

```
#(1 2 3 4) collect: [ :each | each * 2 ]   →  #( 2 4 6 8 )
#(1 2 3 4)
  inject: 0
into: [ :each :result | each + result ]   →  10
```

```
"testing "
#( 2 4 ) anySatisfy: [ :each | each odd ]   →  false
#( 2 4 ) allSatisfy: [ :each | each even ]   →  true
```

```
"finding "
'abcdef' includes: $e   →  true
'abcdef' contains: [ :each | each isUppercase ]   →  false
'abcdef'
  detect: [ :each | each isVowel ]
  ifNone: [ $u ]   →  $a
```

```
"String – a collection of characters "
string := 'abc'.
string := string , 'DEF'   →  'abcDEF'
string beginsWith: 'abc'   →  true
string endsWith: 'abc'   →  false
string includesSubString: 'cD'   →  true
string asLowercase   →  'abcdef'
string asUppercase   →  'ABCDEF'
```

```
"OrderedCollection – an ordered collection of objects "
ordered := OrderedCollection new.
ordered addLast: 'world'.
ordered addFirst: 'hello'.
ordered size   →  2
ordered at: 2   →  'world'
ordered removeLast   →  'world'
ordered removeFirst   →  'hello'
ordered isEmpty   →  true
```

```
"Set – an unordered collection of objects without duplicates "
set := Set new.
set add 'hello'; add: 'hello'.
set size   →  1
```

```
"Bag – an unordered collection of objects with duplicates "
bag := Bag new.
bag add: 'this'; add: 'that'; add: 'that'.
bag occurrencesOf: 'that'   →  2
bag remove: 'that'.
bag occurrencesOf: 'that'   →  1
```

```
"Dictionary – associates unique keys with objects "
dictionary := Dictionary new.
dictionary at: 'smalltalk' put: 80.
dictionary at: 'smalltalk'   →  80
dictionary at: 'squeak' ifAbsent: [ 82 ]   →  82
dictionary removeKey: 'smalltalk'.
dictionary isEmpty   →  true
```

Streams

"ReadStream – to read a sequence of objects from a collection"

```
stream := 'Hello World' readStream.
stream next   →  $H
stream upTo: $o   →  'ell'
stream skip: 2.
stream peek   →  $o
stream upToEnd   →  'orld'
```

"WriteStream – to write a sequence of objects to a collection"

```
stream := WriteStream on: Array new.
stream nextPut: 'Hello'.
stream nextPutAll: #( 1 2 3 ).
stream contents   →  #( 'Hello' 1 2 3 )
```

File Streams

```
fileStream := FileDirectory default newFileNamed: 'tmp.txt'.
fileStream nextPutAll: 'my cool stuff'.
fileStream close.
```

```
fileStream := FileDirectory default oldFileNamed: 'tmp.txt'.
fileStream contents   →  'my cool stuff'
```

Method Definition

```
messageSelectorAndArgumentNames
  "comment stating purpose of message"
  | temporary variable names |
  statements
```

Class Definition

```
Object subclass: #NameOfSubclass
  instanceVariableNames: 'instVar1 instVar2'
  classVariableNames: "
  poolDictionaries: "
  category: 'Category-Name'
```

References

1. Andrew Black, Stéphane Ducasse, Oscar Nierstrasz and Damien Pollet, *Squeak by Example*, Square Bracket Associates, 2007, squeakbyexample.org.
2. Chris Rathman, *Terse guide to Squeak*, wiki.squeak.org/squeak/5699.
3. *Smalltalk*, Wikipedia, the free encyclopedia, en.wikipedia.org/wiki/Smalltalk.

