Language dynamism, scripting and functional programming

Václav Pech

NPRG014 2017/2018

http://jroller.com/vaclav

http://www.vaclavpech.eu

@vaclav_pech

Today's agenda

Groovy syntax and interoperability

Language dynamism

Scripting

Aug 2016	Aug 2015	Change	Programming Language	Ratings	Change
1	1		Java	19.010%	-0.26%
2	2		C	11.303%	-3.43%
3	3		C++	5.800%	-1.94%
4	4		C#	4.907%	+0.07%
5	5		Python	4.404%	+0.34%
6	7	▲	PHP	3.173%	+0.44%
7	9	▲	JavaScript	2.705%	+0.54%
8	8		Visual Basic .NET	2.518%	-0.19%
9	10	▲	Perl	2.511%	+0.39%
10	12	▲	Assembly language	2.364%	+0.60%
11	14	▲	Delphi/Object Pascal	2.278%	+0.87%
12	13	▲	Ruby	2.278%	+0.86%
13	11	▼	Visual Basic	2.046%	+0.26%
14	17	▲	Swift	1.983%	+0.80%
15	6	▼▼	Objective-C	1.884%	-1.31%
16	37	▲▲	Groovy	1.637%	+1.27%
17	20	▲	R	1.605%	+0.60%
18	15	▼	MATLAB	1.538%	+0.31%
19	19		PL/SQL	1.349%	+0.21%
20	95	▲▲	Go	1.270%	+1.19%

Groovy

A JVM programming language

- Dynamic
- Dynamically-typed
- Scripting
- Object-oriented
- Building on Java syntax

Groovy

Ecosystem

Grails

Gradle

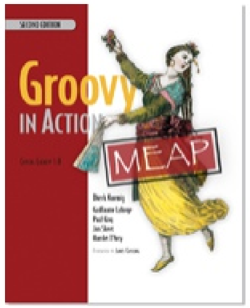
Spock

GPars

Ratpack

Griffon

SDKMAN!



The 7 usage patterns

- Super Glue
- Liquid Heart
- Keyhole Surgery
- Smart Configuration
- Unlimited Openness
- House-Elf Scripts
- Prototype

Examples in Groovy

Part 1

Groovy syntax and interoperability

Interoperability

Groovy and Java can **implement**, **extend**, **refer** and **call** each other at will.

groovyc supports mixed mode

Groovy sources compile into *.class* files

IDEs provide cross-reference support

Java

```
public class Person {
    private final String name;
    public Person(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```

Groovy

```
public class Person {
    private final String name;
    public Person(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```

Groovy

```
public class Person {
    private final String name
    public Person(String name) {
        this.name = name
    }
    public String getName() {
        return name
    }
}
```

Groovy

```
public class Person {
    private final String name
    public Person(String name) {
        this.name = name
    }
    public String getName() {
        return name
    }
}
```

Groovy

```
public class Person {
    private final String name
    public Person(String name) {
        this.name = name
    }
    public String getName() {
        name
    }
}
```

Groovy

```
public class Person {
    private final String name
    public Person(String name) {
        this.name = name
    }
    public String getName() {
        name
    }
}
```

Groovy

```
class Person {
    private final String name
    Person(String name) {
        this.name = name
    }
    public String getName() {
        name
    }
}
```

Groovy

```
class Person {
    private final String name
    Person(String name) {
        this.name = name
    }
    public String getName() {
        name
    }
}
```

Groovy

```
class Person {
    final String name
    Person(String name) {
        this.name = name
    }
}
```

Groovy

```
class Person {
    final String name
    Person(String name) {
        this.name = name
    }
}
```

Groovy is Java

```
class Person {
    final String name
}
```

Variables, constants, params

String a

def a

final a

- Equality a == b
- Identity a.is(b)
- () sometimes optional: println 'Joe'

String interpolation

```
final s = 'Hi Joe'

final s = "Hi Dave"

final s = "Hi $name"

final s = "Hi ${user.name}"

final s = """Hi Dave,

How are you?

"""
```

Numbers and primitive types

15 - integer

15G - BigInteger

1.5 - BigDecimal

1.5d - Double

All values are objects: 5.upto(10)

