extends 509, 82
- ArgReader, class in 825, 82
- ArityException, 556
  - extends 1723, 556
  - Class, 556
- ArrayChunk, 556
  - implements 772, 556
  - implements 1723, 556
  - Class, 556
- ArrayNode, 58
  - implements 58, 955
- ArrayNode, class in 969, 955
- ArrayNode.find, 58
- ArraySeq, 558
  - extends 571, 558
  - implements 804, 558
  - implements 799, 558
  - Class, 558
- ASeq, 571
  - extended by 558, 558
  - extended by 586, 586
  - extended by 766, 766
  - extended by 770, 770
  - extended by 806, 806

- extended by 982, 982
- extended by 1000, 60
- extended by 1031, 1031
- extended by 1141, 1141
- extends 947, 571
- implements 805, 571
- implements 1723, 571
- implements 1723, 571
- Class, 571
- Associative, 100, 576
  - extended by 801, 801
  - extended by 802, 802
  - extends 797, 576
  - extends 800, 576
  - Interface, 576
- Atom, 577
  - extends 553, 577
  - Class, 577
- ATransientMap, 579
  - extends 509, 579
  - implements 808, 579
  - Class, 579
- ATransientSet, 581
  - extends 509, 581
  - implements 809, 581
  - Class, 581
- Attribute, 171
  - Class, 171
- balanceLeftDel, method in 1000, 53
- balanceRightDel, method in 1000, 52
- BigDecimal, 63
  - Class, 63
- BigInt, 582
  - extends 1723, 582
  - Class, 582
- BigInteger, 63
  - Class, 63
- Binding, 585
  - Class, 585
- bit-partitioning, 135
- BitmapIndexedNode, 59
  - implements 58, 959
- BitmapIndexedNode, class in 969, 959
- Black

- extended by 44, 44
- extended by 43, 43
- extends 41, 42
  - black, 42–45
  - Black, class in 1000, 42
  - black, method in 1000, 45
  - BlackBranch
    - extended by 45, 45
    - extends 42, 44
  - BlackBranch, class in 1000, 44
  - BlackBranchVal
    - extends 44, 45
  - BlackBranchVal, class in 1000, 45
  - BlackVal
    - extends 42, 43
  - BlackVal, class in 1000, 43
  - Boolean, 63
    - Class, 63
  - Box, 585
    - Class, 585
  - ByteVector, 176
    - Class, 176
- Callable, 1723
  - extended by 774, 774
  - Interface, 1723
- CancellationException, 97
- Character, 63
  - Class, 63
- CharacterReader, 71
  - extends 509, 81
- CharacterReader, class in 825, 81
- ChunkBuffer, 586
  - implements 768, 586
  - Class, 586
- ChunkedCons, 586
  - extends 571, 586
  - implements 773, 586
  - Class, 586
- Class
  - AbstractMap, 1723
  - AdviceAdapter, 413
  - AFn, 509
  - AFunction, 519

Agent, 520  
AMapEntry, 527  
AnalyzerAdapter, 426  
AnnotationWriter, 165  
APersistentMap, 530  
APersistentSet, 538  
APersistentVector, 541  
ARef, 553  
AReference, 552  
ArgReader in 825, 82  
ArityException, 556  
ArrayChunk, 556  
ArrayNode in 969, 955  
ArraySeq, 558  
ASeq, 571  
Atom, 577  
ATransientMap, 579  
ATransientSet, 581  
Attribute, 171  
BigDecimal, 63  
BigInt, 582  
BigInteger, 63  
Binding, 585  
BitmapIndexedNode in 969, 959  
Black in 1000, 42  
BlackBranch in 1000, 44  
BlackBranchVal in 1000, 45  
BlackVal in 1000, 43  
Boolean, 63  
Box, 585  
ByteVector, 176  
Character, 63  
CharacterReader in 825, 81  
ChunkBuffer, 586  
ChunkedCons, 586  
ClassAdapter, 183  
ClassReader, 185  
ClassWriter, 233  
CodeSizeEvaluator, 445  
CommentReader in 825, 74  
Compile, 588  
Compiler, 590  
Cons, 766  
DefaultHandler, 1723  
Delay, 768  
DiscardReader in 825, 89

DispatchReader in 825, 83  
Double, 63  
DynamicClassLoader, 769  
Edge, 262  
EmptyVisitor, 449  
EnumerationSeq, 770  
EvalReader in 825, 87  
FieldWriter, 264  
FnReader in 825, 86  
Frame, 269  
GeneratorAdapter, 453  
Handler, 300  
HashCollisionNode in 969, 965  
IllegalArgumentException, 1723  
Item, 301  
IteratorSeq, 806  
KeyIterator in 1000, 103  
Keyword, 810  
KeywordLookupSite, 816  
Label, 305  
LazilyPersistentVector, 817  
LazySeq, 818  
LineNumberingPushbackReader, 823  
LispReader, 825  
ListReader in 825, 80  
LocalVariablesSorter, 484  
LockingTransaction, 836  
Long, 63  
main, 1167  
MapEntry, 850  
MapReader in 825, 81  
MetaReader in 825, 75  
Method, 491  
MethodAdapter, 314  
MethodImplCache, 851  
MethodWriter, 326  
MultiFn, 852  
Namespace, 861  
Node, 41  
NodeIterator in 1000, 103  
Number, 1723  
Obj, 947  
Pattern, 63  
PersistentArrayMap, 948  
PersistentHashMap, 969  
PersistentHashSet, 980

