Dynamic meta-programming and Domain-specific languages

Václav Pech

NPRG014 2017/2018

http://jroller.com/vaclav

http://www.vaclavpech.eu

@vaclav_pech

Last time agenda

- Groovy syntax and interoperability
- Language dynamism
- Scripting
- Functional programming

Agenda for today

- Dynamic meta-programming
- Domain specific languages
  - DSL-based frameworks – Grails, Griffon
- Builders

Review

Groovy syntax and interoperability

Power assert

```
assert 5 == customer.score
```

```
Exception thrown
17.2.2012 12:30:12 org.codehaus.groovy.runtime.StackTraceUtils sanitize

WARNING: Sanitizing stacktrace:

Assertion failed:

assert 5 == customer.score
         |  |        |
         |  |        4
         |  [score:4]
         false
```

Groovy is functional

```
def multiply = {a, b -> a * b}
def double = multiply.curry(2)
def triple = multiply.curry(3)

assert 4 == multiply(2, 2)
assert 8 == double(4)
assert 6 == triple(2)
```

Closure scope

owner

delegate

this

Collections

```
final emptyList = []
final list = [1, 2, 3, 4, 5]
final emptyMap = [:]
final capitals = [cz : 'Prague', uk : 'London']

final list = [1, 2, 3, 4, 5] as LinkedList
final emptyMap = [:] as ConcurrentHashMap
```

Review

Scripting

Scripting

Evaluate custom Groovy code

At run-time!!!

```
new GroovyShell().evaluate('println Hi!')
```

http://groovyconsole.appspot.com/

Review

Functional programming

Agenda

- Functors
- Monoids
- Function composition
- Endofunctors

Inspired by http://www.slideshare.net/ScottWlaschin/fp-patterns-buildstufflt/

Functors

Dealing with wrapped data

map: ([A], f: A -> B) → [B]

map: (Maybe<A>, f: A -> B) → Maybe<B>

Functors are *mappable* (they have a **map** operation)

Monoids

Aggregating data and operations

Monoids

Aggregating data and operations

- A set of elements
- An operation that combines two elements
- An 'id' element neutral with respect to the operation
- Closure of the set with respect to the operation

1. $a + id = id + a = a$
2. $(a + b) + c = a + (b + c)$
3. $a \in M \,\&\, b \in M \Rightarrow a+b \in M$

Monoids

Reducible – any set of elements from a monoid can be reduced into a single value

reduce: ([A], f: (A, A) → A) → A

Monoids

class Customer {name, address, orders}

vs.

class CustData {orders, totalAmount}

Monoids

class Customer {name, address, orders}

not a monoid

vs.

class CustData {orders, totalAmount}

a monoid

Monoids

class Customer {name, address, orders}

not a monoid

vs.

class CustData {orders, totalAmount}

a monoid

Composing functions

f: A → B

g: B → C

f >> g: A → C

Composing functions

f: A → B

g: B → C

f >> g: A → C

```
def f = {String s → s.size()}

def g = {Integer i → i%2==0 ? true : false}

def h = f >> g
```

Composing functions

f: A → B

g: B → C

f >> g: A → C

Not a monoid

Endofunctors

f: A → A

with composition (**>>**) and an **id()** function form a monoid

[f1, f2, f3, f4, f5, …].reduce(id, >>)

Other monoids of functions

Elements: f: String → Boolean

Other monoids of functions

Elements: f: String → Boolean

id() – returns *true/false*

Operation: logical AND/OR

- Functors
- Monoids
- Function composition
- Endofunctors

Part 1

Dynamic meta-programming

Agenda

Dynamic dispatch

Dynamic cast

Dynamic object creation

Categories

Meta-programming

Dynamic dispatch

The target method is decided at run-time using the run-time type of the arguments

def calculate(String value)

def calculate(Integer value)

calculate('10' as Integer) ???

