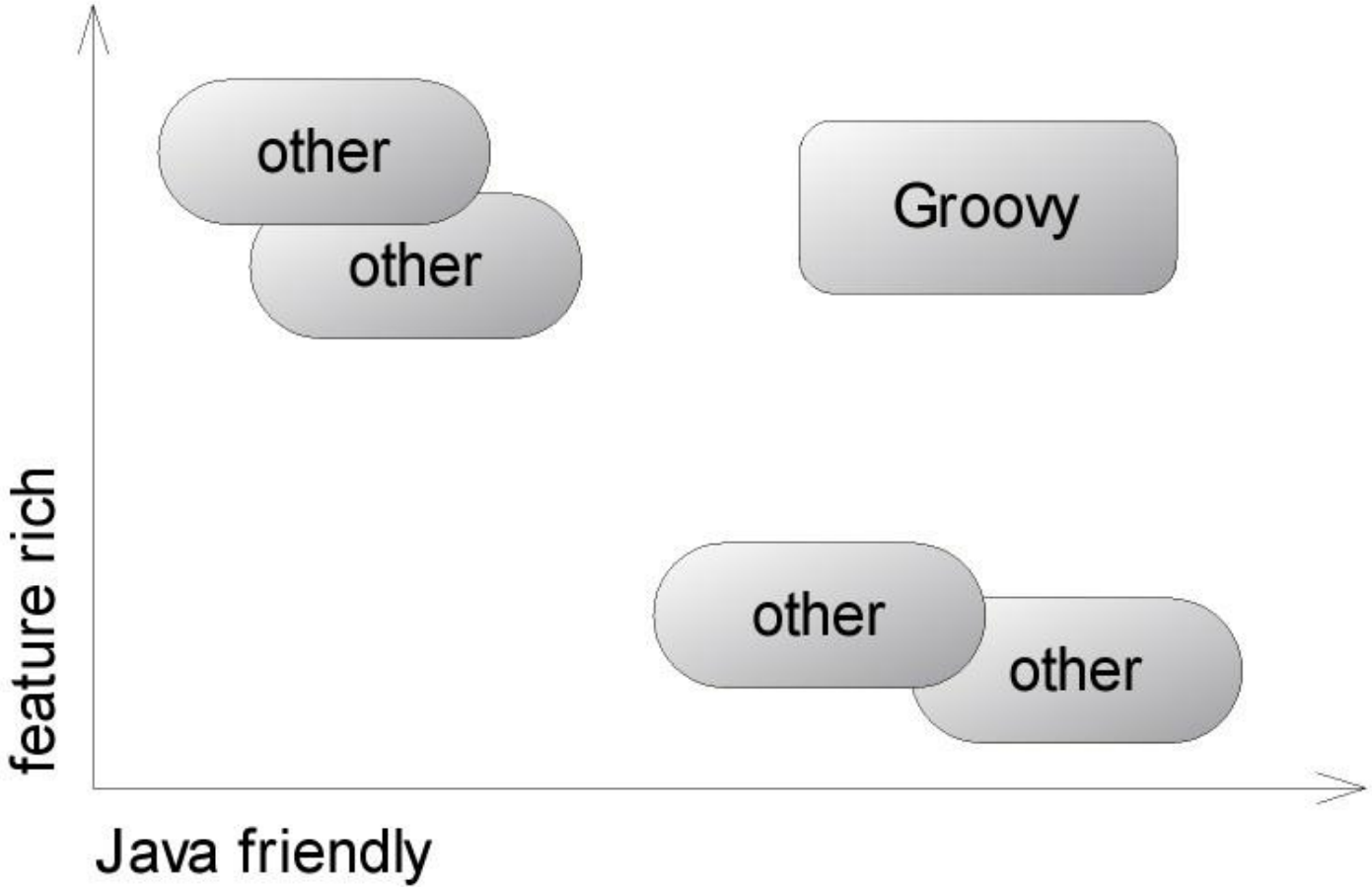

Groovy

Groovy History

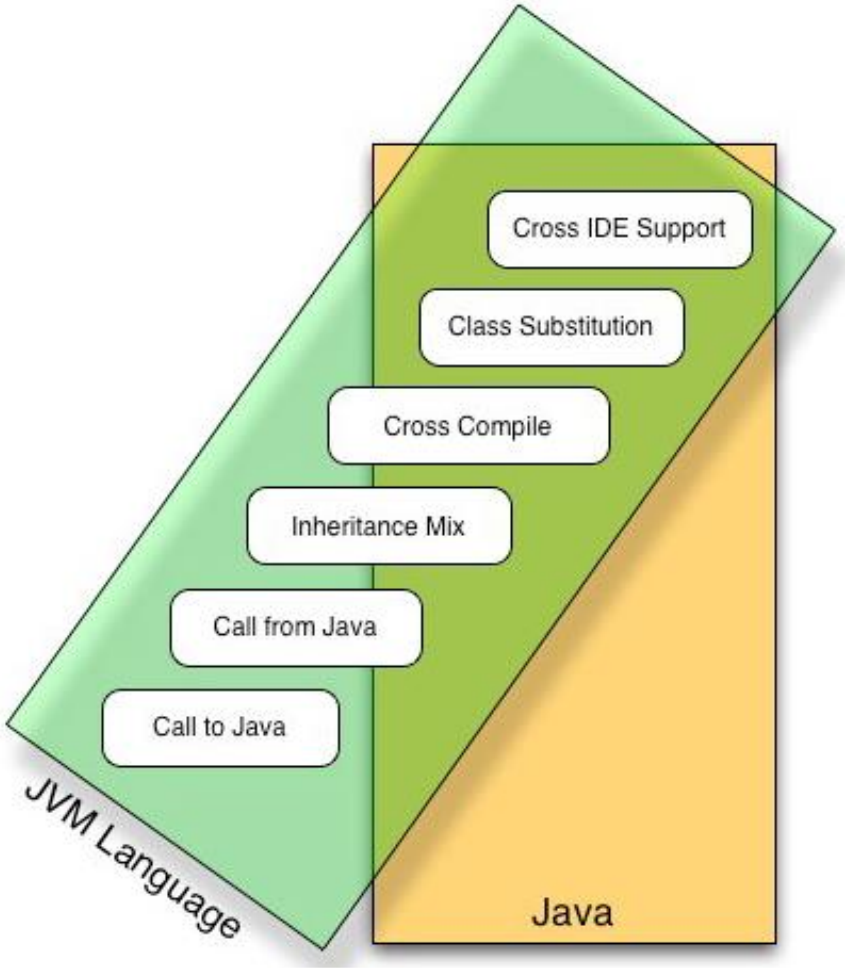
- James Strachan started with groovy at 2003
- Groovy 1.0 – 2007
- Groovy 2.0 – 2012
- Groovy 3.0 – already exists snapshot
- Today – 2.5

Groovy in JVM ecosystem



Java & Groovy – brothers in arms

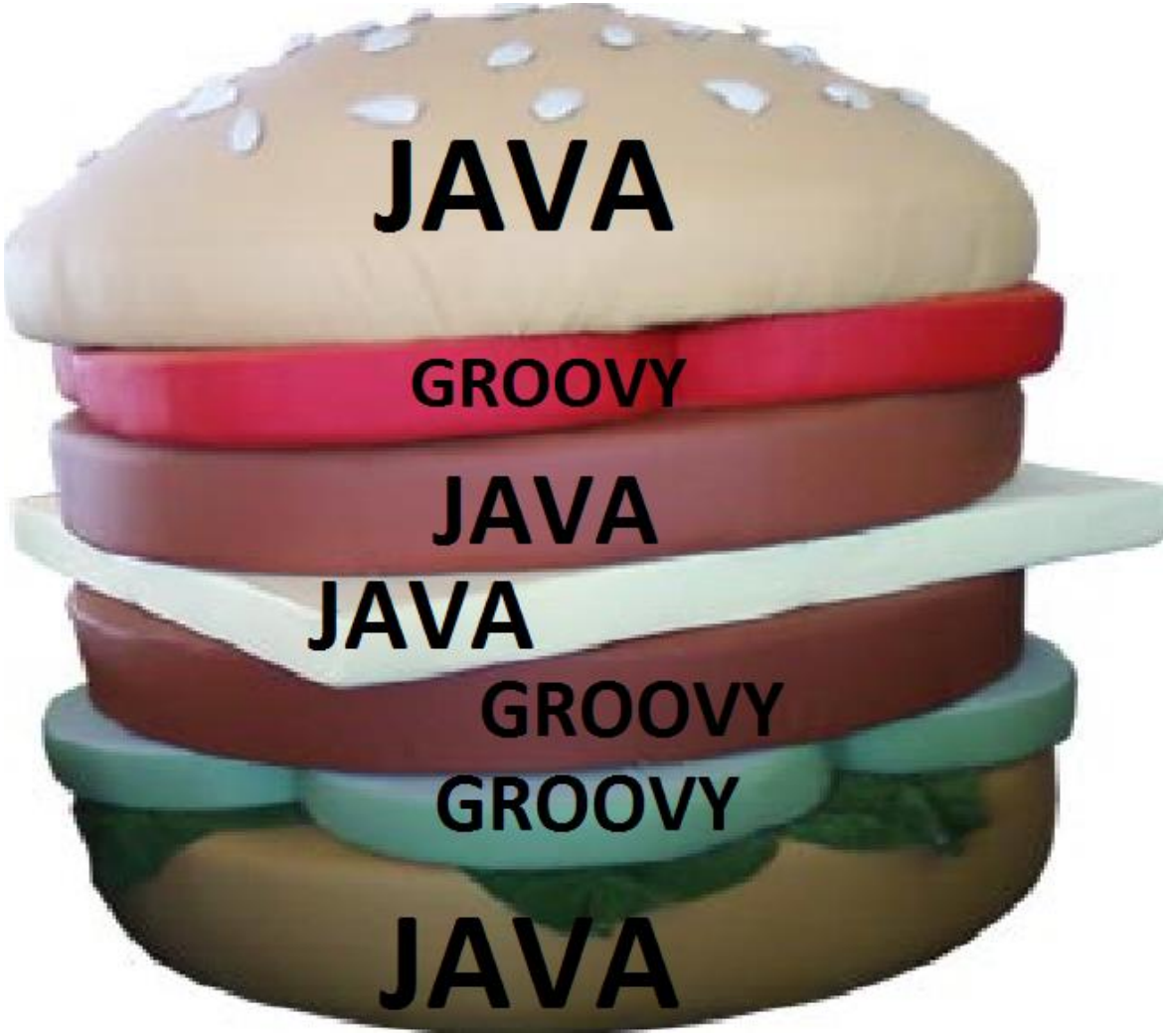
Interraction with Java



Groovy

- Java-like syntax (you can write groovy almost with java syntax)
- Groovy took the best from Python, Ruby, Smalltalk, Java Script and C#
- Compiled to java bytecode
- Can be used in any java project
- Can be used as script language

