Intro to Groovy

by Craig L. Jones

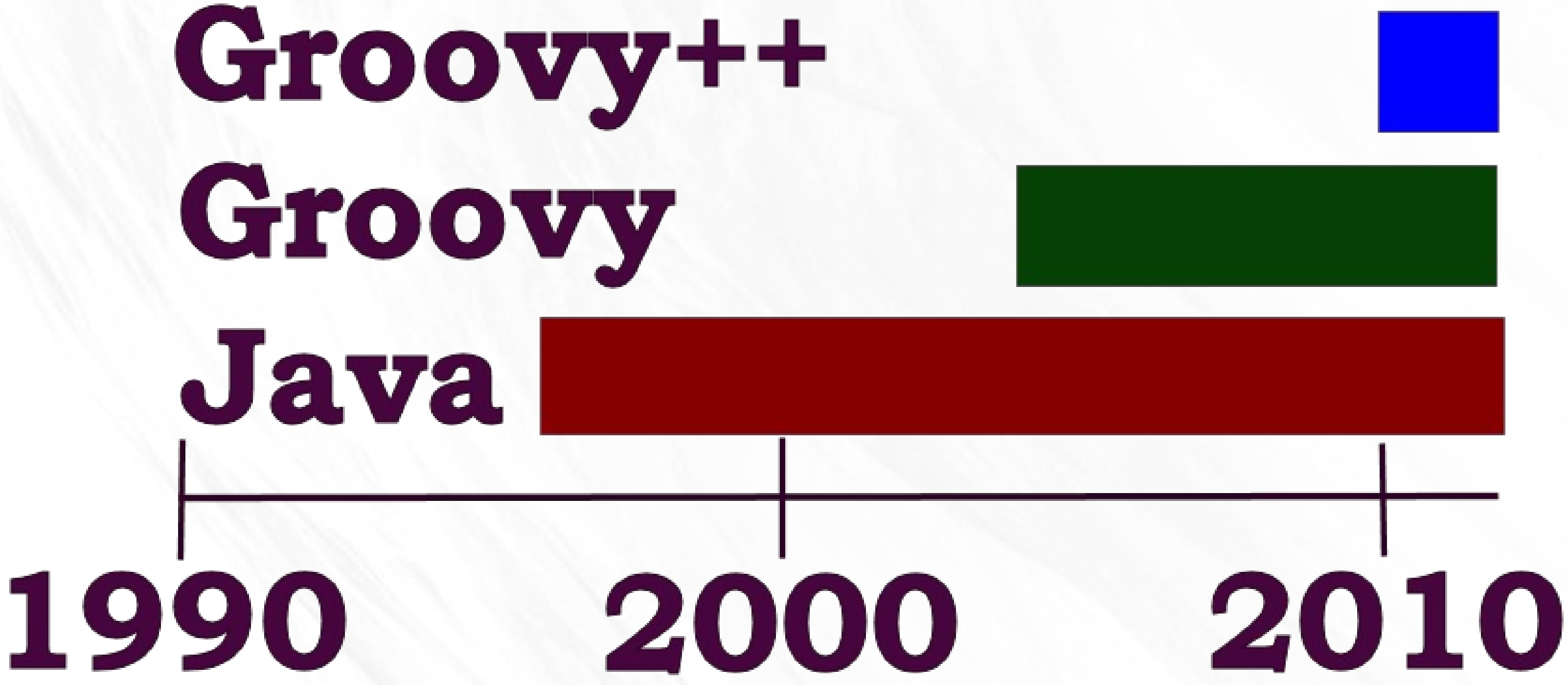
craig@ChiefSimplicityOfficer.com

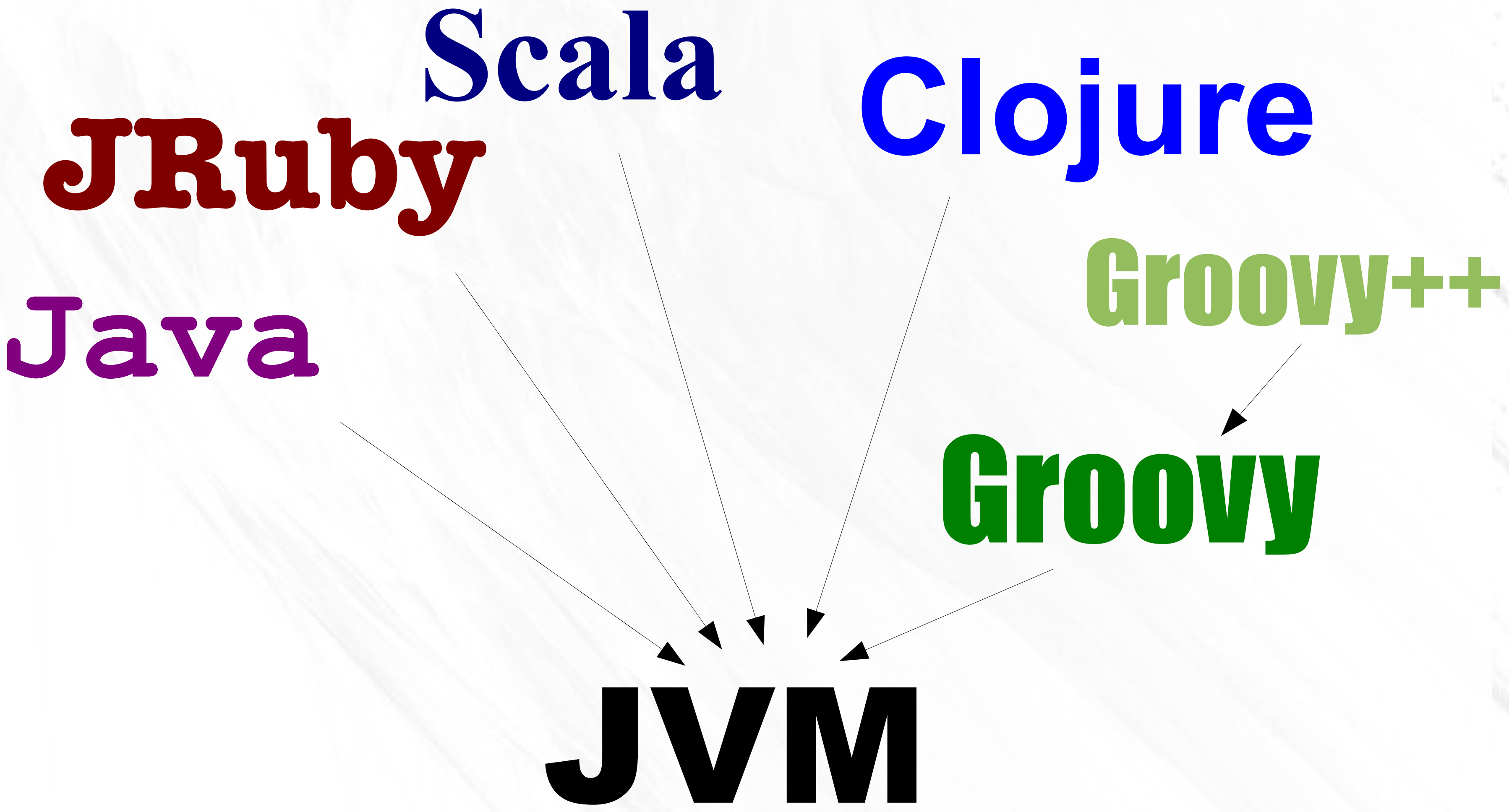
30 Years Software Development

Basic, Fortran, Cobol, C/C++,
Z80 Assember, Foxpro, Paradox,
Pro-IV, Pascal/Delphi, Java, C#,
PHP, JavaScript, Groovy

(Algol, RPG, PL/1, Prolog, Lisp,
Snobol, APL, Objective-C, Scala)

www.ChiefSimplicityOfficer.com


Groovy++
Groovy
Java
1990
2000
2010


Scala
Clojure
JRuby
Groovy++
Java
Groovy
JVM

Java	Groovy	Groovy++
• Best Performance	• **Syntatic Sugar**	• Most Groovy Features
• Too Strict	• **Dynamic Features**	• No Dynamic Overhead
• Too Verbose	• **Library Extensions**	• Stricter Compilation
• Missing Features	• Performance Issues	• Safer Code
		• Regained Performance