Clever boxing and unboxing

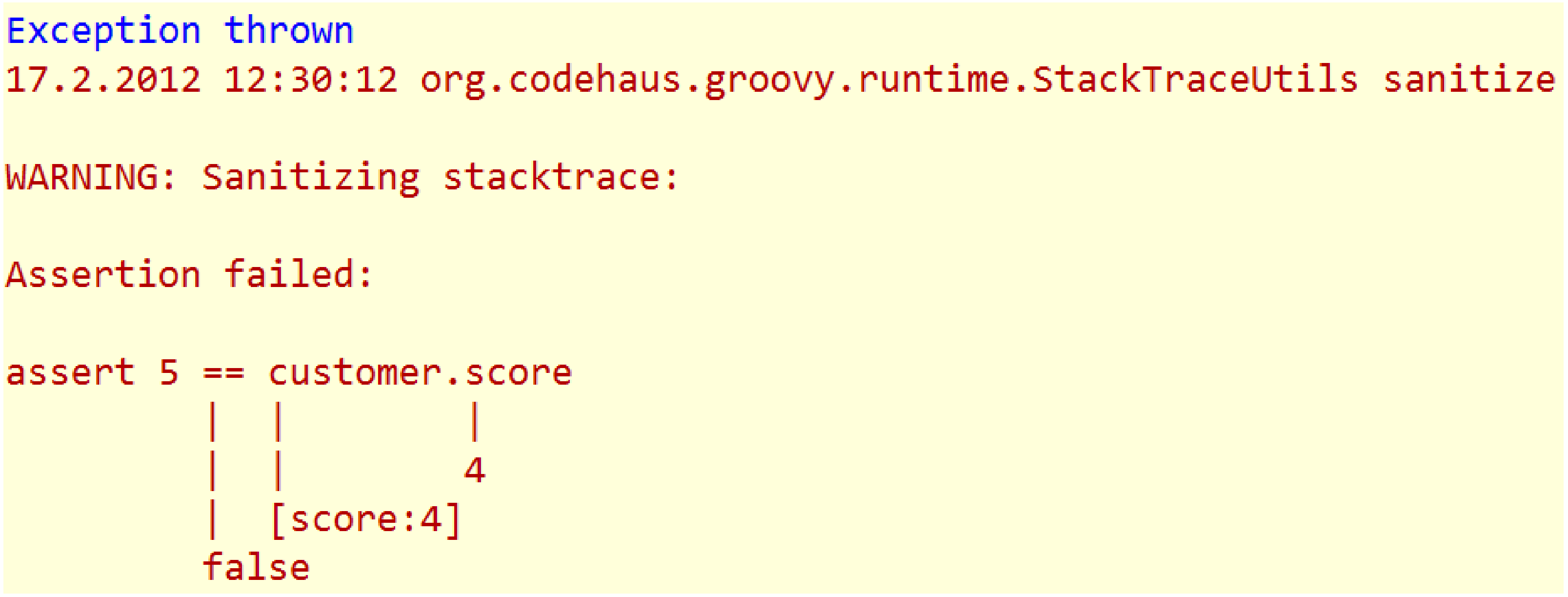
Properties

```
class ProgrammingLanguage {
    String name
    String version
    boolean easy=true
}
def groovy=new ProgrammingLanguage(
        name:'Groovy', version:'1.5', easy:true)

def java=new ProgrammingLanguage(name:'Java')
java.version='1.6'
```

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
```

Closures

```
Closure multiply1 = {int a, int b -> return a * b}

Closure multiply2 = {int a, int b -> a * b}

Closure multiply3 = {a, b -> a * b}

def multiply4 = {a, b -> a * b}
```

Closures – implicit parameter

```
def triple1 = {int number -> number * 3}

def triple2 = {number -> number * 3}

def triple3 = {it * 3}
```

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)
```

Currying vs. Partial application

```
def multiply = {a, b → a * b}


def partial = multiply.curry(3)


def curried1 = {x → multiply.curry(x)}


def curried2 = {x → {y → multiply(x, y)}}
```

Memoize

```
def triple = {3 * it}

def fastTriple = triple.memoize()
```

Closure scope

owner

delegate

this

closure.resolveStrategy =

DELEGATE_FIRST / OWNER_FIRST

DELEGATE_ONLY / OWNER_ONLY

Iterations

```
(1..10).each{number -> println number * 3}

1.upto(10) {println it * 3}


Closure triple = {it * 3}

1.step(11, 1){println triple(it)}
```

(Not exhaustive) list

each (aka for loop)

collect (aka map)

inject (aka reduce)

findAll (aka filter)