You can mix not only Vodka and Beer



Groovy is laconic

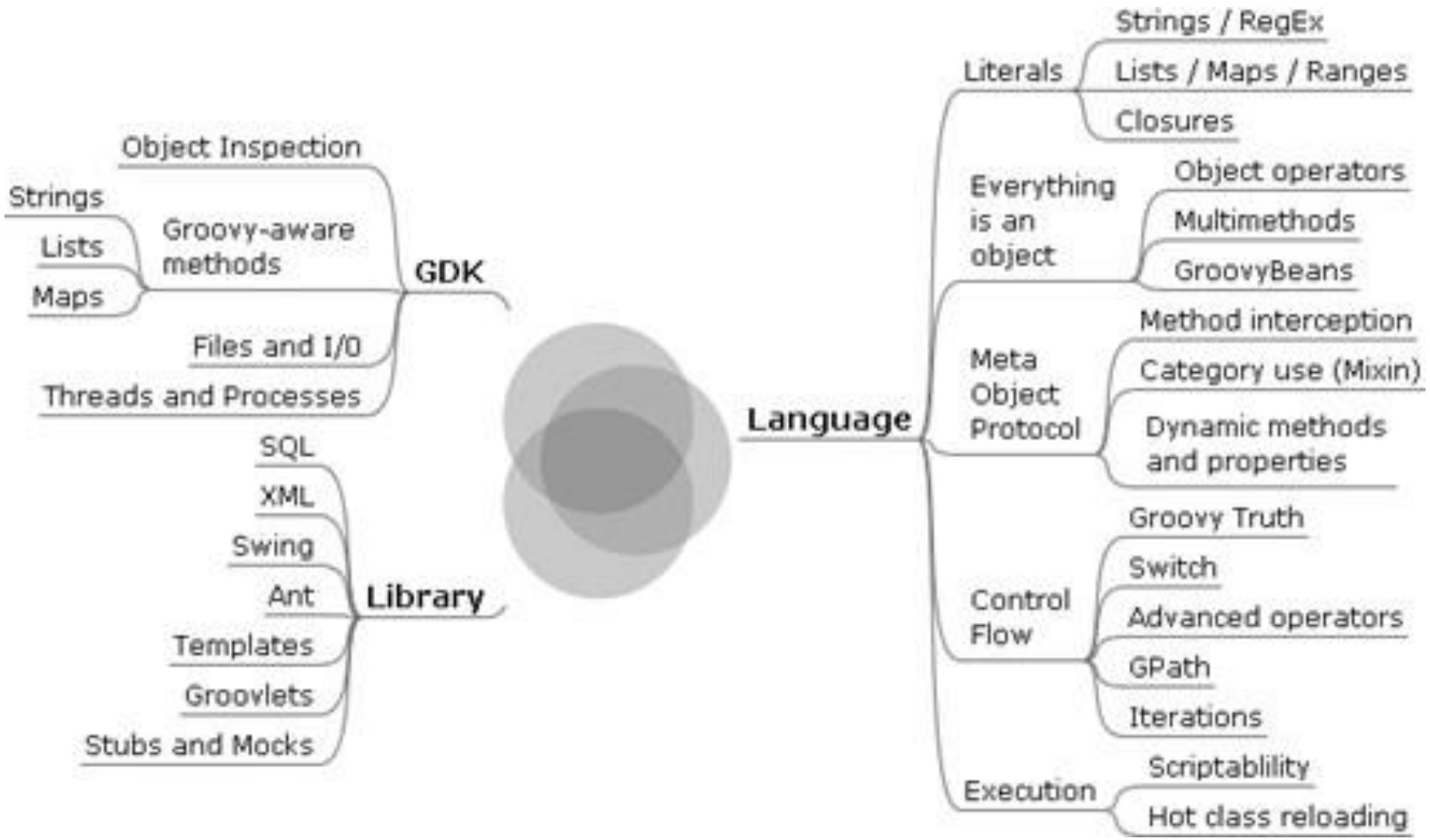
- today = new Date() // Groovy

- import java.util.*;
- Date today = new Date(); // Java

- require 'date'
- today = Date.new # Ruby

- import java.util._
- var today = new Date // Scala

Very powerful



Tasting: Closures and IO

```
def number = 0
new File('data.txt').eachLine { line ->
    number++
    println "$number: $line"
}
```

Output:

```
    1: first line
    2: second line
```

Tasting: Collection and properties

```
def classes = [String, List, File]
for (clazz in classes) {
     println clazz.package.name
}
```

Output:

```
     java.lang
     java.util
     java.io
```

Or even:

```
println( [String, List, File]*.package*.name )
```

Tasting: even XML is simple

XML: content:

```
<?xml version="1.0" ?>
<customers>
	<corporate>
		<customer name="Bill Gates" company="Microsoft" />
		<customer name="Tim Cook" company="Apple" />
		<customer name="Larry Ellison" company="Oracle" />
	</corporate>
	<consumer>
		<customer name="John Doe" />
		<customer name="Jane Doe" />
	</consumer>
</customers>
```

Tasting: even XML is simple

```
def customers = new XmlSlurper().parse(new
     File('customers.xml'))
for (customer in customers.corporate.customer) {
     println "${customer.@name} works for
     ${customer.@company}"
}
```

Output:

```
Bill Gates works for Microsoft
Steve Jobs works for Apple
Jonathan Schwartz works for Sun
```

Lab01

Hello World

Basics

Code style

Writing some groovy

Some dynamism

Differences from Java

- Not supported:
  - Anonymous inner classes (no need)
  - Multiple parameters Initialization in for
  - Initializing arrays on declaration (different syntax)
  - Java 8 functional style will be supported since groovy 3
  - Checked exceptions
  - Try with resources (No need)

Throwing out all unnecessary

Java:

```
java.net.URLEncoder.encode("a b", "UTF-8");
```

Groovy:

```
URLEncoder.encode 'a b' , 'UTF-8'
```

Default Imports

- groovy.lang.*
- groovy.util.*
- java.lang.*
- java.util.*
- java.net.*
- java.io.*
- java.math.BigInteger
- java.math.BigDecimal

Optional

- return
- Type declaration
- Casting
- throws

Improvement of existing functionality

assert:

```
def a = 5
def b = 9
assert b == a + a
```

output:

```
Assertion failed:
assert b == a + a
       |  | | | |
       9  | 5 | 5
          |   10
        false
at snippet22_failing_assert.run(snippet22_failing_assert.groovy:3)
```

Classes

```
class Course{
    private String title
     Course(String theTitle) {
        title = theTitle
    }
    String getTitle(){
        return title
    }
}
```

Scripts

```
Course groovy = new Course('Groovy')

assert groovy.getTitle() == 'Groovy'
assert getTitleBackwards(course) == 'yvoorG'

String getTitleBackwards(course) {
    String title = course.getTitle()
    return title.reverse()
}
```

GroovyBean, class inside script

```
class CourseBean {
    String title
}

def groovyCourse = new CourseBean()
groovyCourse.setTitle('Groovy')
assert groovyCourse.getTitle() == 'Groovy'
groovyCourse.title = 'Groovy rulez'
assert groovyCourse.title == 'Groovy rulez'
```

Annotations

```
import groovy.transform.Immutable
@Immutable class FixedCourse {
        String title
}
def groovy = new FixedCourse('Groovy')
def grooovy = new FixedCourse(title:'Groovy')
assert groovy.title == 'Groovy'
assert groovy == grooovy

try {
        groovy.title = "Oops!"
        assert false, "should not reach here"
} catch (ReadOnlyPropertyException expected) {
        println "Expected Error: '$expected.message'"
}
```

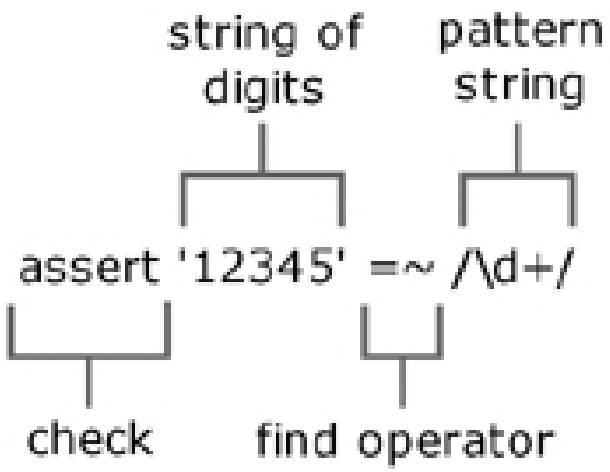
Grapes: Dependency manager for poor

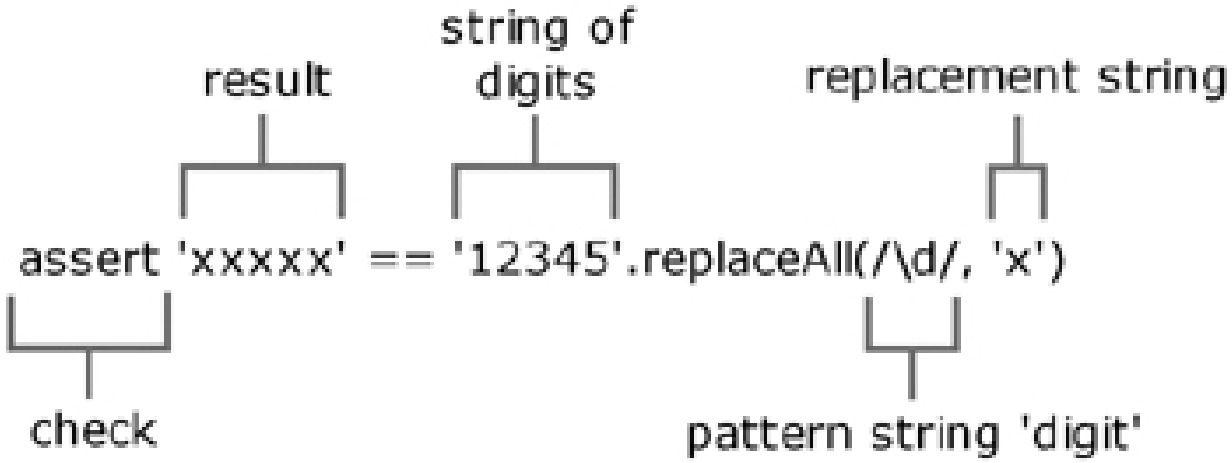
```
@Grab('commons-lang:commons-lang:2.4')
import org.apache.commons.lang.ClassUtils

class Outer {
    class Inner {}
}

assert !ClassUtils.isInnerClass(Outer)
assert ClassUtils.isInnerClass(Outer.Inner)
```

GString

```
def acronym = 'Gr8'
def fullWord = 'Great'

assert "$acronym stands for $fullWord"
== 'Gr8 stands for Great'
```

How I … and started to Love the Regular expressions



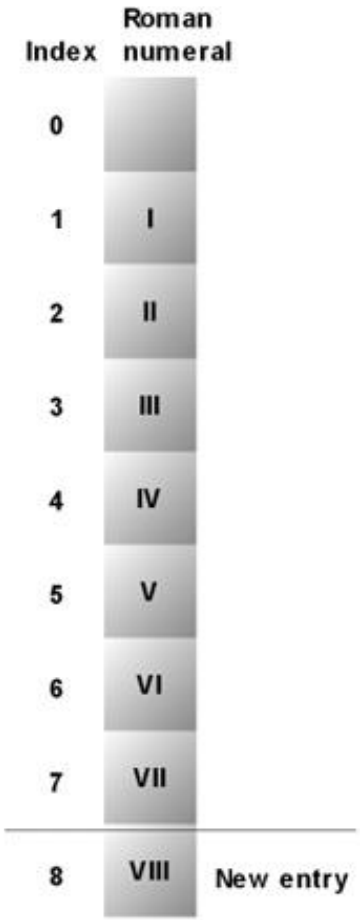


Really object oriented

```
def x = 1
def y = 2
assert x + y == 3
assert x.plus(y) == 3
assert x instanceof Integer
```