PersistentList, 982  
PersistentQueue, 989  
PersistentStructMap, 995  
PersistentTreeMap, 1000  
PersistentTreeSet, 1012  
PersistentVector, 1014  
ProxyHandler, 1030  
PushbackReader, 1723  
Range, 1031  
Ratio, 1033  
Red in 1000, 46  
RedBranch in 1000, 48  
RedBranchVal in 1000, 49  
RedVal in 1000, 47  
Ref, 1034  
Reflector, 1044  
RegexReader in 825, 85  
Repl, 1054  
RestFn, 1055  
RT, 1094  
Script, 1138  
Seq in 1000, 60  
SeqEnumeration, 1138  
SeqIterator, 1139  
SerialVersionUIDAdder, 496  
SetReader in 825, 87  
StaticInitMerger, 506  
String, 63  
StringReader in 825, 72  
StringSeq, 1141  
Symbol, 1143  
SyntaxQuoteReader in 825, 76  
TransactionalHashMap, 1145  
Type, 394  
UnmatchedDelimiterReader in 825, 80  
UnquoteReader in 825, 79  
UnreadableReader in 825, 89  
URLClassLoader, 1723  
Util, 1149  
ValIterator in 1000, 104  
Var, 1152  
VarReader in 825, 85  
VectorReader in 825, 80  
WrappingReader in 825, 74  
XMLHandler, 1164

- ClassAdapter, 183
  - extended by 496, 496
  - extended by 506, 506
  - implements 229, 183
  - Class, 183
- ClassReader, 185
  - Class, 185
- ClassVisitor, 229
  - implemented by 183, 183
  - implemented by 233, 233
  - implemented by 449, 449
  - Interface, 229
- ClassWriter, 233
  - implements 229, 233
  - Class, 233
- CodeSizeEvaluator, 445
  - extends 314, 445
  - implements 387, 445
  - Class, 445
- Collection, 1723
  - implemented by 538, 538
  - implemented by 989, 989
  - Interface, 1723
- CommentReader, 71, 84
  - extends 509, 74
- CommentReader, class in 825, 74
- Comparable, 1723
  - implemented by 541, 541
  - implemented by 810, 810
  - implemented by 1033, 1033
  - implemented by 1034, 1034
  - implemented by 1143, 1143
  - Interface, 1723
- Comparator, 1723
  - implemented by 519, 519
  - Interface, 1723
- Compile, 588
  - Class, 588
- Compiler, 590
  - implements 387, 590
  - Class, 590
- ConcurrentMap, 1723
  - implemented by 1145, 1145
  - Interface, 1723

- Cons, 766
  - extends 571, 766
  - implements 1723, 766
  - Class, 766
- Counted, 100, 768
  - extended by 801, 801
  - extended by 802, 802
  - extended by 808, 808
  - extended by 809, 809
  - extended by 799, 799
  - extended by 799, 799
  - implemented by 586, 586
  - implemented by 982, 982
  - implemented by 989, 989
  - implemented by 1031, 1031
  - Interface, 768
- DefaultHandler, 1723
  - extended by 1164, 1164
  - Class, 1723
- Delay, 768
  - implements 773, 768
  - Class, 768
- DiscardReader, 84
  - extends 509, 89
- DiscardReader, class in 825, 89
- Dispatch Macro Table, 71, 84
- dispatchMacros, 83, 84
- DispatchReader, 72, 83
  - extends 509, 83
- DispatchReader, class in 825, 83
- Double, 63
  - Class, 63
- DynamicClassLoader, 769
  - extends 1723, 769
  - Class, 769
- Edge, 262
  - Class, 262
- EmptyVisitor, 449
  - implements 163, 449
  - implements 229, 449
  - implements 263, 449
  - implements 317, 449
  - Class, 449