Signal-to-Noise Ratio

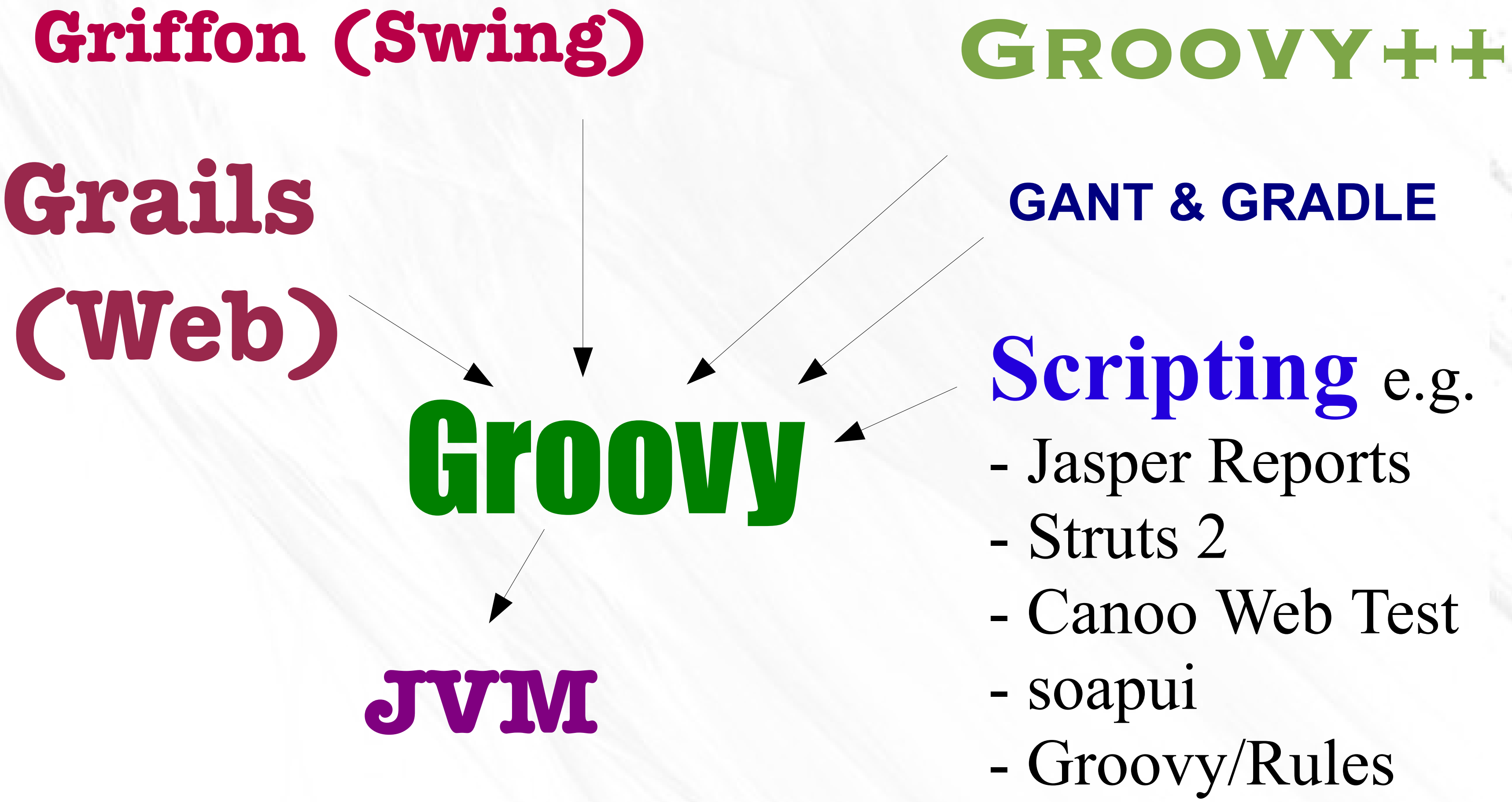
Java

```java
List foo = new
   ArrayList();
foo.add("bar");
foo.add("baz");

String s = "";
if (foo != null &&
  foo.size() > 0) {
    s = foo.get(0);
}
```