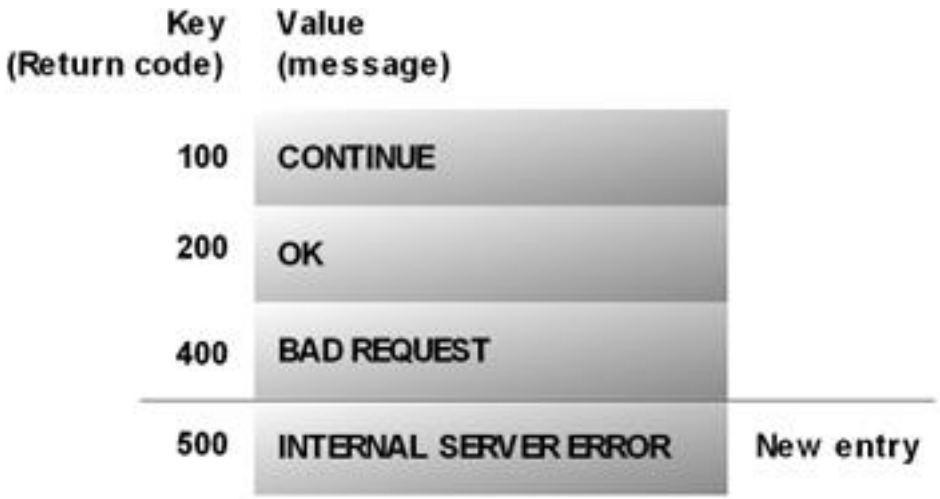
List

```
def roman = ['', 'I', 'II', 'III',
'IV', 'V', 'VI', 'VII']

assert roman[4] == 'IV'

roman[8] = 'VIII'

assert roman.size()
```



Maps

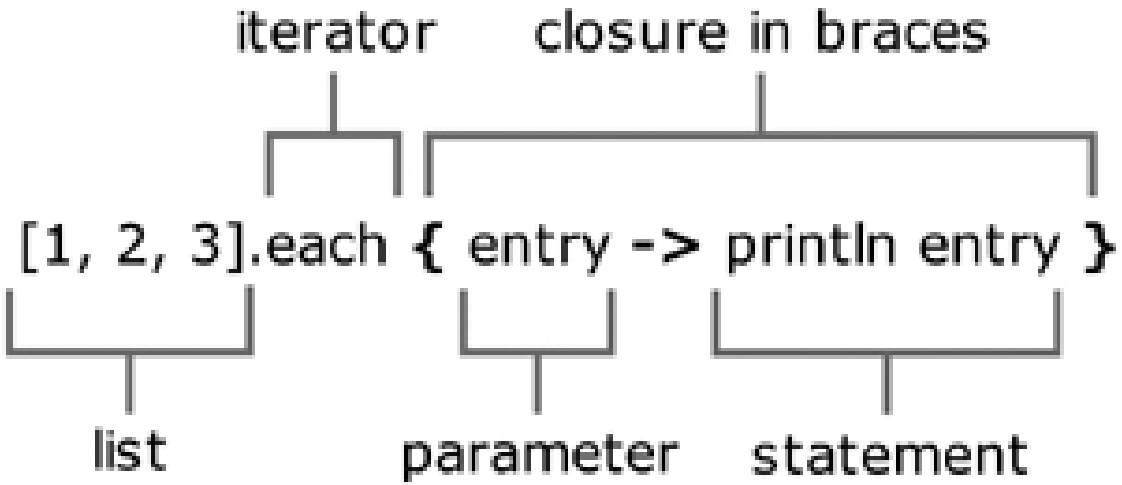
```
def http = [
100 : 'CONTINUE',
200 : 'OK',
400 : 'BAD REQUEST'
]

assert http[200] == 'OK'

http[500] = 'INTERNAL SERVER ERROR'

assert http.size() == 4
```



Ranges

```
def x = 1..10
assert x.contains(5)
assert !x.contains(15)
assert x.size() == 10
assert x.from == 1
assert x.to == 10
assert x.reverse() == 10..1
```

closures


iterator
closure in braces
[1, 2, 3].each { entry -> println entry }
list
parameter
statement

Why closures?

- Ability to pass a pointer to the code
  - Callbacks, handlers, listeners
- The ability to access any external variables, not only final
- It is beautiful!

Flow Control – more simple than Java

```
if (false) assert false
if (null) {
        assert false
} else {
        assert true
}

def clinks = 0
for (remainingGuests in 0..9) {
        clinks += remainingGuests
}
assert clinks == (10*9)/2
```

```
def list = [0, 1, 2, 3]
for (j in list) {
        assert j == list[j]
}

list.each() { item ->
        assert item == list[item]
}

switch(3) {
        case 1 : assert false; break
        case 3 : assert true; break
        default: assert false
}
```

Groovy truth

Groovy Truth

- With Java, a condition must be of a Boolean type.
- With Groovy:

Expression type	Outcome
Collections and maps	False is empty
Strings	False is empty
Numbers	False is zero
Objects	False is null

Task

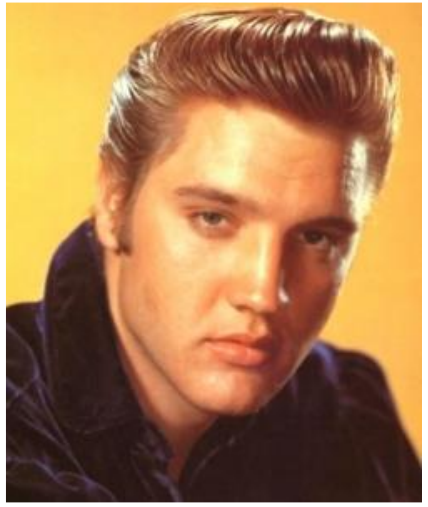
- What will happen if you will write the following:
  Person person = new Person(age: 200)
  if(person) println 'YES‘
- Make the same code print YES, only if age of person is under 120
- Wanna hint? - > Override asBoolean

Elvis Operator

Groovy provides the Elvis operator for easily returning a default value if an expression is false.

?:

```
String x;
String str = x ?: 'default value'
print str
```



Switch

- Switch is anti pattern in Java, but not in Groovy
- You can make switch on: primitives, enums, strings, regex or any object from class which implements isCase() method

```
switch (x) {
    case 1..10 :print 'between 1 and 10'
        break
    case ~/\d*/:print 'digits!!!'
        break
    case ['java','groovy','scala']: print 'it is a language'
        break
    default: print 'strange...'
}
```

Is Case already implemented in

- Object a.equals(b)
- Class a.isInstance(b)
- Collection a.contains(b)
- Range a.contains(b)
- Pattern a.matcher(b.toString()).matches()
- String (a==null && b==null) || a.equals(b) Closure a.call(b)

More switch

```
def x = new Employee()
switch (x){
    case Employee : println 'this is employee'
        break
    case Customer : println 'this is customer'
        break
    default:println'other...'
}
```

Power of switch –how can it work?
You need a hint? Implement isCase method

```
switch (x) {
    case Dog: println 'you talking like dog'
        break
    case Employee: println 'your salary is not good'
        break
    default:println 'I dont know who you are'
}
```

- You need a hint? Implement isCase method

While & for

- While works like in Java(except from condition, remember asBoolean?)
- For:
- **for (**variable **in** iterable**) {** body **}**
- Works on everything
- All iterable methods come from Object

Iterable implementations

No.	Candidate	Use with
1	java.util.Iterator	Itself
2	org.w3c.dom.NodeList	Iterator over Nodes
3	java.util.Enumeration	Convert to iterator
4	java.util.regex.Matcher	Iterator over matches
5	java.lang.Iterable	Iterator.iterator()
6	Responds to iterator method	Call it
7	Collectable	Collection.iterator()
8	java.util.Map	Iterator over Map.Entry objects
9	Array	Iterator over array items
10	MethodClosure	Iterator over calls
11	java.lang.String	Iterator over characters
12	java.io.File	Iterator over lines
13	null	Empty iterator
13	Otherwise	Iterator that only contains the candidate

Fixing Java

Тип	Java	Groovy
Array	length	size()
Array	java.lang.reflect.Array.getLength(array)	size()
String	length()	size()
StringBuffer	length()	size()
Collection	size()	size()
Map	size()	size()
File	length()	size()
Matcher	groupCount()	size()

Lab

- write class Human wich can have several names like in real life.
- Class Human will have property names, which will be list of String
- Write method addName which will take a String and will add it to the list of names
- Write method getRandomName which will return random name from the list.
- "groovy true" should return true in case Human has at least one name **(hint implement asBoolean in Human class)**
- Switch on a human should work like this:

```
def therion = new Human(names=['Therion','Beast','Dwarf','halfman']
def john = new Human(names=['John','John Stark','John Snow']
switch(someHuman){
        case john: println('lanister allways pay debts')          break;
        case therion: println('starks forever')...}     (hint implement isCase method)
```

- Bonus: Make it possible to iterate on the names like this:
  human.each{println(it)}

output: Therion Beast Dwarf halfman **hint(implement method: iterator()**

Exceptions

- Almost like in Java
- Compiler doesn’t check for checked exceptions
  - All exceptions are Runtime
- Type of exception inside catch block is optional
  - Duck typing rules

Groovy Beans

Improved JavaBeans

- Unlike Java, they have REAL properties
- Fields without access modifiers
- class Person {
  String firstName
  String lastName
  }
- Getters and setters will be generated in bytecode
- If you will declare getter/setter they will not be generated
- Yes, like default java constructor
- When you apply to the field you will apply to getter/setter

Improved JavaBeans

```
println person.firstName
person.lastName= 'Doe'
```

- extreme cases:
- If you want directly apply to the field:person.@lastName
- If you don’t need a setter (read-only)
  Write a setter which throws exception

Improved JavaBeans

- Groovy also will generated mapped constructor
- Yes, like in C#
- new Person(firstName: 'john', lastName: 'smith')
- new Person(firstName: 'jane')

Just compare

- JAVA:
  createConnection(**'http'**, **'localhost'**, 8080, **'artifactory'**, 600, 6000)
- GROOVY:
  createConnection(protocol: **'http'**, host: **'localhost'**, port: 8080, uri: **'artifactory'**, connectionTimeout: 600, socketTimeout: 6000)

Where dynamism comes from?