- Enumeration, 1723
  - implemented by 1138, 1138
  - Interface, 1723
- EnumerationSeq, 770
  - extends 571, 770
  - Class, 770
- equals, method in 1143, 69
- EvalReader, 84
  - extends 509, 87
- EvalReader, class in 825, 87
- exceptions, 95, 97
- Extends
  - AbstractMap, by TransactionalHashMap, 1145
  - AFn, by AFunction, 519
  - AFn, by APersistentMap, 530
  - AFn, by APersistentSet, 538
  - AFn, by APersistentVector, 541
  - AFn, by ArgReader, 82
  - AFn, by ATTransientMap, 579
  - AFn, by ATTransientSet, 581
  - AFn, by CharacterReader, 81
  - AFn, by CommentReader, 74
  - AFn, by DiscardReader, 89
  - AFn, by DispatchReader, 83
  - AFn, by EvalReader, 87
  - AFn, by FnReader, 86
  - AFn, by ListReader, 80
  - AFn, by MapReader, 81
  - AFn, by MetaReader, 75
  - AFn, by MultiFn, 852
  - AFn, by RegexReader, 85
  - AFn, by SetReader, 87
  - AFn, by StringReader, 72
  - AFn, by Symbol, 1143
  - AFn, by SyntaxQuoteReader, 76
  - AFn, by UnmatchedDelimiterReader, 80
  - AFn, by UnquoteReader, 79
  - AFn, by UnreadableReader, 89
  - AFn, by VarReader, 85
  - AFn, by VectorReader, 80
  - AFn, by WrappingReader, 74
- AFunction, by RestFn, 1055
- AMapEntry, by MapEntry, 850
- AMapEntry, by Node, 41
- APersistentMap, by PersistentStructMap, 995

APersistentMap, by PersistentTreeMap, 1000  
APersistentSet, by PersistentHashSet, 980  
APersistentSet, by PersistentTreeSet, 1012  
APersistentVector, by AMapEntry, 527  
APersistentVector, by PersistentVector, 1014  
ARef, by Agent, 520  
ARef, by Atom, 577  
ARef, by Ref, 1034  
ARef, by Var, 1152  
AReference, by ARef, 553  
AReference, by Namespace, 861  
ASeq, by ArraySeq, 558  
ASeq, by ChunkedCons, 586  
ASeq, by Cons, 766  
ASeq, by EnumerationSeq, 770  
ASeq, by IteratorSeq, 806  
ASeq, by PersistentList, 982  
ASeq, by PersistentTreeMap, 60  
ASeq, by Range, 1031  
ASeq, by StringSeq, 1141  
Associative, by IPersistentMap, 801  
Associative, by IPersistentVector, 802  
Black, by BlackBranch, 44  
Black, by BlackVal, 43  
BlackBranch, by BlackBranchVal, 45  
Callable, by IFn, 774  
ClassAdapter, by serialVersionUIDAdder, 496  
ClassAdapter, by StaticInitMerger, 506  
Counted, by Indexed, 799  
Counted, by IndexedSeq, 799  
Counted, by IPersistentMap, 801  
Counted, by IPersistentSet, 802  
Counted, by ITransientMap, 808  
Counted, by ITransientSet, 809  
DefaultHandler, by XMLHandler, 1164  
GeneratorAdapter, by AdviceAdapter, 413  
IDeref, by IRef, 805  
IllegalArgumentException, by ArityException, 556  
ILookup, by Associative, 576  
ILookup, by ITransientAssociative, 807  
IMeta, by IObj, 800  
IMeta, by IReference, 804  
Indexed, by IChunk, 772  
Indexed, by IPersistentVector, 802  
Indexed, by ITransientVector, 809  
IPersistentCollection, by Associative, 576

- IPersistentCollection, by IPersistentSet, 802
- IPersistentCollection, by IPersistentStack, 802
- IPersistentCollection, by ISeq, 805
- IPersistentStack, by IPersistentList, 801
- IPersistentStack, by IPersistentVector, 802
- ISeq, by IChunkedSeq, 773
- ISeq, by IndexedSeq, 799
- Iterable, by IPersistentMap, 801
- ITransientAssociative, by ITransientMap, 808
- ITransientAssociative, by ITransientVector, 809
- ITransientCollection, by ITransientAssociative, 807
- ITransientCollection, by ITransientSet, 809
- LocalVariablesSorter, by GeneratorAdapter, 453
- Map.Entry, by IMapEntry, 798
- MethodAdapter, by AnalyzerAdapter, 426
- MethodAdapter, by CodeSizeEvaluator, 445
- MethodAdapter, by LocalVariablesSorter, 484
- Node, by Black, 42
- Node, by Red, 46
- Number, by BigInt, 582
- Number, by Ratio, 1033
- Obj, by ASeq, 571
- Obj, by LazySeq, 818
- Obj, by PersistentQueue, 989
- PushbackReader, by LineNumberingPushbackReader, 823
- Red, by RedBranch, 48
- Red, by RedVal, 47
- RedBranch, by RedBranchVal, 49
- Reversible, by IPersistentVector, 802
- Runnable, by IFn, 774
- Seqable, by IPersistentCollection, 800
- Sequential, by IPersistentList, 801
- Sequential, by IPersistentVector, 802
- Sequential, by ISeq, 805
- Serializable, by INode, 58
- URLClassLoader, by DynamicClassLoader, 769
- FieldVisitor, 263
  - implemented by 449, 449
  - implemented by 264, 264
  - Interface, 263
- FieldWriter, 264
  - implements 263, 264
  - Class, 264
- find, method in 955, 58
- Fn, 772