Dynamic object creation

```
Runnable r = {println 'Asynchronous'} as Runnable
```

Dynamic object creation

Duck-typing

```
Calculator c = [ add : {a, b, → a + b},
                 multiply : {a, b → a * b},
                 increment : {it + 1}
               ] as Calculator

assert 6 == c.multiply(2, 3)
```

Traits

```
trait Flying {
    void fly() {println "I am flying!"}
}
trait Quacking {
    void quack() {println "Quack!"}
}
class Duck implements Flying, Quacking {}
```

Traits

- Componentisation of the design
- Generalized delegation
- Stackable
- Can be specified as argument types

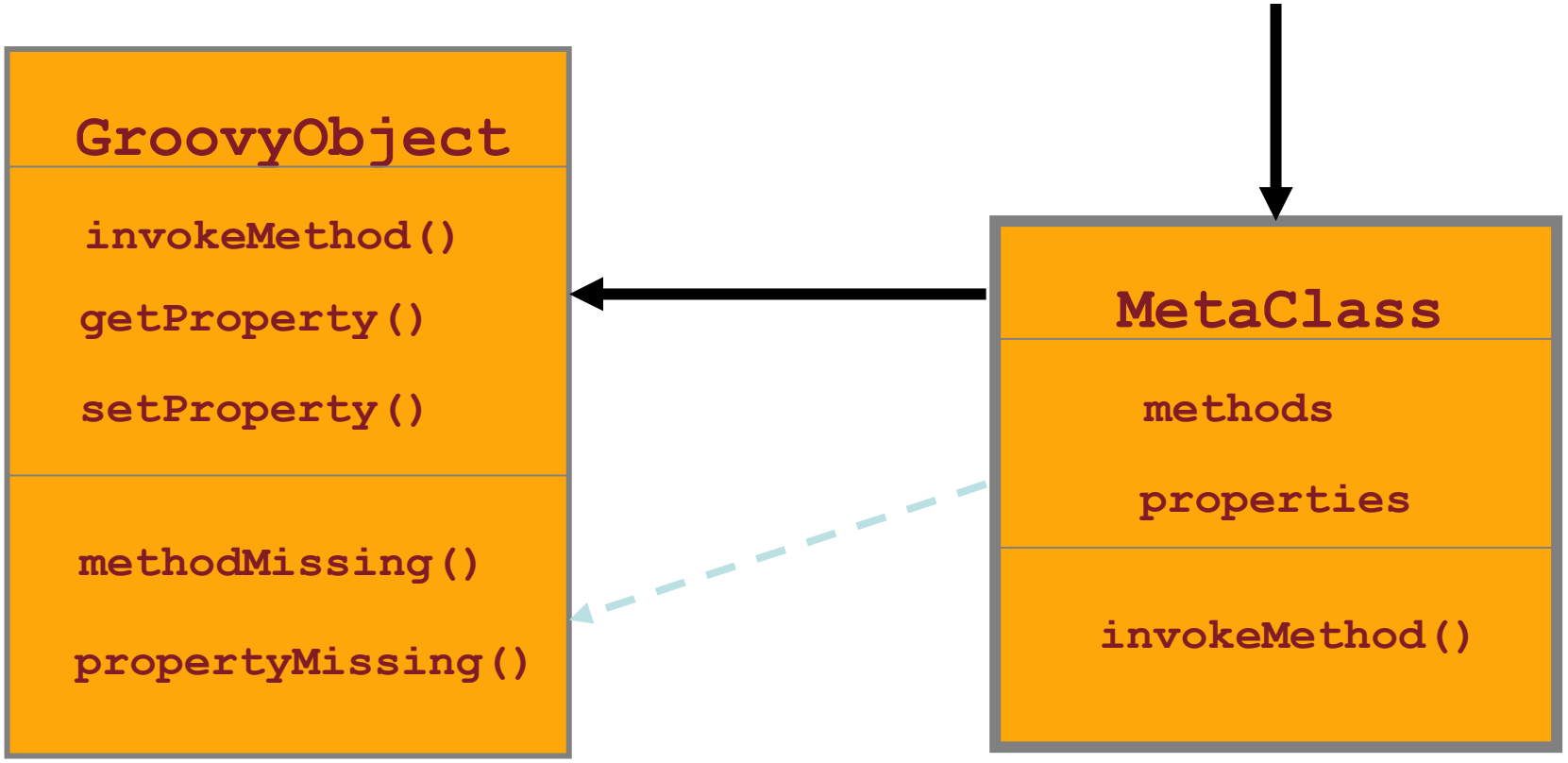
https://speakerdeck.com/melix/rethinking-api-design-with-traits

Categories

```
StringUtils.countMatches(myString, 'Groovy')
```

```
use(StringUtils) {
    myString.countMatches('Groovy')
}
```

Dynamic method invocation



Querying objects' methods

o.respondsTo()

o.hasProperty()

o.metaClass.getMetaMethod(name, args)

o.metaClass.getMetaProperty(name)