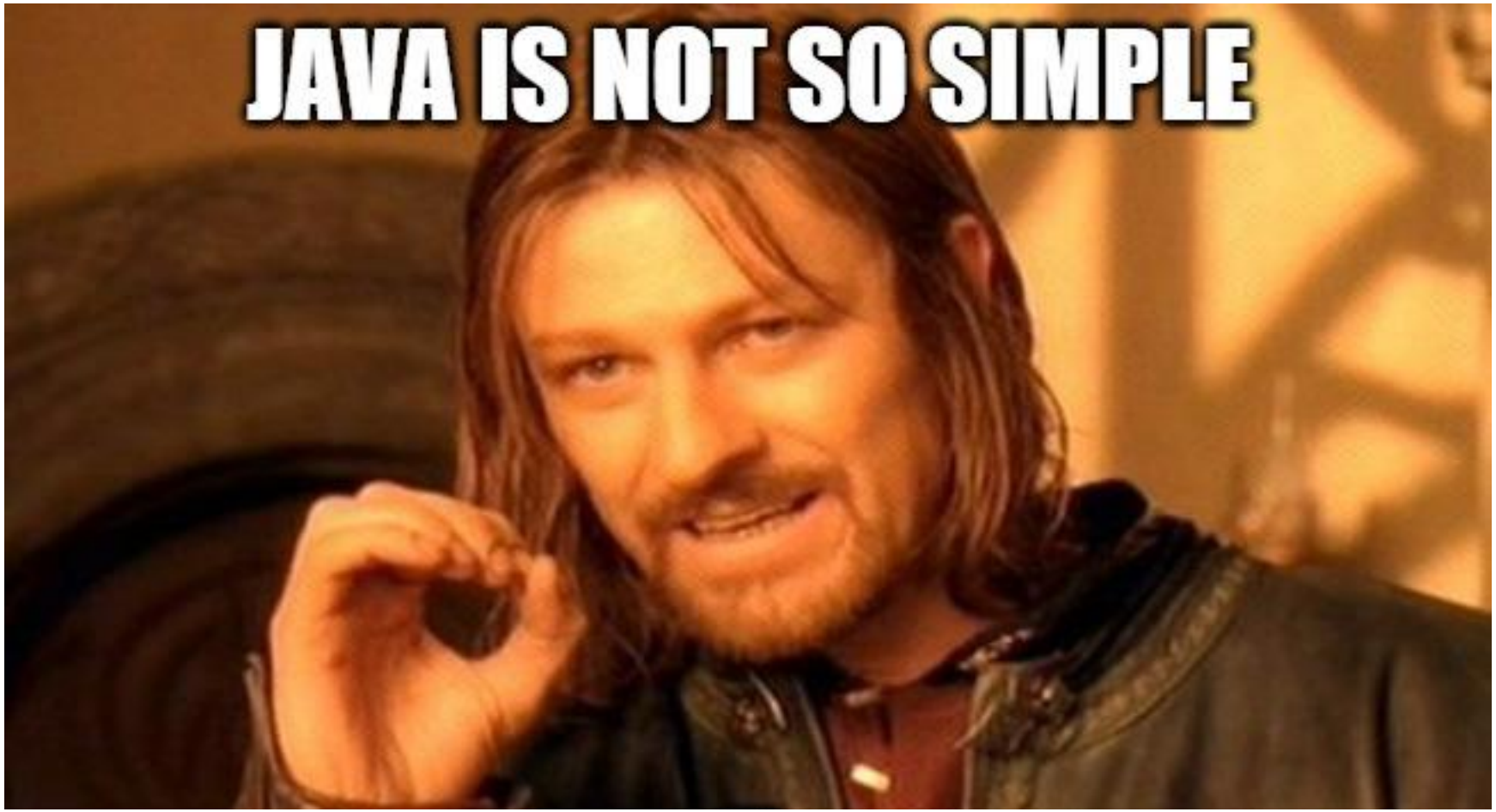
- If Groovy compiled to .class, and Java is static language?
- Answer: Runtime Proxy (it called MetaClass)
  - MetaClass intercepts all method invocation
  - Can be disabled by @CompileStatic
- @TypeChecked – disable def variables

Primitive types

Typing in Groovy

Operators are methods

Strings, regular expressions and numbers



Operators work only on primitive

Methods work only on objects

Groovy Solution

1. All types are objects
2. All operators are methods

```
(60 * 60 * 24 * 365).toString(); // invalid Java

int secondsPerYear = 60 * 60 * 24 * 365;

secondsPerYear.toString(); // invalid Java

new Integer(secondsPerYear).toString();

assert "abc" - "a" == "bc" // invalid Java
```

Types are optional

- Rule of a thumb: if you know the variable type –declare it!
- When you will not declare types: Slurpers, Builders and other magic
- @TypeChecked–if you want to disable optional typing

Casting in Groovy

- You can use regular Java syntax:
- Dog dog–(Dog) (AnimalFactory.getAnimal())
- Or you can use Groovy syntax:
- Dog dog–AnimalFactory.getAnimal() as Dog

Duck typing

Don’t implement interface, just be it!

Duck typing

- Groove does not require a cast to type
- But your IDE will thank you

```
def nameAndAge() {
    [name: 'john', age:30]
}
Person p = nameAndAge() as Person
```

```
interface Duck {
    void swim()
}
```

Bolton doesn't implement **Duck** interface

```
class Bolton {
    void swim() {
        println "I'm swimming like wounded Chapaev"
    }
}
```

```
static void main(String[] args) {
    Duck duck = new Bolton()
    duck.swim()
}
```

Doesn't Work!!!

```
static void main(String[] args) {
    Duck duck = new Bolton() as Duck
    duck.swim()
}
```

Works!!!

Duck typing rules

- Works against interfaces, cast operator ‘as’ can be omitted
- Doesn’t work against classes, but in some cases still will work without a casting.
  - E.g. you have list of ducks and you add a boat (in case of casting it will fail, in case generic type of list is explicitly defined or resolved (you instantiate this list with Ducks only) it also will fail because groovy will try implicitly cast Boat to Duck even without ‘as’ operator) but if this list of ducks declared as list of objects, you can add Boat to this list and later if you will try to call swim method on each element it will work.

Maps instead of class

```
Map map = [:]
map.firstName = 'Joe'
map.lastName = 'Doe'
map.sayHello = {
    println "Hello, my name is $map.firstName $map.lastName"
}
Person person = map as Person
person.sayHello()
```

Operators overloading

- I know what you think: “no-no-no, never and no way!”

  #define TRUE FALSE //enjoy debugging
- But what if operators are just method names?
  - More correct, alias for methods
  - Yes you can shoot to your leg, but how it is differs from method overriding?
- 1 + 2 is just 1.plus(2)
- Numbers are objects, remember?

Simple task

- Write Person class with age and name properties
- Create 2 persons with the same state and compare them with ==
- Override equals on person
- Run the same test, which compares 2 persons

So how do I compare references?

is

```
person.is(person2)
```

Operators overloading

Operator	Method	Already overridden for:
`a + b`	`a.plus(b)`	`Number`, `String`, `StringBuffer`, `Collection`, `Map`, `Date`, `Duration`
`a[b]`	`a.getAt(b)`	`Object`, `List`, `Map`, `CharSequence`, `Matcher`, more
`a << b`	`a.leftShift(b)`	`Append in many cases`

`switch(a){`
`  case b:`
`}`
`a in b`
	`b.isCase(a)`	`Object`, `Class`, `Range`, `Collection`, `Pattern`, `Closure`;
		Any class can be used in a switch

`a == b`	`a.equals(b)`	In groovy == is an equals
`a <=> b`	`a.compareTo(b)`	`java.lang.Comparable`
`a as type`	`a.asType(typeClass)`	everywhere

More operators overloading

- a –b is: a.minus(b)
- a * b is: a.multiply(b)
- a** b is: a.power(b)
- a / b is: a.div(b)
- a % b is: a.mod(b)

More operators overloading

- a | b is: a.or(b)
- a & b is: a.and(b)
- a ^ b is: a.xor(b)
- a++ or ++a is: a.next()
- a--or --a is: a.previous()

The last slide of operators, I promise

- ~a is: a.bitwiseNegate()
- -a is: a.negative()
- +a is: a.positive()
- a <=> is: a.compareTo(b)
- a > is: a.compareTo(b) > 0
- a >= is: a.compareTo(b) >= 0
- a <= is: a.compareTo(b) <= 0
- a < is: a.compareTo(b) < 0

לא בטוח – על תעכוף

Simple task

- You have a Map<Integer,Person>
- Each Person has property Address
- Each address has property name
- You need to print street name of person from the map
- You can’t be sure that this person exists in the map

Simple task

```
class PersonService{
    void printStreetName(Map<Person, Integer> map, int key) {
        //print safely name of the street of
        // print null instead of NPE
    }
}
```

Java style solution

```
void printStreetName(Map<Person, Integer> map, int i) {
    String printedValue = null
    if(map != null) {
        Person person = map.get(i)
        if (person != null) {
            Address address = person.getAddress()
            if (address != null) {
                String name = address.getName()
                if (name != null) {
                    printedValue = name
                }
            }
        }
    } println printedValue
}
```

Groovy style solution

- Println map?.get(i)?.address?.name

Lab02

Write class Money and support + operator which will work on money or numbers

- E.g: Money money1 = new Money(100)
- Money money2 = new Money(200)money1 = money1+money2money1 = money1 +100
- Bonus: with different currency according to the rate
- http://rate-exchange-1.appspot.com/currency?from=USD&to=ILS

Strings

Syntax	Example	Support expressions	No need to escape special chars	Multi line
single quotes	`‘Hello, World!’`	✗	✗	✗
double quotes	`“Hello, $name!”`	✓	✗	✗
three single quotes	`‘‘‘Hello, World!’’’`	✗	✗	✓
three double quotes	`“““Hello, $name!”””`	✓	✗	✓
slash	`/.*"(.*)".*\/(.*)\//`	✓	✓	✓
dollarsand slashed	`$/.*"(.*)".*/(.*)//$`	✓	✓	✓

```
String str = """You can
do that
in Groovy"""

String str2 = /James said:
“2 \\are shorter than 4\\”/
```

GString

- Use double quotes
- Can contain expression

```
println “quote is: $str"
println "diff between x and y = ${x-y}"
println “Trumpeldor said:
        ${new Trumpeldor().goodToDieForOurCountry()}"
```

Games with strings

```
String greeting = 'Hello Groovy!'
assert greeting.startsWith('Hello')
assert greeting.getAt(0) == 'H'
assert greeting[0] == 'H'
assert greeting.indexOf('Groovy') >= 0
assert greeting.contains('Groovy')
assert greeting[6..11] == 'Groovy'
assert 'Hi' + greeting - 'Hello' == 'Hi Groovy!'
assert greeting.count(‘o’) == 3
assert 'x'.padLeft(3) == '  x'
assert 'x'.padRight(3,'_') == 'x__'
assert 'x'.center(3) == ' x '
assert 'x' * 3 == 'xxx'
```

String is immutable, isn’t it?

```
def greeting = 'Hello'
greeting <<= ' Groovy'
assert greeting instanceof java.lang.StringBuffer
greeting << '!'
assert greeting.toString() == 'Hello Groovy!'
greeting[1..4] = 'i'
assert greeting.toString() == 'Hi Groovy!'
```

Regular expression support

- Developer had a problem, so he tried to solve it with regular expression.No he has two problems

Operator	meaning
=~	find
==~	match

- Do not forget about slashy strings

Games with numbers

```
assert 1 == (-1).abs()
assert 2 == 2.5.toInteger() // conversion
assert 2 == 2.5 as Integer // enforced coercion
assert 2 == (int) 2.5 // cast
assert 3 == 2.5f.round()
assert 3.142 == Math.PI.round(3)
assert 4 == 4.5f.trunc()
assert 2.718 == Math.E.trunc(3)
assert '2.718'.isNumber() // String methods
assert 5 == '5'.toInteger()
assert 5 == '5' as Integer
assert '6 times' == 6 + ' times' // Number + String
(int) '5' // what will happen?
```

Groovy is object oriented

- What can you pass to the method, as its args?
  - primitive, references to objects
  - What about some algorithm or method, function?

Callback method pattern

- Write method which will calculate duplicates of object in list
- public int countDuplicates(T, List<T>…)

Just implement equal method

Closures

Extremely important!

Dou you remember why did we needed in java anonymous inner classes

- To provide some logic, where it will be used by other code, without knowing something about it`JButton.addActionListener(ActionListener a)`
- Somewhere inside:
- `a.actionPerformed(e)`
- Closures are for that, but much more powerful and convenient
- `JButton.addActionListener(Closure c)`
- Somewhere inside:
- `c.call(e)`

So what is the difference?

```
JButton.addActionListener(new ActionListener{
   actionPerformed(Event e){
      //can’t do much except of playing with e
   }
});
```

- vs.

```
JButton.addActionListener { Event e ->
      //can access whatever I want from the caller
   }
});
```

It’s a voodoo!