- implemented by 519, 519  
    Interface, 772
- FnReader, 84
  - extends 509, 86
- FnReader, class in 825, 86
- Frame, 269
  - Class, 269
- future, 97, 98
- GeneratorAdapter, 453
  - extended by 413, 413
  - extends 484, 453
  - Class, 453
- Handler, 300
  - Class, 300
- HashCollisionNode, 59
  - implements 58, 965
- HashCollisionNode, class in 969, 965
- IChunk, 772
  - extends 799, 772
  - implemented by 556, 556
  - Interface, 772
- IChunkedSeq, 773
  - extends 805, 773
  - implemented by 586, 586
  - Interface, 773
- IDeref, 773
  - extended by 805, 805
  - implemented by 768, 768
  - Interface, 773
- IEitableCollection, 774
  - implemented by 948, 948
  - implemented by 969, 969
  - implemented by 980, 980
  - implemented by 1014, 1014
  - Interface, 774
- IFn, 71, 84, 774
  - extends 1723, 774
  - extends 1723, 774
  - implemented by 509, 509
  - implemented by 810, 810
  - implemented by 1034, 1034
  - implemented by 1152, 1152
  - Interface, 774

- IKeywordLookup, 100, 796
  - Interface, 796
- IllegalArgumentException, 1723
  - extended by 556, 556
  - Class, 1723
- IllegalStateException, 95
- ILookup, 100, 797
  - extended by 576, 576
  - extended by 807, 807
  - Interface, 797
- ILookupSite, 797
  - implemented by 816, 816
  - Interface, 797
- ILookupThunk, 798
  - implemented by 816, 816
  - Interface, 798
- IMapEntry, 798
  - extends 1723, 798
  - implemented by 527, 527
  - Interface, 798
- IMeta, 100, 799
  - extended by 800, 800
  - extended by 804, 804
  - Interface, 799
- Implements
  - AnnotationVisitor, by AnnotationWriter, 165
  - AnnotationVisitor, by EmptyVisitor, 449
  - ClassVisitor, by ClassAdapter, 183
  - ClassVisitor, by ClassWriter, 233
  - ClassVisitor, by EmptyVisitor, 449
  - Collection, by APersistentSet, 538
  - Collection, by PersistentQueue, 989
  - Comparable, by APersistentVector, 541
  - Comparable, by Keyword, 810
  - Comparable, by Ratio, 1033
  - Comparable, by Ref, 1034
  - Comparable, by Symbol, 1143
  - Comparator, by AFunction, 519
  - ConcurrentMap, by TransactionalHashMap, 1145
  - Counted, by ChunkBuffer, 586
  - Counted, by PersistentList, 982
  - Counted, by PersistentQueue, 989
  - Counted, by Range, 1031
  - Enumeration, by SeqEnumeration, 1138
  - FieldVisitor, by EmptyVisitor, 449

FieldVisitor, by FieldWriter, 264  
Fn, by AFunction, 519  
IChunk, by ArrayChunk, 556  
IChunkedSeq, by ChunkedCons, 586  
IDeref, by Delay, 768  
IEitableCollection, by PersistentArrayList, 948  
IEitableCollection, by PersistentHashMap, 969  
IEitableCollection, by PersistentHashSet, 980  
IEitableCollection, by PersistentVector, 1014  
IFn, by AFn, 509  
IFn, by Keyword, 810  
IFn, by Ref, 1034  
IFn, by Var, 1152  
ILookupSite, by KeywordLookupSite, 816  
ILookupThunk, by KeywordLookupSite, 816  
IMapEntry, by AMapEntry, 527  
IndexedSeq, by ArraySeq, 558  
IndexedSeq, by StringSeq, 1141  
INode, by ArrayNode, 955  
INode, by BitmapIndexedNode, 959  
INode, by HashCollisionNode, 965  
InvocationHandler, by ProxyHandler, 1030  
IObj, by AFunction, 519  
IObj, by Obj, 947  
IObj, by PersistentArrayList, 948  
IObj, by PersistentHashMap, 969  
IObj, by PersistentHashSet, 980  
IObj, by PersistentStructMap, 995  
IObj, by PersistentTreeMap, 1000  
IObj, by PersistentTreeSet, 1012  
IObj, by PersistentVector, 1014  
IObj, by Symbol, 1143  
IPersistentList, by PersistentList, 982  
IPersistentList, by PersistentQueue, 989  
IPersistentMap, by APersistentMap, 530  
IPersistentSet, by APersistentSet, 538  
IPersistentVector, by APersistentVector, 541  
IReduce, by ArraySeq, 558  
IReduce, by PersistentList, 982  
IReduce, by Range, 1031  
IRef, by ARef, 553  
IRef, by Ref, 1034  
IRef, by Var, 1152  
IReference, by AReference, 552  
ISeq, by ASseq, 571  
ISeq, by LazySeq, 818