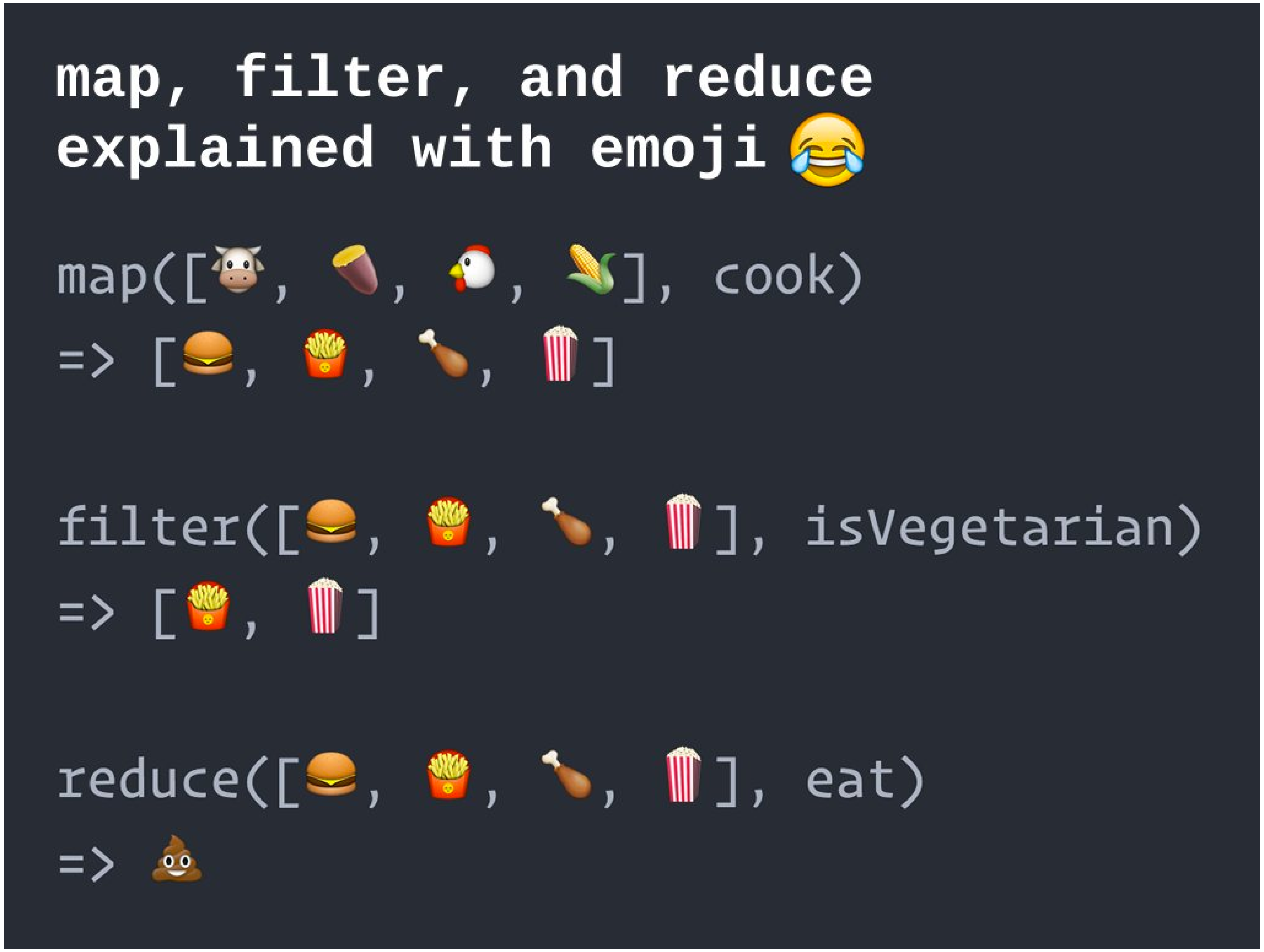
sum, size, findFirst, grep, groupBy

any, every, min, max, ...

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
```



map, filter, and reduce explained with emoji 😂

```
map([🐮, 🍠, 🐔, 🌽], cook)
=> [🍔, 🍟, 🍗, 🍿]

filter([🍔, 🍟, 🍗, 🍿], isVegetarian)
=> [🍟, 🍿]

reduce([🍔, 🍟, 🍗, 🍿], eat)
=> 💩
```

Some operators

```
[‘Java’, ‘Groovy’]*.toUpperCase()

customer?.shippingAddress?.street

return user.locale ?: defaultLocale
```

GDK = JDK + FUN

- java.util.Collection
  - each(), find(), join(), min(), max() …
- java.lang.Object
  - any(), every(), print(), invokeMethod(), …
- java.lang.Number
  - plus(), minus(), power(), upto(), times(), …

Tip: Ask *DefaultGroovyMethods* for help

Syntax enhancements

- Dynamic (duck) typing – optional!
- GDK
- Syntax enhancements
  - Properties, Named parameters
  - Closures
  - Collections and maps
  - Operator overloading
  - …

Part 2

Scripting

Agenda

- Scripting
- Script engine customization
- Grabbing libraries

Scripting

Evaluate custom Groovy code

At run-time!!!

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

http://groovyconsole.appspot.com/

Script customization

- *CompilerConfiguration*
- *CompilationCustomizer*
    - ImportCustomizer
    - ASTCustomizer
    - SecureASTCustomizer

Part 3

Functors and monoids

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

$$\Big\downarrow \text{map}$$

vs.

class CustData {orders, totalAmount}

a monoid

Composing functions

f: $A \rightarrow B$

g: $B \rightarrow C$

f >> g: $A \rightarrow C$

Composing functions

$f: A \rightarrow B$

$g: B \rightarrow C$

$f >> g: A \rightarrow C$

```
def f = {String s → s.size()}
def g = {Integer i → i%2==0 ? true : false}
def h = f >> g
```

Composing functions

$f: A \to B$

$g: B \to C$

$f >> g: A \to C$

Not a monoid

Endofunctors

f: A $\rightarrow$ A

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

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

Other monoids of functions

Elements: f: String $\rightarrow$ Boolean

Other monoids of functions

Elements: f: String → Boolean

id() – returns *true/false*

Operation: logical AND/OR

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/