- If the last argument of the method -closure, it can be taken out of the brackets
- And if there is no interface, how to know which arguments are passed to the closure?

Closure without parameters

```
void printCalculation(Closure closure){
                    println(closure()
        )}
printCalculation{1+1}
```

Step 1 to understand closures mechanism

```
Closure closure = { return 1 + 1 }

void printCalculation(Closure closure) {
    println(closure.call())
}

printCalculation(closure)
```

Step 2 to understand closures mechanism

```
Closure closure = { 1 + 1 }

void printCalculation(Closure closure) {
    println(closure())
}

printCalculation(closure)
```

Step 3 to understand closures mechanism

```
void printCalculation(Closure closure) {
    println(closure())
}

printCalculation({ -> 1 + 1 })
```

Step 4 to understand closures mechanism

```
void printCalculation(Closure closure) {
    println(closure())
}

printCalculation({ 1 + 1 })
```

Step 5 to understand closures mechanism

```
void printCalculation(Closure closure) {
    println(closure())
}

printCalculation { 1 + 1 }
```

Closure with parameters

```
double calcTaxes(List<Integer> salaries, Closure closure) {
    double totalTaxes=0
    for (int salary : salaries) {
        totalTaxes+=closure(salary)
    }
    totalTaxes
}
println calcTaxes(salaries){salary -> salary*0.18}
```

Step 1

```
calcTaxes(salaries, {
    int salary -> return salary * 0.18
}
);
```

Step 2

```
calcTaxes(salaries, {
    int salary -> salary * 0.18
}
);
```

Step 2

```
calcTaxes(salaries){salary -> salary * 0.18}
```

Step 2

```
calcTaxes(salaries){it * 0.18}
```

Closure in closure

```
double calcTaxes(List<Integer> salaries, Closure closure) {
    double totalTaxes = 0
    salaries.each { int salary -> totalTaxes += closure(salary) }
    totalTaxes
}
```

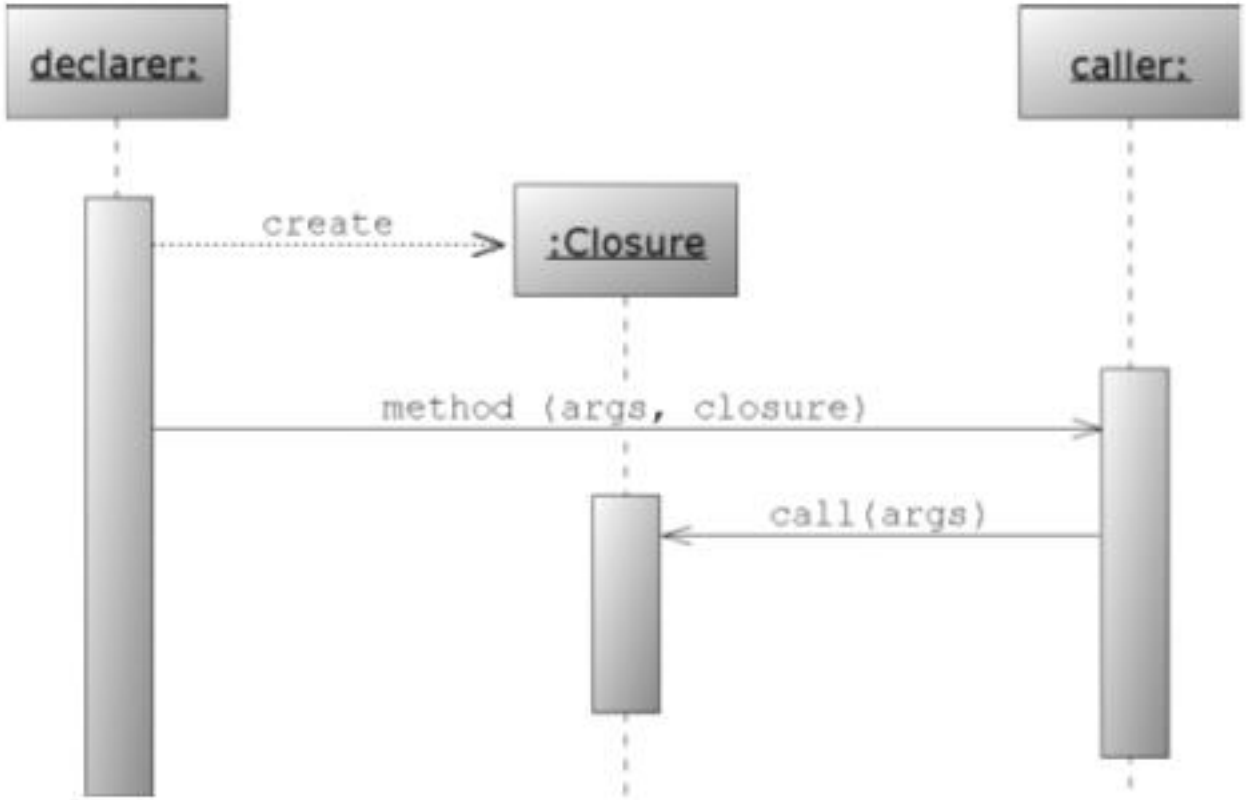
Closure in closure

```
double calcTaxes(List<Integer> salaries, Closure closure) {
    double totalTaxes = 0
    salaries.collect { int salary -> totalTaxes += closure(salary) }
    .sum() as Double
}
```

it closure parameter

```
def benchmark(int repeat, Closure worker) {
    def start = System.nanoTime()
    repeat.times { worker(it) }
    def stop = System.nanoTime()
    stop - start
}
def slow = benchmark(10000) { (int) it / 2 }
def fast = benchmark(10000) { it.intdiv(2) }
assert (fast * 5) < slow
```

The most important –what, who and when



Scopes

- Closure has access to variables of different scopes
  - Variables of object which created Closure
  - Variables of object which call Closure
  - Local variables of the method, where closure is used
- Owner
  - Creator of the closure
- Delegate
  - By default is closure caller, but can be changed

Closure is an object

- getParameterTypes()
- isCase() – any code

```
switch(10){
    case {it%2 == 1} : assert false
}
```

Remember the Java comparator

- Write a code which will sort Persons by age
- persons.sort{it.age}

Same closure with different parameters

- You need to sort list of persons by name and if the names are equals, then by age.
- How many lines of code?
- Should be only one:
- persons.sort({a,b-> a.name.compareTo(b.name) ?: a.age.compareTo(b.age)})

But wait!!!

- We used closure of sort method differently
- With one parameter (it) and with two parameters (a,b)
- How can it work? Magic? Voodoo?
- getParameterTypes()

Implement reduce methods which will work for both closures

```
def totalEmployee1 = reduce(employees,
      {Employee e1,Employee e2 -> new Employee(salary: e1.salary+e2.salary) })

def totalEmployee2 = reduce(employees,{ -> "salary" })
```

References to existing methods

- Closure myMethod= myObject.&someMethod
  - The same with the fields
- String name = myPerson.@firstName

Question

- What is the difference myPerson.firstNameand myPerson.@firstName?
- @firstName–is direct access to the field
- firstName–is access via setter/getter

Example

```
def benchmark(int repeat, Closure worker) {
    def start = System.nanoTime()
    repeat.times{ worker(it) }
    def stop = System.nanoTime()
    stop -start
}
```

Lab07

Write Equalator(like comparator, but for equality check)

But first write a tool which will use it.

Tool is a count method, which receive a list and Equalator, and calculate duplicates with help of Equalator.

Methods with closures for numbers

```
def store = ''
10.times{
     store += 'x'
}
assert store == 'xxxxxxxxxx'

store = ''
1.upto(5) { number ->
     store += number
}
assert store == '12345'
```

Methods with closures for numbers

```
def store = ''
2.downto(-2) { number ->
     store += number + ' '
}
assert store == '2 1 0 -1 -2 `

store = ''
0.step(0.5, 0.1 ){ number ->
     store += number + ' '
}
assert store == '0 0.1 0.2 0.3 0.4 '
```

Collections, ranges, lists, maps, arrays

Ranges –why?

- What this code does?

```
for (int i=0; i<upperBound; i++){
    // do something with i
}
```

- This is much more readable, isn’t it?

```
0..upperBound
```

Declaring Ranges

```
assert (0..10).contains(0)
assert (0..10).contains(5)
assert (0..10).contains(10)
assert (0..10).contains  (-1) == false
assert (0..10).contains(11) == false
assert (0..<10).contains(9)
assert (0..<10).contains(10) == false

def a = 0..10
assert a instanceof Range
assert a.contains(5)
```

Declaring Ranges

```
def today = new Date()
def yesterday = today - 1
assert (yesterday..today).size() == 2
assert ('a'..'c').contains('b')

def log = ''
for (element in 5..9){
     log += element
}
assert log == '56789‘
```

Declaring Ranges

```
log = ''
for (element in 9..5){
     log += element
}
assert log == '98765'

log = ''
(9..<5).each { element ->
     log += element
}
assert log == '9876'
```

Declaring Ranges

```
assert (0.0..1.0).contains(1.0)
assert (0.0..1.0).containsWithinBounds(0.5)
```

Range is an object, yep

```
def result = ''
(5..9).each { element ->
     result += element
}
assert result == '56789'
assert 5 in 0..10
assert (0..10).isCase(5)
assert (0..10).isCase(5.1)  // fails


def ages = [20, 36, 42, 56]
def midage = 21..50
assert ages.grep(midage) == [36, 42]
```

Range is an object, yep

```
def age = 36
switch(age){
     case 16..20 : insuranceRate = 0.05 ; break
     case 21..50 : insuranceRate = 0.06 ; break
     case 51..65 : insuranceRate = 0.07 ; break
     default: throw new IllegalArgumentException()
}
assert insuranceRate == 0.06
```

Using classes in ranges

- Such class should have next() and previous() methods
  - Duck typing, right!
- And implement Comparable (ranges are sorted)
- E.g.: Date. So you can:

```
Date backIn70s = new Date(0)
Date today = new Date()
Range almost45yearsRange = backIn70s..today
Date yesterday = today - 1
almost45yearsRange.containsWithinBounds(yesterday)
```

Lab04

- Write a script which will rank people according amount of money they have. You should use switch on Money objects
- Make Money be rangable
- Write Weekday class and make it “rangable”

Java is not simple…

- It is more comfortable to work with arrays
- Lists are dynamic

Lists

- Arrays comfort, lists dynamism

```
List myList = [1, 2, 3]
assert myList.size() == 3
assert myList[0] == 1
assert myList instanceof ArrayList
List emptyList = []
assert emptyList.size() == 0
```

Lists

```
List myList = [1, 2, 3]
List longList = (0..1000).toList()
assert longList[555] == 555
List explicitList = new ArrayList()
explicitList.addAll(myList)
assert explicitList.size() == 3
explicitList[0] = 10
assert explicitList[0] == 10
explicitList = new LinkedList(myList)
assert explicitList.size() == 3
explicitList[0] = 10
assert explicitList[0] == 10
```

More than just [0]

```
myList = ['a','b','c','d','e','f']
assert myList[0..2] == ['a','b','c']
assert myList[0,2,4] == ['a','c','e']
myList[0..2] = ['x','y','z']
assert myList == ['x','y','z','d','e','f']
myList[3..5] = []
assert myList == ['x','y','z']
myList[1..1] = [0, 1, 2]
assert myList == ['x', 0, 1, 2, 'z']
```

And even: `myList[-1]`

More interesting operators

```
myList = []
myList += 'a'
assert myList == ['a']
myList += ['b','c']
assert myList == ['a','b','c']
myList = []
myList << 'a' << 'b'
assert myList == ['a','b']
assert myList - ['b'] == ['a']
assert myList * 2 == ['a','b','a','b']
```