- Iterable, by APersistentMap, 530
- Iterable, by APersistentVector, 541
- Iterator, by KeyIterator, 103
- Iterator, by NodeIterator, 103
- Iterator, by SeqIterator, 1139
- Iterator, by ValIterator, 104
- ITransientMap, by ATransientMap, 579
- ITransientSet, by ATransientSet, 581
- List, by APersistentVector, 541
- List, by ASeq, 571
- List, by LazySeq, 818
- List, by PersistentList, 982
- Map, by APersistentMap, 530
- MapEquivalence, by APersistentMap, 530
- MethodVisitor, by EmptyVisitor, 449
- MethodVisitor, by MethodAdapter, 314
- MethodVisitor, by MethodWriter, 326
- Named, by Keyword, 810
- Named, by Symbol, 1143
- Opcodes, by AdviceAdapter, 413
- Opcodes, by CodeSizeEvaluator, 445
- Opcodes, by Compiler, 590
- RandomAccess, by APersistentVector, 541
- Reversible, by PersistentTreeMap, 1000
- Reversible, by PersistentTreeSet, 1012
- Serializable, by AFunction, 519
- Serializable, by APersistentMap, 530
- Serializable, by APersistentSet, 538
- Serializable, by APersistentVector, 541
- Serializable, by ArrayChunk, 556
- Serializable, by ASeq, 571
- Serializable, by Cons, 766
- Serializable, by Keyword, 810
- Serializable, by Namespace, 861
- Serializable, by Obj, 947
- Serializable, by Symbol, 1143
- Set, by APersistentSet, 538
- Settable, by Var, 1152
- Sorted, by PersistentTreeMap, 1000
- Sorted, by PersistentTreeSet, 1012
- Indexed, 799
  - extended by 772, 772
  - extended by 802, 802
  - extended by 809, 809
  - extends 768, 799

- Interface, 799
- IndexedSeq, 799
  - extends 768, 799
  - extends 805, 799
  - implemented by 558, 558
  - implemented by 1141, 1141
  - Interface, 799
- INode, 58
  - extends 1723, 58
  - implemented by 955, 955
  - implemented by 959, 959
  - implemented by 965, 965
  - Interface, 58
- Interface
  - AnnotationVisitor, 163
  - Associative, 576
  - Callable, 1723
  - ClassVisitor, 229
  - Collection, 1723
  - Comparable, 1723
  - Comparator, 1723
  - ConcurrentMap, 1723
  - Counted, 768
  - Enumeration, 1723
  - FieldVisitor, 263
  - Fn, 772
  - IChunk, 772
  - IChunkedSeq, 773
  - IDeref, 773
  - IEditableCollection, 774
  - IFn, 774
  - IKeywordLookup, 796
  - ILookup, 797
  - ILookupSite, 797
  - ILookupThunk, 798
  - IMapEntry, 798
  - IMeta, 799
  - Indexed, 799
  - IndexedSeq, 799
  - INode, 58
  - InvocationHandler, 1723
  - IObj, 800
  - IPersistentCollection, 800
  - IPersistentList, 801
  - IPersistentMap, 801

- IPersistentSet, 802
- IPersistentStack, 802
- IPersistentVector, 802
- IPromiseImpl, 803
- IProxy, 803
- IReduce, 804
- IRef, 805
- IReference, 804
- ISeq, 805
- Iterable, 1723
- Iterator, 1723
- ITransientAssociative, 807
- ITransientCollection, 808
- ITransientMap, 808
- ITransientSet, 809
- ITransientVector, 809
- List, 1723
- Map, 1723
- Map.Entry, 1723
- MapEquivalence, 850
- MethodVisitor, 317
- Named, 861
- Numbers, 867
- Opcodes, 387
- RandomAccess, 1723
- Reversible, 1094
- Runnable, 1723
- Seqable, 1140
- Sequential, 1140
- Serializable, 1723
- Set, 1723
- Settable, 1140
- Sorted, 1141
- TableSwitchGenerator, 508
- intern(2), method in 1143, 68
- intern, method in 1143, 67
- InvocationHandler, 1723
  - implemented by 1030, 1030
  - Interface, 1723
- IObj, 100, 800
  - extends 799, 800
  - implemented by 519, 519
  - implemented by 947, 947
  - implemented by 948, 948
  - implemented by 969, 969