Part 4

Domain Specific Languages

```
File.metaClass.div = { path ->
    new File(delegate, path)
}
```

```
File.metaClass.div = { path ->
    new File(delegate, path)
}
def file = new File(".")/'test'/'hello'/'file.txt'
```

Agenda

- Domain-specific languages
- DSL frameworks – Grails, Griffon
- Builders

Grails

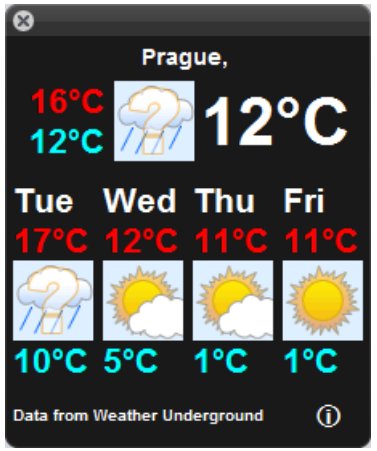
Web applications on JVM

- Hibernate, Spring, ...
- Groovy DSLs
- Scaffolding
- Convention over configuration
- DRY
- KISS

Griffon

Rich-client applications on JVM

- Swing
- Groovy DSLs
- Scaffolding
- Convention over configuration
- DRY
- KISS



BDD - Spock

```
class DataDriven extends Specification {
    def "maximum of two numbers"() {
        expect:
        Math.max(a, b) == c
        where:
        a | b | c
        7 | 3 | 7
        4 | 5 | 5
        9 | 9 | 9
} }
```

http://code.google.com/p/spock/

DSL

- Limited purpose language
- Targeted to a particular domain
- Friendlier API to a framework
  - External
    - SQL, HTML, CSS, …
  - Internal

DSL – Date manipulation

```
use(org.codehaus.groovy.runtime.TimeCategory) {
    println "Tomorrow: ${1.day.from.today}"
    println "A week ago: ${1.week.ago}"
    println "Date: ${1.month.ago + 1.week + 2.hours - 5.minutes}"
    println "Date ${(1.month + 10.days).ago}"
}
```

DSL – Hibernate criteria

```
def participants = Participant.createCriteria().list {
    gt('age', age)
    or{
        eq('interest', 'Java')
        eq('interest', 'Groovy')
    }
    jug {
        ilike('country', 'de')
    }
    order('lastName', 'asc')
}
```

DSL – Account manipulation

```
Money money = new Money(amount: 350, currency: 'eur')
getAccount('Account1').withDraw money
getAccount('Account3').deposit money
```

```
"Account1" >> 350.eur >> "Account3"
```

order cake with plums and apples
and cream to "Malostranske namesti"

```
order(cake).with(plums).and(apples)
.and(cream).to("Malostranske namesti")
```

Builders

Construct hierarchies

- html, xml, json, swing, configuration
- objects
- db queries
- ...

Builders - GAnt

```
ant.sequential {
    myDir = "target/AntTest/"
    mkdir(dir: myDir)
    copy(todir: myDir) {
        fileset(dir: "src/test") {
            include(name: "**/*.groovy")
        }
    }
    List dirs = ['core', 'lib', 'engine', 'gui', 'db']
    for(String currentDir:dirs) {
        String targetDir="target/$currentDir"
        mkdir(dir:targetDir)
```

Cli Builder

```groovy
def cli = new CliBuilder (usage:'simpleHtmlServer -p PORT -d DIRECTORY')
cli.with {
 h longOpt:'help', 'Usage information'
 p longOpt:'port',argName:'port', args:1, type:Number.class,'Default is 8080'
 d longOpt:'dir', argName:'directory', args:1, 'Default is .'
}

def opts = cli.parse(args)
if(!opts) return
if(opts.help) {
  cli.usage()
  return
}
```

Builders – Spring config

```
dataSource(BasicDataSource) {
    driverClassName = "org.hsqldb.jdbcDriver"
    url = "jdbc:hsqldb:mem:shopDB"
}

sessionFactory(ConfigurableLocalSessionFactoryBean) {
    dataSource = dataSource
    hibernateProperties = ["hibernate.hbm2ddl.auto": "create-drop",
            "hibernate.show_sql": true]
}

calculator(demo.shop.CalculatorImpl) {bean ->
    bean.singleton = true
    bean.autowire = 'byType'
}
```

Summary

The joy of Ruby for Java programmers

http://jroller.com/vaclav
vaclav@vaclavpech.eu

References

http://www.groovy.cz

http://groovy.codehaus.org

http://grails.org

http://groovyconsole.appspot.com/

http://www.manning.com/koenig2/