Lists and flow control

```
myList = ['a', 'b', 'c']
assert myList.isCase('a')
assert 'b' in myList
def candidate = 'c`
switch(candidate){
    case myList : assert true; break
    default : assert false
}
```

Lists and flow control

```
myList = []
if (myList) assert false
def expr = ''
for (i in [1,'*',5]){
    expr += i
}
assert expr == '1*5'
```

Manipulating with content

```
assert [1,[2,3]].flatten() == [1,2,3]
assert [1,2,3].intersect([4,3,1])== [3,1]
assert [1,2,3].disjoint([4,5,6])==[]
list = [1,2,3]
popped = list.pop()
assert popped == 3
assert list == [1,2]
assert [1,2].reverse() == [2,1]
assert [3,1,2].sort() == [1,2,3]
```

Manipulating with content

```
def list = [ [1,0], [0,1,2] ]
list = list.sort { a,b -> a[0] <=> b[0] }
assert list == [ [0,1,2], [1,0] ]
list = list.sort { item -> item.size() }
assert list == [ [1,0], [0,1,2] ]
list = ['a','b','c']
list.remove(2)
assert list == ['a','b']
list.remove('b')
assert list == ['a']
```

Manipulating with content

```
list = ['a','b','b','c']
list.removeAll(['b','c'])
assert list == ['a']
def doubled = [1,2,3].collect{ item ->
   item*2
}
assert doubled == [2,4,6]
def odd = [1,2,3].findAll{ item ->
     item % 2 == 1
}
assert odd == [1,3]
```

Getting the content

```
def list = [1, 2, 3]
assert list.first() == 1
assert list.head() == 1
assert list.tail() == [2, 3]
assert list.last() == 3
assert list.count(2) == 1
assert list.max() == 3
assert list.min() == 1
def even = list.find { item ->
        item % 2 == 0
}
assert even == 2
assert list.every { item -> item < 5 }
assert list.any { item -> item < 2 }
```

Iterating the content

```
def list = [1, 2, 3]
def store = ''
list.each { item ->
        store += item
}
assert store == '123'
store = ''
list.reverseEach { item ->
        store += item
}
assert store == '321'
store = ''
list.eachWithIndex { item, index ->
        store += "$index:$item "
}
assert store == '0:1 1:2 2:3 '
```

Accumulating the content

```
assert list.join('-') == '1-2-3'
result = list.inject(0) { clinks, guests ->
    clinks + guests
}
assert result == 0 + 1 + 2 + 3
assert list.sum() == 6
factorial = list.inject(1) { fac, item ->
    fac * item
}
assert factorial == 1 * 1 * 2 * 3
```

One more Boromir picture

- It would be great to use [0], but for maps
- In groovy it is possible:

```
def emptyMap = [:]
assert emptyMap.size() == 0
def explicitMap = new TreeMap()
explicitMap.putAll(myMap)
assert explicitMap['a'] == 1
def myMap = [:]
myMap << [b:2] << [c:3]
def composed = [x:'y', *:myMap]
assert composed == [x:'y', a:1, b:2, c:3]
```

Operations on maps

- By key, by value and by entry

```
myMap.containsKey('a')
myMap.containsValue(1)

myMap.each { entry ->
    store += entry.key
}
myMap.each { key, value ->
    store += key
}
```

Maps and duck typing

- Map can be cast to any object in case it has appropriative properties

```
class Person {
    String firstName
    String lastName
}
def personMap = [firstName:'John', lastName: 'Doe']
Person john = personMap
assert john.firstName == 'John'
```

Important collection methods

Groovy	Java 8 stream api
Inject	Reduce
Filter	
findAll	filter
Collect	map
groupBy	collect(groupingBy)
countBy	
collectEntries	collect(toMap)

tasks

Lab

- Write method which will receive list of employees and will return string with their names separated by comma:
- Input:

```
ArrayList<Employee> employees = new ArrayList<>();
employees.add(new Employee("Hirsh"));
employees.add(new Employee("Avishay"));
employees.add(new Employee("Hadas"));
```

- Output: String which contains text: “Hirsh,Avishay,Hadas”

```
Optional<Integer> optional = stream.reduce((x, y) -> x + y);
```

Lab

- Write method which will receive list of employees and will return string with their names separated by comma:
- Write method which will receive list of employees and will return List of their names (only the uppercased ones) sorted by length.
- Write method which take list of employees and return map (name of employee uppercased vs salary per year)

Lab

Lab

- Write method which will receive List of Employees and will return map<CompanyName, List<Employee>>
- Yes each employee has property: String companyName

- Write method which receive List of Employees and will return Map<CompanyName,Integer>
- Integer – is number of the workers

Lab

- There are three categories of employees
- Juniors – salary <14000
- Middle – salary < 21000
- Seniors - >21000
- Create appropriative enum
- Write method which will receive list of employees and return Map of seniority vs list of employees

Lab05

Write backend for Beer House CRM

Lab06

Write method which will receive a file and return number of words

Write method which will receive a file and return average length of the word

Write method which will find X most popular words in text

GPath

- *. -spread dot operator
- list*.member
- works for fields, properties andmethods

GPath

```
def multiply = (1..10)*.multiply(2)
multiply.each {println(it)}
```

Riddle

```
def list=[1,2,3]

println(list.multiply(2))

println(list*.multiply(2))
```

?

?

AST Transformations

- @ToString
- @EqualsAndHashCode
- @TupleConstructor
- @Canonical
- @Lazy
- @IndexedProperty
- @InheritConstructors

AST Transformations

- @Delegate
- @Singleton
- @Immutable
- Loggers
  - @Log –(java utillogging)
  - Log4j
  - @Slf4j
  - @Commons

AST Transformations

- @Synchronized
- @WithReadLock
- @WithWriteLock

AST Transformations

- @AutoClone
  - Classic
  - Copy constructor
  - Serialization
- @AutoExternalize
- @PackageScope
- @Category
- @Mixin

Groovy Script

- If code is written in file (outside the class) it is scipt
- Script code will be placed in generated class
- Code, which exists outside methods, will be place in run method
- In addition main method will be generated, which will create an object of this class and will invoke run method.

Meta-programming in Runtime

Lets add some methods

- Option 1: use Category

```
class StringCalculationCategory {
    static def plus(String self, String operand) {
        try {
            return self.toInteger() + operand.toInteger()
        }
        catch (NumberFormatException fallback) {
            return (self << operand).toString()
        }
    }
}

use(StringCalculationCategory) {
    assert 1 == '1' + '0'
    assert 2 == '1' + '1'
    assert 'x1' == 'x' + '1'
}
```

Lets improve!

- using AST!

```
@Category(String)
class StringCalculationCategory {
    def plus(String operand) {
        try {
            return this.toInteger() + operand.toInteger()
        }
        catch (NumberFormatException fallback) {
            return (this << operand).toString()
        }
    }
}
```

Lets add methods

- Option 2: extension methods
- The same but packaged into jar + descriptor
- META-INF/services/org.codehaus.groovy.runtime.ExtensionModule

```
moduleName=string-calculation
moduleVersion=1.0
extensionClasses=com.acme.StringCalculationCategory
staticExtensionClasses=in this example this line will
not exists
```

Lab09

1. Fight with technical debt –add countDuplicates to List
2. Add all methods of commons-lang StringUtils to String!

Let’s add methods

- Option 3: @Mixin
- Let’s take some class:

```
class First {
    String hello(String name) { "Hello $name!" }
}
```

- And let’s add another class:

```
@Mixin(First)
class Second {
    // more methods
}

assert new Second().hello('JB') == 'Hello JB!'
```

Let’s add methods

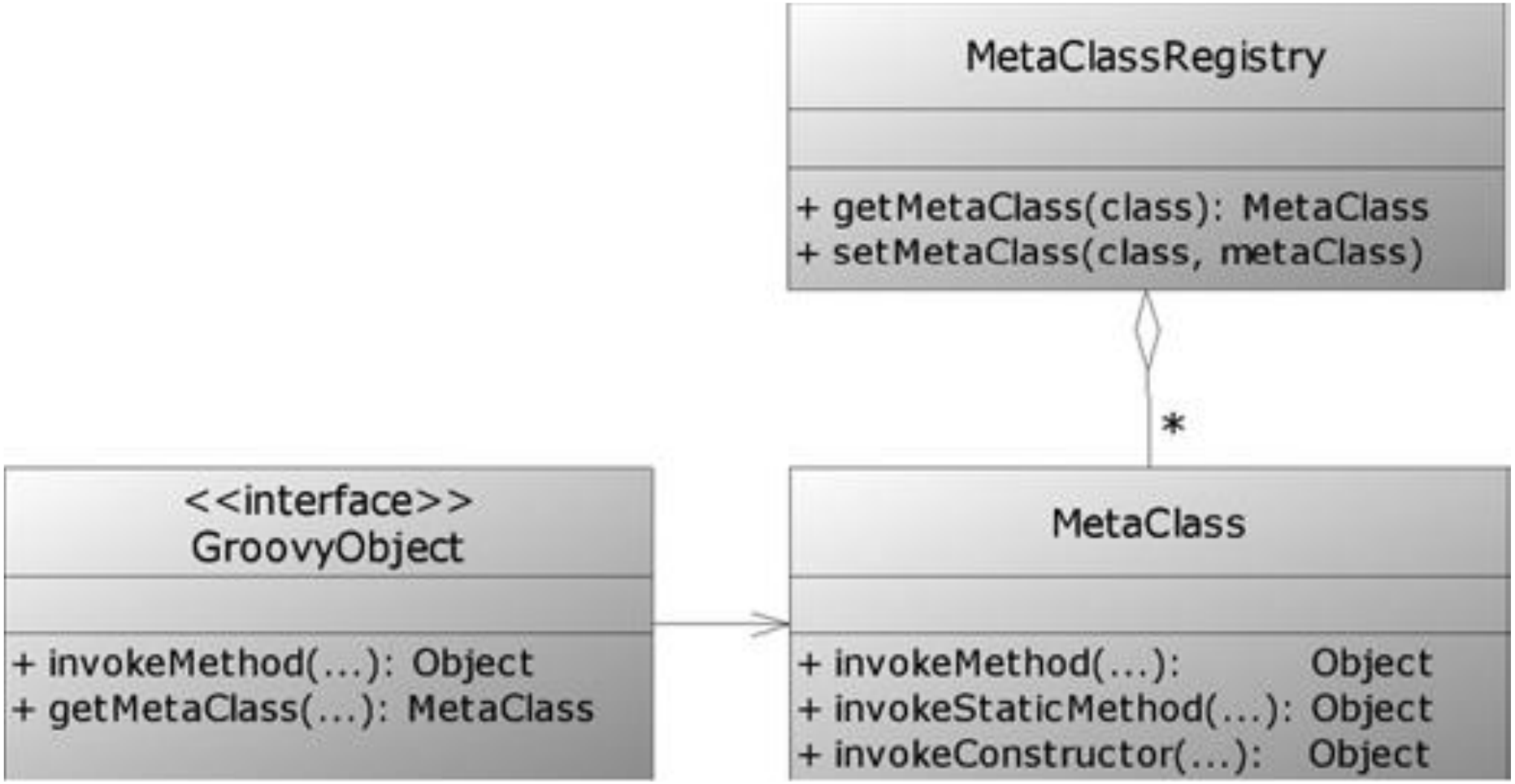
- Option 4:
  Trait replace @Mixin

```
trait Name {
    abstract String name()
    String myNameIs() { "My name is ${name()}!" }
}
trait Age {
    int age() { 42 }
}

class Character implements Name, Age {
    String name() { 'JB' }
}

def player = new Character()
assert player.myNameIs() == 'My name is JB!'
assert player.age() == 42
assert player instanceof Name
assert player instanceof Age
```