- implemented by 980, 980
- implemented by 995, 995
- implemented by 1000, 1000
- implemented by 1012, 1012
- implemented by 1014, 1014
- implemented by 1143, 1143
- Interface, 800
- IPersistentCollection, 100, 800
  - extended by 576, 576
  - extended by 802, 802
  - extended by 802, 802
  - extended by 805, 805
  - extends 1140, 800
  - Interface, 800
- IPersistentList, 801
  - extends 802, 801
  - extends 1140, 801
  - implemented by 982, 982
  - implemented by 989, 989
  - Interface, 801
- IPersistentMap, 100, 801
  - extends 576, 801
  - extends 768, 801
  - extends 1723, 801
  - implemented by 530, 530
  - Interface, 801
- IPersistentSet, 802
  - extends 768, 802
  - extends 800, 802
  - implemented by 538, 538
  - Interface, 802
- IPersistentStack, 802
  - extended by 801, 801
  - extended by 802, 802
  - extends 800, 802
  - Interface, 802
- IPersistentVector, 802
  - extends 576, 802
  - extends 802, 802
  - extends 799, 802
  - extends 1094, 802
  - extends 1140, 802
  - implemented by 541, 541
  - Interface, 802
- IPromiseImpl, 803

- Interface, 803
- IProxy, 803
  - Interface, 803
- IReduce, 804
  - implemented by 558, 558
  - implemented by 982, 982
  - implemented by 1031, 1031
  - Interface, 804
- IRef, 805
  - extends 773, 805
  - implemented by 553, 553
  - implemented by 1034, 1034
  - implemented by 1152, 1152
  - Interface, 805
- IReference, 804
  - extends 799, 804
  - implemented by 552, 552
  - Interface, 804
- ISeq, 805
  - extended by 773, 773
  - extended by 799, 799
  - extends 800, 805
  - extends 1140, 805
  - implemented by 571, 571
  - implemented by 818, 818
  - Interface, 805
- Item, 301
  - Class, 301
- Iterable, 1723
  - extended by 801, 801
  - implemented by 530, 530
  - implemented by 541, 541
  - Interface, 1723
- Iterator, 1723
  - implemented by 103, 103
  - implemented by 103, 103
  - implemented by 1139, 1139
  - implemented by 104, 104
  - Interface, 1723
- Iterators, 55
- IteratorSeq, 806
  - extends 571, 806
  - Class, 806
- ITransientAssociative, 807
  - extended by 808, 808

- extended by 809, 809
- extends 797, 807
- extends 808, 807
- Interface, 807
  - ITransientCollection, 808
    - extended by 807, 807
    - extended by 809, 809
    - Interface, 808
  - ITransientMap, 808
    - extends 768, 808
    - extends 807, 808
    - implemented by 579, 579
    - Interface, 808
  - ITransientSet, 809
    - extends 768, 809
    - extends 808, 809
    - implemented by 581, 581
    - Interface, 809
  - ITransientVector, 809
    - extends 807, 809
    - extends 799, 809
    - Interface, 809
  - KeyIterator
    - implements 1723, 103
  - KeyIterator, class in 1000, 103
  - Keyword, 810
    - implements 1723, 810
    - implements 774, 810
    - implements 861, 810
    - implements 1723, 810
    - Class, 810
  - KeywordLookupSite, 816
    - implements 797, 816
    - implements 798, 816
    - Class, 816
  - Label, 305
    - Class, 305
  - LazilyPersistentVector, 817
    - Class, 817
  - LazySeq, 818
    - extends 947, 818
    - implements 805, 818
    - implements 1723, 818
    - Class, 818

- leftBalance, method in 1000, 54
- LineNumberingPushbackReader, 823
  - extends 1723, 823
  - Class, 823
- LispReader, 83, 825
  - Class, 825
- LispReader.read, 69
- LispReader\$ArgReader, 82
- LispReader\$CharacterReader, 81
- LispReader\$CommentReader, 74
- LispReader\$DiscardReader, 89
- LispReader\$DispatchReader, 83
- LispReader\$EvalReader, 87
- LispReader\$FnReader, 86
- LispReader\$ListReader, 80
- LispReader\$MapReader, 81
- LispReader\$MetaReader, 75
- LispReader\$RegexReader, 85
- LispReader\$SetReader, 87
- LispReader\$StringReader, 72
- LispReader\$SyntaxQuoteReader, 76
- LispReader\$UnmatchedDelimiterReader, 80
- LispReader\$UnquoteReader, 79
- LispReader\$UnreadableReader, 89
- LispReader\$VarReader, 85
- LispReader\$VectorReader, 80
- LispReader\$WrappingReader, 74
- List, 1723
  - implemented by 541, 541
  - implemented by 571, 571
  - implemented by 818, 818
  - implemented by 982, 982
  - Interface, 1723
- ListReader, 71
  - extends 509, 80
- ListReader, class in 825, 80
- LocalVariablesSorter, 484
  - extended by 453, 453
  - extends 314, 484
  - Class, 484
- LockingTransaction, 836
  - Class, 836
- Long, 63