Smalltalk Cheat Sheet

Software Composition Group
University of Bern

May 21, 2008

1. The Environment

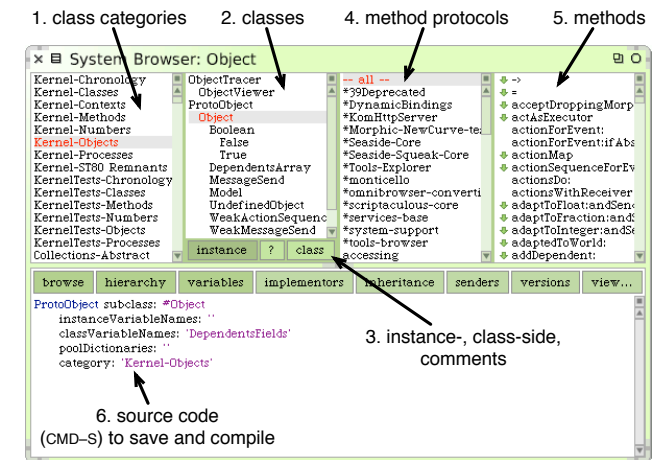


Figure 1: The Smalltalk Code Browser

- Do it (CMD-D): Evaluate selected code.
- Print it (CMD-P): Display the result of evaluating selected code.
- Debug it: Evaluate selected code step-by-step with the integrated debugger.
- Inspect it (CMD-I): Show an *object inspector* on the result of evaluating selected code.
- Explore it (CMD-SHIFT-I): Show an *object explorer* on the result of evaluating selected code.

2. The Language

- Everything is an object.
- Everything happens by sending messages.
- Single inheritance.
- Methods are public.
- Instance variables are private to objects.

Keywords

- self, the receiver.
- super, the receiver, method lookup starts in superclass.
- nil, the unique instance of the class UndefinedObject.
- true, the unique instance of the class True.
- false, the unique instance of the class False.
- thisContext, the current execution context.

Literals

- Integer
123
2r1111011 (123 in binary)
16r7B (123 in hexadecimal)
- Float
123.4
1.23e-4
- Character
\$a
- String
'abc'
- Symbol
#abc
- Array
#(123 123.4 \$a 'abc' #abc)

Message Sends

1. *Unary messages* take no argument.
1 factorial sends the message factorial to the object 1.
2. *Binary messages* take exactly one argument.
3 + 4 sends message + with argument 4 to the object 3.
#answer -> 42 sends -> with argument 42 to #answer.
Binary selectors are built from one or more of the characters +, -, *, =, <, >, ...

3. *Keyword messages* take one or more arguments.
2 raisedTo: 6 modulo: 10 sends the message named raisedTo:modulo: with arguments 6 and 10 to the object 2.

Unary messages are sent first, then binary messages and finally keyword messages:

2 raisedTo: 1 + 3 factorial → 128

Messages are sent left to right. Use parentheses to change the order:

1 + 2 * 3 → 9
1 + (2 * 3) → 7

Syntax

- Comments
"Comments are enclosed in double quotes"
- Temporary Variables
| var |
| var1 var2 |
- Assignment
var := aStatement
var1 := var2 := aStatement
- Statements
aStatement1. aStatement2
aStatement1. aStatement2. aStatement3
- Messages
receiver message (unary message)
receiver + argument (binary message)
receiver message: argument (keyword message)
receiver message: argument1 with: argument2
- Cascade
receiver message1; message2
receiver message1; message2: arg2; message3: arg3
- Blocks
[aStatement1. aStatement2]
[:argument1 | aStatement1. aStatement2]
[:argument1 :argument2 || temp1 temp2 | aStatement1]
- Return Statement
^ aStatement

3. Standard Classes

Logical expressions

true not → false
1 = 2 or: [2 = 1] → false
1 < 2 and: [2 > 1] → true

Conditionals

1 = 2 ifTrue: [Transcript show: '1 is equal to 2'].
1 = 2 ifFalse: [Transcript show: '1 isn't equal to 2'].

100 factorial / 99 factorial = 100
ifTrue: [Transcript show: 'condition evaluated to true']
ifFalse: [Beeper beep].

Loops

" conditional iteration "
[Sensor anyButtonPressed]
whileFalse: ["wait"].

pen := Pen newOnForm: Display.
pen place: Sensor cursorPoint.
[Sensor anyButtonPressed]
whileTrue: [pen goto: Sensor cursorPoint].

" fixed iteration "

180 timesRepeat: [
pen turn: 88.
pen go: 250].

1 to: 100 do: [:index |
pen go: index * 4.
bic turn: 89].

" infinite loop (press CMD+. to break) "
[pen goto: Sensor cursorPoint] repeat.

Blocks (anonymous functions)

" evaluation "
[1 + 2] value → 3
[:x | x + 2] value: 1 → 3
[:x :y | x + y] value: 1 value: 2 → 3

" processes "

[(Delay forDuration: 5 seconds) wait.
Transcript show: 'done'] fork → aProcess

Collections

" iterating "
'abc' do: [:each | Transcript show: each].
'abc'
do: [:each | Transcript show: each]
separatedBy: [Transcript cr].

" transforming "

#(1 2 3 4) select: [:each | each even] → #(2 4)
#(1 2 3 4) reject: [:each | each = 2] → #(1 3 4)