GroovyObject

```
package groovy.lang;

public interface GroovyObject {

    Object invokeMethod(String name, Object args);

    Object getProperty(String propertyName);

    void setProperty(String propertyName, Object
newValue);

    MetaClass getMetaClass();

   void setMetaClass(MetaClass metaClass);
}
```

GroovyObject <-> MetaClass Mapping



methodMissing/propertyMissing

- Catch all failed methods
- If invokeMethod/getPropertydeclared, methodMissing/propertyMissing will be never called

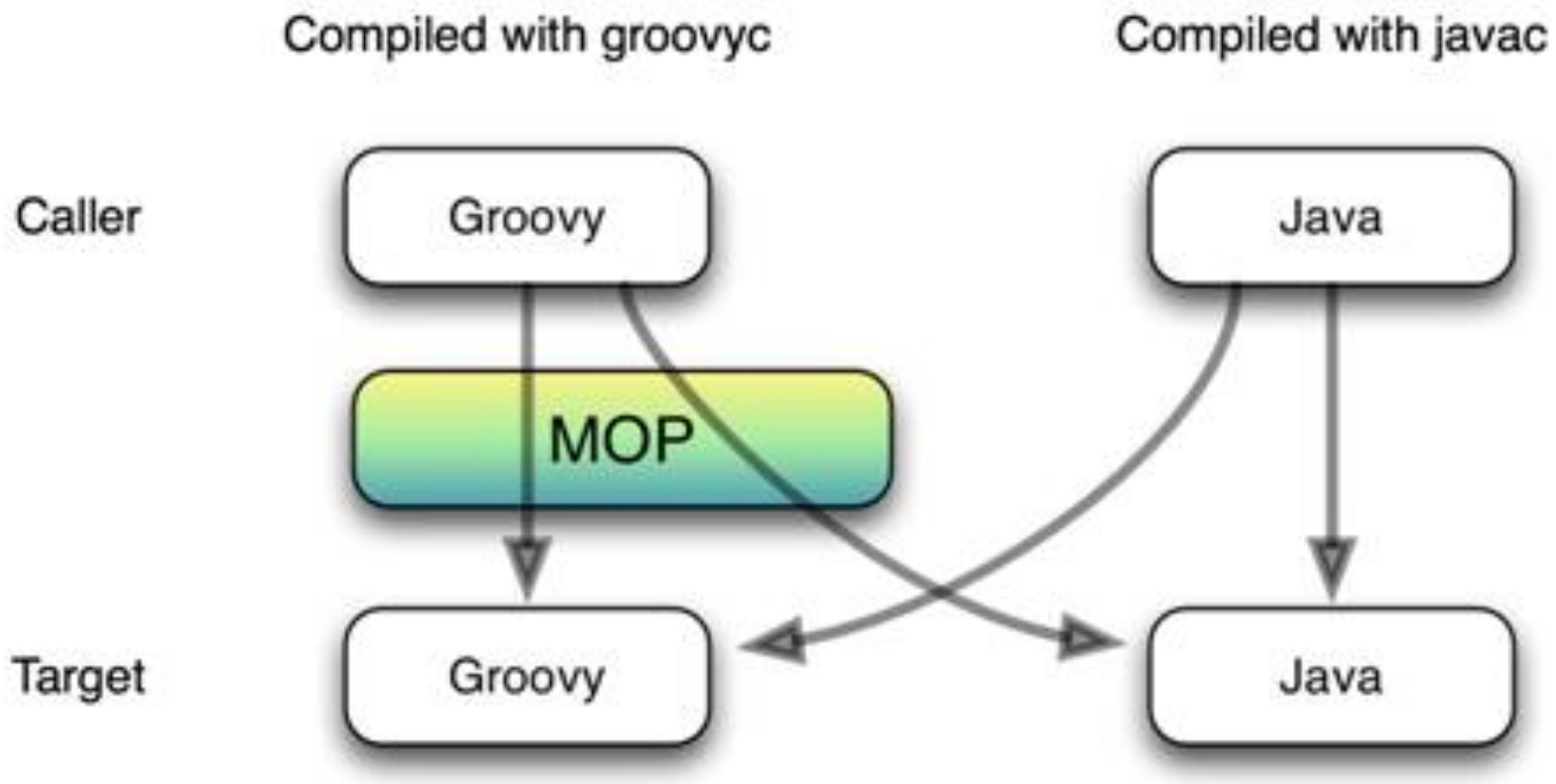
methodMissing/propertyMissing

```
class AnyMethodSpeaker {

    def methodMissing(String name, def args) {
        if(name.startsWith('say') && !args){
            return name - 'say'
        } else {
            throw new MissingMethodException(name, this.class, args)
        }
    }
}
```

methodMissing/propertyMissing

```
def speaker = new AnyMethodSpeaker()
assert speaker.sayBla() == 'Bla'
try {
    speaker.sayBla('param')
    assert false
} catch (mme){
    assert mme.message == 'No signature of method: AnyMethodSpeaker.sayBla() is
       applicable for argument types: (java.lang.String) values: [param]'
}
try {
    speaker.bla()
    assert false
} catch (mme){
    assert mme.message.contains('No signature of method: AnyMethodSpeaker.bla() is
       applicable for argument types: () values: []')
}
```

Cross calls Java <-> Groovy



Games with MetaClass

- You can change metaClass of specific class
  - MyClass.metaClass =
- You can change metaClass of specific object
  - myObject.metaClass =
- But in groovy it can be more simple.
  It’s enough to apply for him and he will be automatically replaced with ExpandoMetaClass, which will contain all methods of DefaultMetaClassa + what we have added or overrided

ExpandoMetaClass

- When we expand MetaClass, usual MetaClass changes to ExpandoMetaClass

Remember I promised more ways to add methods?

```
assert String.metaClass =~ /MetaClassImpl/
String.metaClass.low = {
    delegate.toLowerCase()
}
assert String.metaClass =~ /ExpandoMetaClass/
assert 'JohN'.low() == 'john'
```

Object MetaClass

```
String strWithLow = 'John'
strWithLow.metaClass.low = {
     delegate.toLowerCase()
}
assert strWithLow.low() == 'john'
String strWithoutLow = 'Jane'
try{
    strWithoutLow.low()
} catch (mme){
    assert mme
}
```

MetaClass of static context

```
Integer.metaClass.static.answer = {-> 42}
assert Integer.answer() == 42
```

Lab11

Write parser for properties

Статический Groovy

Чо?

Что мы знаем про Груви

- Компилятор делает много штук:
  - Не надо точек-с-запятыми
  - Операторы превращаются в вызовы метода
  - Удобный синтаксис для списков/мап
  - Новые методы в классах (на самом деле – вызовы статических методов в других классах)

Что мы знаем про Груви:

- Опциональное типизирование
  - Все, что мы не типизируем – Object
  - Хреновые приведения падают в рантайме
- MOP дает динамизм
  - Всё проходит через MetaClass
  - Существование методов и свойств определяется в рантайме
  - Отсутствие оных, к сожалению, тоже (в джаве бы это просто не компилируется)

А что если убить MOP?

- Все фишки компилятора всё ещё в силе
- Вызовы напрямую
  - Скорость как в Java
  - HotSpot опять нормально себя чувствует
- Опциональное типизирование больше не нужно
  - Никаких сюрпризов и методов, возвращающих неизвестно что
- Статический анализ снова работает!
  - Меньше багов в рантайме

- Если динамизм не нужен, имеет смысл его отключить.

@TypeChecked

- Проверяет типы во время компиляции
- Неправильные присвоения
- Вызовы несуществующих методов
- Обращения к несуществующим полям
- Дженерикс проверяются!
- Все еще можно пользоваться def
  - При странных присвоениях будут генерироваться новые типы на лету!

@TypeChecked

- Можно на класс
- Можно на метод
- Можно отключить на метод

```
import groovy.xml.MarkupBuilder
import groovy.transform.TypeChecked

@TypeChecked
class HTMLExample {
   @TypeChecked(TypeCheckingMode.SKIP)
    private static String buildPage(String pageTitle) {
        def writer = new StringWriter()
        def xml = new MarkupBuilder(writer)
        xml.html {
            head {
                title(pageTitle)
            }
        }
        writer
    }

    static String page404() {
        buildPage '404 - Not Found'
    }
}

HTMLExample.page404()
```

Когда использовать TypeChecked

- Помните 2 подхода к вопросу, когда использовать def?
  - Когда вам важен тип
  - Когда вам нужно не знать про тип
- Тут тоже самое – 2 подхода
  - Только когда есть опасность косяков
  - Всегда, кроме случаев, когда не нужно

@CompileStatic

- Отключает вызов методов через MOP
- Bytecode практически как в Джаве
- Скорость как в Джаве!

Иногда нужна помощь

- Например, delegate в closure неизвестен до runtime
- Как проверять?
- Использовать подсказки:
  - @DelegatesTo
  - «Подсказочные» callbacks

```
unresolvedVariable { var ->
    if (var.name=='robot') {
        storeType(var,lookupClassNodeFor('com.Robot'))
        handled = true
    }
}
```

Builders and Slurpers

Builder

- DSL для создания древо-подобных структур
- В коде – NodeBuilder
- XML и HTML – MarkupBuilder
- HttpBuilder
- Swing – SwingBuilder
- И так далее