- Class, 63
  - main, 1167
    - Class, 1167
  - Makefile, i, ii, 1737
  - Map, 1723
    - implemented by 530, 530
    - Interface, 1723
  - Map.Entry, 1723
    - extended by 798, 798
    - Interface, 1723
  - MapEntry, 850
    - extends 527, 850
    - Class, 850
  - MapEquivalence, 850
    - implemented by 530, 530
    - Interface, 850
  - MapReader, 71
    - extends 509, 81
  - MapReader, class in 825, 81
  - mask, method in 969, 57
  - MetaReader, 71, 84
    - extends 509, 75
  - MetaReader, class in 825, 75
  - Method, 491
    - Class, 491
  - MethodAdapter, 314
    - extended by 426, 426
    - extended by 445, 445
    - extended by 484, 484
    - implements 317, 314
    - Class, 314
  - MethodImplCache, 851
    - Class, 851
  - MethodVisitor, 317
    - implemented by 449, 449
    - implemented by 314, 314
    - implemented by 326, 326
    - Interface, 317
  - MethodWriter, 326
    - implements 317, 326
    - Class, 326
  - MultiFn, 852
    - extends 509, 852
    - Class, 852

- Named, 861
  - implemented by 810, 810
  - implemented by 1143, 1143
- Interface, 861
- Namespace, 861
  - extends 552, 861
  - implements 1723, 861
- Class, 861
- Node, 41
  - extended by 42, 42
  - extended by 46, 46
  - extends 527, 41
- Class, 41
- NodeIterator
  - implements 1723, 103
- NodeIterator, class in 1000, 103
- Number, 1723
  - extended by 582, 582
  - extended by 1033, 1033
- Class, 1723
- Numbers, 867
  - Interface, 867
- Obj, 947
  - extended by 571, 571
  - extended by 818, 818
  - extended by 989, 989
  - implements 800, 947
  - implements 1723, 947
- Class, 947
- Opcodes, 387
  - implemented by 413, 413
  - implemented by 445, 445
  - implemented by 590, 590
- Interface, 387
- Pattern, 63
  - Class, 63
- PersistentArrayMap, 948
  - implements 774, 948
  - implements 800, 948
- Class, 948
- PersistentHashMap, 57, 969
  - implements 774, 969
  - implements 800, 969
- Class, 969

- PersistentHashMap.mask, 57
- PersistentHashMap\$ArrayNode, 955
- PersistentHashMap\$BitmapIndexedNode, 959
- PersistentHashMap\$HashCollisionNode, 965
- PersistentHashSet, 980
  - extends 538, 980
  - implements 774, 980
  - implements 800, 980
  - Class, 980
- PersistentList, 982
  - extends 571, 982
  - implements 768, 982
  - implements 801, 982
  - implements 804, 982
  - implements 1723, 982
  - Class, 982
- PersistentQueue, 989
  - extends 947, 989
  - implements 1723, 989
  - implements 768, 989
  - implements 801, 989
  - Class, 989
- PersistentStructMap, 995
  - extends 530, 995
  - implements 800, 995
  - Class, 995
- PersistentTreeMap, 55, 1000
  - extends 530, 1000
  - extends 571, 60
  - implements 800, 1000
  - implements 1094, 1000
  - implements 1141, 1000
  - Class, 1000
- PersistentTreeMap.balanceLeftDel, 53
- PersistentTreeMap.balanceRightDel, 52
- PersistentTreeMap.black, 45
- PersistentTreeMap.leftBalance, 54
- PersistentTreeMap.red, 50
- PersistentTreeMap.remove, 1000
- PersistentTreeMap.removeRight, 42
- PersistentTreeMap.replace, 55
- PersistentTreeMap.rightBalance, 54
- PersistentTreeMap\$Black, 42
- PersistentTreeMap\$BlackBranch, 44

- PersistentTreeMap\$BlackBranchVal, 45
- PersistentTreeMap\$BlackVal, 43
- PersistentTreeMap\$KeyIterator, 103
- PersistentTreeMap\$NodeIterator, 103
- PersistentTreeMap\$Red, 46
- PersistentTreeMap\$RedBranch, 48
- PersistentTreeMap\$RedBranchVal, 49
- PersistentTreeMap\$RedVal, 47
- PersistentTreeMap\$Seq, 60
- PersistentTreeMap\$ValIterator, 104
- PersistentTreeSet, 1012
  - extends 538, 1012
  - implements 800, 1012
  - implements 1094, 1012
  - implements 1141, 1012
  - Class, 1012
- PersistentVector, 1014
  - extends 541, 1014
  - implements 774, 1014
  - implements 800, 1014
  - Class, 1014
- promise, 95
- ProxyHandler, 1030
  - implements 1723, 1030
  - Class, 1030
- PushbackReader, 1723
  - extended by 823, 823
  - Class, 1723
- RandomAccess, 1723
  - implemented by 541, 541
  - Interface, 1723
- Range, 1031
  - extends 571, 1031
  - implements 768, 1031
  - implements 804, 1031
  - Class, 1031
- Ratio, 1033
  - extends 1723, 1033
  - implements 1723, 1033
  - Class, 1033
- read, method in 825, 69
- Red
  - extended by 48, 48

- extended by 47, 47
- extends 41, 46
- red, 46–49
- Red, class in 1000, 46
- red, method in 1000, 50
- RedBranch
  - extended by 49, 49
  - extends 46, 48
- RedBranch, class in 1000, 48
- RedBranchVal
  - extends 48, 49
- RedBranchVal, class in 1000, 49
- RedVal
  - extends 46, 47
- RedVal, class in 1000, 47
- Ref, 1034
  - extends 553, 1034
  - implements 1723, 1034
  - implements 774, 1034
  - implements 805, 1034
  - Class, 1034
- Reflector, 1044
  - Class, 1044
- RegexReader, 84
  - extends 509, 85
- RegexReader, class in 825, 85
- remove, 52
- remove, method in 1000, 1000
- removeRight, 52
- removeRight, method in 1000, 42
- Repl, 1054
  - Class, 1054
- replace, method in 1000, 55
- RestFn, 1055
  - extends 519, 1055
  - Class, 1055
- Reversible, 1094
  - extended by 802, 802
  - implemented by 1000, 1000
  - implemented by 1012, 1012
  - Interface, 1094
- rightBalance, method in 1000, 54
- RT, 1094
  - Class, 1094

- Runnable, 1723
  - extended by 774, 774
  - Interface, 1723
- Script, 1138
  - Class, 1138
- Seq, class in 1000, 60
- Seqable, 100, 1140
  - extended by 800, 800
  - Interface, 1140
- SeqEnumeration, 1138
  - implements 1723, 1138
  - Class, 1138
- SeqIterator, 1139
  - implements 1723, 1139
  - Class, 1139
- Seqs, 55
- Sequential, 1140
  - extended by 801, 801
  - extended by 802, 802
  - extended by 805, 805
  - Interface, 1140
- Serializable, 1723
  - extended by 58, 58
  - implemented by 519, 519
  - implemented by 530, 530
  - implemented by 538, 538
  - implemented by 541, 541
  - implemented by 571, 571
  - implemented by 556, 556
  - implemented by 766, 766
  - implemented by 810, 810
  - implemented by 861, 861
  - implemented by 947, 947
  - implemented by 1143, 1143
  - Interface, 1723
- serialVersionUIDAdder, 496
  - extends 183, 496
  - Class, 496
- Set, 1723
  - implemented by 538, 538
  - Interface, 1723
- SetReader, 84
  - extends 509, 87
- SetReader, class in 825, 87

- Settable, 1140
  - implemented by 1152, 1152
  - Interface, 1140
- Sorted, 1141
  - implemented by 1000, 1000
  - implemented by 1012, 1012
  - Interface, 1141
- StaticInitMerger, 506
  - extends 183, 506
  - Class, 506
- String, 63
  - Class, 63
- StringReader, 71
  - extends 509, 72
- StringReader, class in 825, 72
- StringSeq, 1141
  - extends 571, 1141
  - implements 799, 1141
  - Class, 1141
- Symbol, 1143
  - extends 509, 1143
  - implements 1723, 1143
  - implements 800, 1143
  - implements 861, 1143
  - implements 1723, 1143
  - Class, 1143
- Symbol.equals, 69
- Symbol.intern, 67
- Symbol.intern(2), 68
- Symbol.toString, 68
- Syntax Macro Table, 72, 83
- SyntaxQuoteReader, 71
  - extends 509, 76
- SyntaxQuoteReader, class in 825, 76
- TableSwitchGenerator, 508
  - Interface, 508
- tangle.c, i, 1733
- toString, method in 1143, 68
- TransactionalHashMap, 1145
  - extends 1723, 1145
  - implements 1723, 1145
  - Class, 1145
- Type, 394

Class, 394

- UnmatchedDelimiterReader, 71
  - extends 509, 80
- UnmatchedDelimiterReader, class in 825, 80
- UnquoteReader, 71
  - extends 509, 79
- UnquoteReader, class in 825, 79
- UnreadableReader, 84
  - extends 509, 89
- UnreadableReader, class in 825, 89
- URLClassLoader, 1723
  - extended by 769, 769
  - Class, 1723
- Util, 1149
  - Class, 1149
- ValIterator
  - implements 1723, 104
- ValIterator, class in 1000, 104
- Var, 1152
  - extends 553, 1152
  - implements 774, 1152
  - implements 805, 1152
  - implements 1140, 1152
  - Class, 1152
- VarReader, 83–85
  - extends 509, 85
- VarReader, class in 825, 85
- VectorReader, 71
  - extends 509, 80
- VectorReader, class in 825, 80
- WrappingReader, 71
  - extends 509, 74
- WrappingReader, class in 825, 74
- XMLHandler, 1164
  - extends 1723, 1164
  - Class, 1164