Принцип builder-а

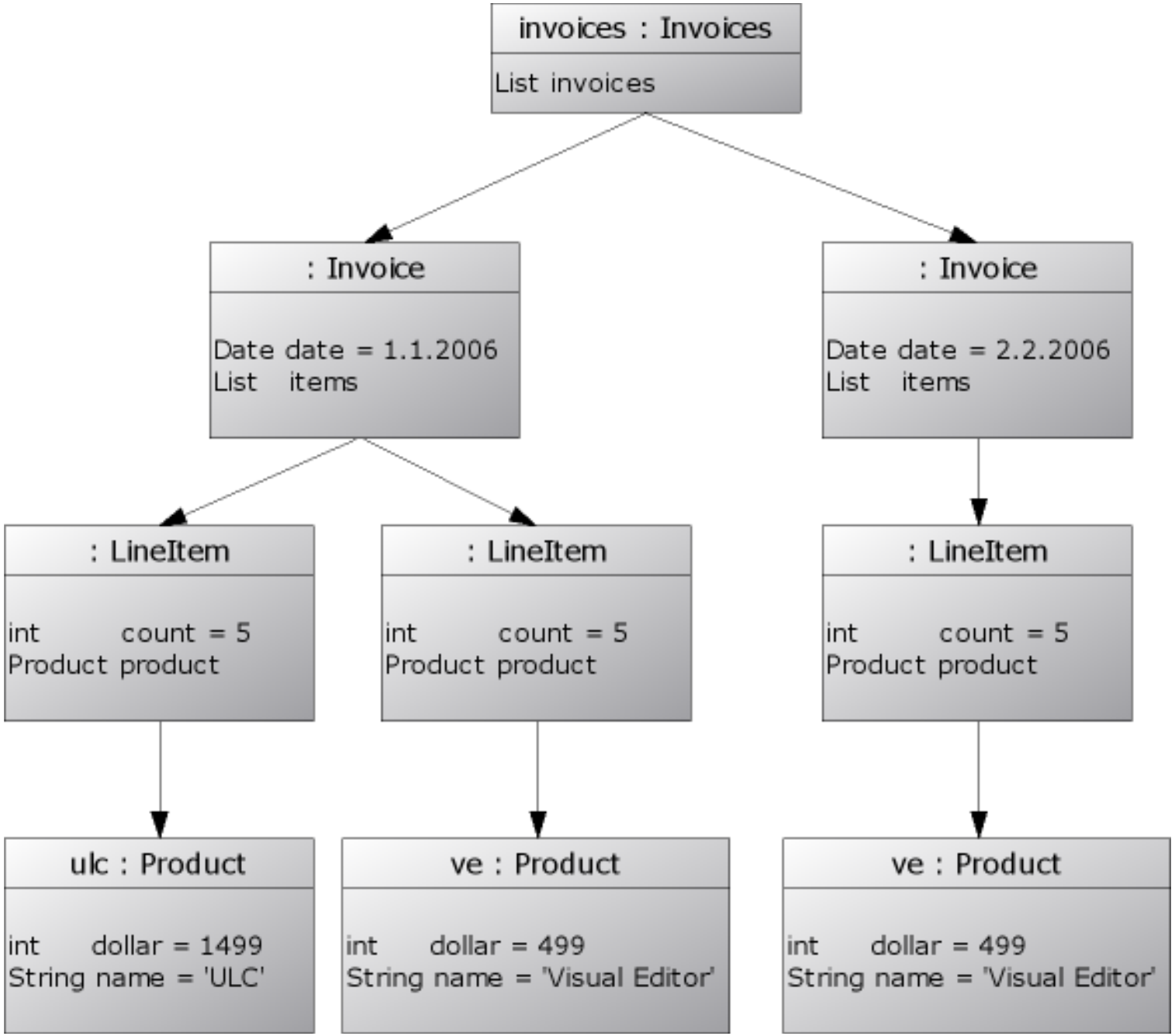
- Поскольку речь идет о древовидной структуре, надо научиться различать ветки и листья.
- Всё.
- Листья – методы
- Ветки – closures, в которых другие ветки и листья
- Иногда, правда, бывают всякие странности
  - Например, атрибуты у тэгов XML
  - Можно использовать еще и поля, параметры к методам и т.д.

```
def writer = new FileWriter('markup.html')
def html = new MarkupBuilder(writer)
html.html {
    head {
        title 'Constructed by MarkupBuilder'
    }
    body {
        h1 'What can I do with MarkupBuilder?'
        form(action: 'whatever') {
            for (line in ['Produce HTML', 'Produce XML', 'Have some fun']) {
                input(type: 'checkbox', checked: 'checked', id: line, '')
                label(for: line, line)
                br('')
            }
        }
    }
}
```

Lab 12

Создаем квитанции и заказы с помощью NodeBuilder

Lab 12


invoices : Invoices
List invoices
: Invoice
Date date = 1.1.2006
List items
: Invoice
Date date = 2.2.2006
List items
: LineItem
int count = 5
Product product
: LineItem
int count = 5
Product product
: LineItem
int count = 5
Product product
ulc : Product
int dollar = 1499
String name = 'ULC'
ve : Product
int dollar = 499
String name = 'Visual Editor'
ve : Product
int dollar = 499
String name = 'Visual Editor'

Slurper

- Обратное от Builder
- Считывает чего-то там (xml, json, html) в структуру на Groovy
- Мы не хотим знать, что это за структура
- def и duck typing рулят

Lab 13

Парсим JSON

Lab 13

- Пример поиска городов в GeoNames
  - http://www.geonames.org/export/JSON-webservices.html#citiesJSON
- Печатаем красивую табличку в консоль:

Time	Place	Magnitude

The GDK

Groovy’s additions to JDK

Object

- dump()
- inspect()
- properties
- is(other)
- isCase(caseValue, switchValue)
- obj.identity {closure}
- sleep(millis)
- use(categoryClass) {closure}
- with {closure}

Object

- print()
- print(value)
- println()
- println(value)
- printf(formatStr, value)
- printf(formatStr, value[])
- sprint(formatStr, value)
- sprint(formatStr, value[])

Обход любого объекта

- Все итерационные методы приходят из Object (да-да!)
- Унифицирущая логика ищет возможные варианты применения методов для объекта

Попытки обхода

No.	Candidate	Use with
1	java.util.Iterator	Itself
2	org.w3c.dom.NodeList	Iterator over Nodes
3	java.util.Enumeration	Convert to iterator
4	java.util.regex.Matcher	Iterator over matches
5	java.lang.Iterable	Iterator.iterator()
6	Responds to iterator method	Call it
7	Collectable	Collection.iterator()
8	java.util.Map	Iterator over Map.Entry objects
9	Array	Iterator over array items
10	MethodClosure	Iterator over calls
11	java.lang.String	Iterator over characters
12	java.io.File	Iterator over lines
13	null	Empty iterator
13	Otherwise	Iterator that only contains the candidate

Properties

```
class MyClass {
    def first = 1 // read-write property
    def getSecond() { first * 2 } // read-only property
    public third = 3 // public field
    def myMethod() { } // public method
}
def obj = new MyClass()
assert obj.hasProperty('first')
assert obj.respondsTo('myMethod')
def keys = ['first', 'second', 'class']
assert obj.properties.keySet() == new HashSet(keys)
assert 1 == obj.properties['first']
assert 1 == obj.properties.first
assert 1 == obj.first
assert 1 == obj['first'] // getAt('first')
def one = 'first'
def two = 'second'
obj[one] = obj[two] // putAt(one)
assert obj.dump() =~ 'first=2'
```

Работа с Файлами

- Правильное управление ресурсами благодаря замыканиям
- Удобные методы для работы с содержимым файлов

Знаете ли вы, как правильно закрыть файл?

1. Это надо делать в finally
2. Сам метод close() кидает checked exception
3. Это значит, что его тоже нужно обвернуть в try-catch
4. И что делать в catch?!

Ответ Java7 – try-with-resources

- Инициализация (открытие) файла происходит прямо в выражении try
  - Сильно не разгуляешься!
- В этом случае, все закроется само
- Уродливо, и неудобно

```
 try (
        ZipFile zf =
              new ZipFile(zipFileName);
        BufferedWriter writer =
Files.newBufferedWriter(outputFilePath,
charset)) {
        …
          }
     }
}
```

The Groovy way

- Использовать замыкания для того, чтобы передать код, который мы хотим выполнить
- Все остальное GDK берет на себя
- Это паттерн, всегда ищите методы with...

```
new File('1.txt').withReader('UTF-8') {reader ->
    reader.eachLine {String line ->
        println line
    }
}
```

Search for: with | ResourceGroovyMethods.java

☐ Show inherited members (Ctrl+F12) ☐ Show Anonymous Classes (Ctrl+I)

ResourceGroovyMethods

- withDataInputStream(File, Closure<T>): T
- withDataOutputStream(File, Closure<T>): T
- withInputStream(File, Closure): Object
- withInputStream(URL, Closure<T>): T
- withObjectInputStream(File, ClassLoader, Closure<T>): T
- withObjectInputStream(File, Closure<T>): T
- withObjectOutputStream(File, Closure<T>): T
- withOutputStream(File, Closure): Object
- withPrintWriter(File, Closure<T>): T
- withPrintWriter(File, String, Closure<T>): T
- withReader(File, Closure<T>): T
- withReader(File, String, Closure<T>): T
- withReader(URL, Closure<T>): T
- withReader(URL, String, Closure<T>): T
- withWriter(File, Closure<T>): T
- withWriter(File, String, Closure<T>): T
- withWriterAppend(File, Closure<T>): T
- withWriterAppend(File, String, Closure<T>): T

☑ Narrow down on typing

Методы для работы с содержимым

- Не существуют в JDK!

```
Path path = Paths.get("1.txt");
Charset charset = Charset.forName("UTF-8");

try (BufferedReader reader = Files.newBufferedReader(path , charset)) {
    while ((line = reader.readLine()) != null ) {
        System.out.println(line);
    }
} catch (IOException e) {
    System.err.println(e);
}
```

Крутой <<

- Оператором << можно записать почти что угодно во что угодно
  - File
  - Writer
  - OutputStream
- Записать можно даже reader!
- Для своих файлов поддержка << обеспечивается имплементацией интерфейса Writable

Запуск внешних процессов

- Java:

```
ProcessBuilder p = new ProcessBuilder("curl", "--show-error",
"--request", "GET",
        "--header", "Accept: application/json", "--user",
userName + ":" + password, getApiRootUrlString());

final Process shell = p.start();
InputStream errorStream= shell.getErrorStream();
InputStream shellIn = shell.getInputStream();
```

- Groovy

```
def proc = "curl --show-error --request GET --header Accept:
application/json --user $userName:$password
${getApiRootUrlString()}".execute()
proc.text
```

Groovlets

- Запускаем полноценный servlet container:

```
@Grab('org.eclipse.jetty.aggregate:jetty-server:8.1.9.v20130131')
@Grab('org.eclipse.jetty.aggregate:jetty-servlet:8.1.9.v20130131')
@Grab('javax.servlet:javax.servlet-api:3.0.1')
import org.eclipse.jetty.server.Server
import org.eclipse.jetty.servlet.*
import groovy.servlet.*
import static org.eclipse.jetty.servlet.ServletContextHandler.*
def server = new Server(1234)
def context = new ServletContextHandler(server, "/", SESSIONS)
context.resourceBase = "."
context.addServlet(GroovyServlet, "*.groovy")
server.start()
```

Groovlets

- Пишем Groovlet с HtmlBuilder

```
html.html{
    head {
        title 'Groovlet Demonstrator'
    }
    body { h1 'Welcome to the World of Groovlets' }
}
```

- Он уже binded в Groovlet под именем html
- А как добавить своё в binding?
  - Наследовать от GroovyServlet и переопределить setVariables
  - Не забываем вызвать super

Lab14

Показываем прогноз погоды:
Всё вместе – читаем URL из файла, обращаемся к URL curl-ом, рисуем результат в Groovlet

Немного GPars

GPars – мощнейшая библиотека для concurrency

- Ничем не уступает, например, Akka
  - Кроме пиара ☺
- Возможность запараллелить обычные closures:

```
import static groovyx.gpars.GParsPool.withPool
def numbers = [1, 2, 3, 4, 5, 6]
withPool {
    assertSquares(numbers.makeConcurrent())
}
def assertSquares(numbers) {
    assert [1, 4, 9, 16, 25, 36] == numbers.collect { it * it }
}
```

Map Reduce

- Прибавляем один, возводим в квадрат, складываем – все паралельно!

```
import static groovyx.gpars.GParsPool.withPool
withPool {
    assert 55 == [0, 1, 2, 3, 4].parallel
            .map { it + 1 }
            .map { it ** 2 }
            .reduce { a, b -> a + b }
}
```

Всякое разное

- private scope сломан
  - К сожалению, будет починен в 3.0 (или нет)
- Методы могут иметь значение по умолчанию
  - def print(String value, String encoding = ‘UTF-8’)
  - Сгенерирует overloading в байткоде
- with{} переносит в контекст объекта, на котором вызван

Всякое разное

- Import alias
  - import com.somecompany.RobotDefaultImpl as Robot
- К названию методов можно обращаться стрингом
  - Снимает некоторые ограничения на Java literals

```
def 'I can name a method with spaces'(){ }
```

О чём не успели поговорить

- GPars как следует
- Тестирование (Spock & Geb)
- DSL-ы
- Прочие проекты в экосистеме (Gradle, Grails, Ratpack, Lazybones…)