Groovy

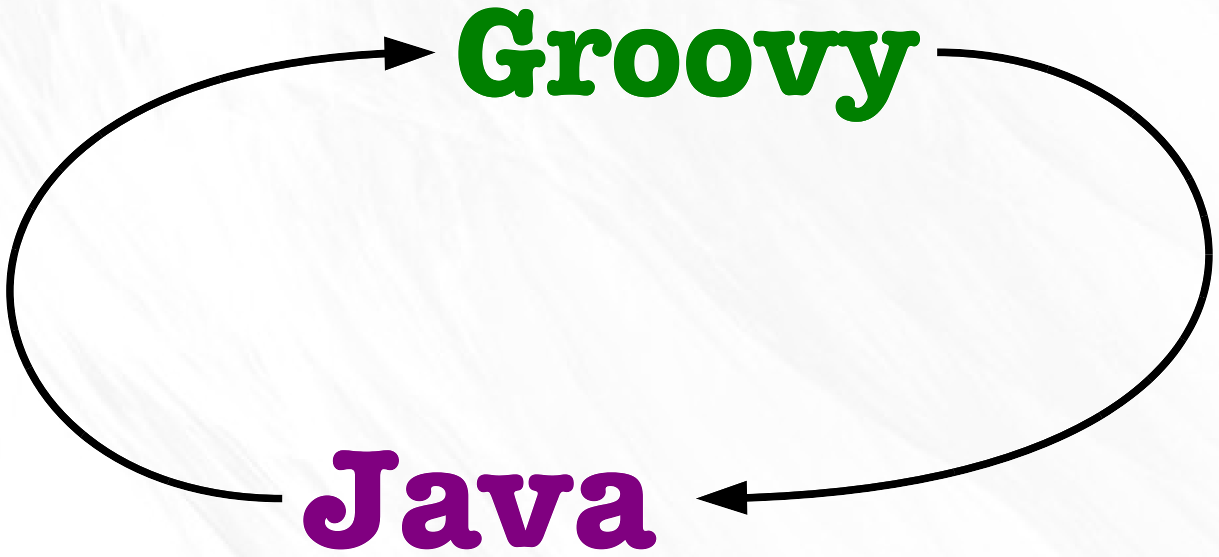
```groovy
def foo = ['bar', 'baz']

def s = foo?.first()
```

The World of Groovy



- Jasper Reports
- Struts 2
- Canoo Web Test
- soapui
- Groovy/Rules

The World of Groovy



Groovy Highlights

(The 20~40% of Groovy with 80% of the Power)

- Dynamic Typing
- Closures
  ==> framework says what, you say how
- Groovy Logic & Syntatic Sugar basics
  ==> supports DSL's, named parameters, etc.
- Mixins & Operator Overloading
  ==> allows for GORM's magic, etc.
- Library extensions (lists, maps, ranges)

Dynamic Typing

```
Map<String> params = new HashMap()
```

$$\Downarrow$$

```
def params = new HashMap()
```

$$\Downarrow$$

```
// in a script
params = new HashMap()
```

Dynamic Typing

Methods, too...

```
public int grandTotal(int group)
{ ... }
```

⇓

```
public def grandTotal(group) { ... }
```

Closures

```
Closure c = {println “Hi”}
def c = {println “Hi”}

> c()
Hi
```

Closures

```
Closure c = {println “Hi”}
> println (c instanceof Runnable)
true

> c.run()
Hi
```

Closures

```
def c = {msg -> println msg}

> c(“Hi”)
Hi
```

Closures

```
def action = {it -> println it}
> colors.each(action)
                 ⇓
> colors.each({it -> println it})
red
green
blue
```

Closures

```
def c1 = {s -> println s}

> c1("Hi")
Hi

def c2 = {s,s2 -> println s+' '+s2}

> c2("Hello","World")
Hello World
```

Closures

```
def c1 = {println it}

> c1("Hi")
Hi
```

Closures

```
def c2 = {s,s2 -> println s+' '+s2}
def c3 = c2.curry("Hello")
                          ↘
def c3 = {s2->println "Hello "+s2}

> c3("World")
Hello World
> c3("Galaxy")
Hello Galaxy
> c3("Universe")
Hello Universe
```

Syntatic Sugar

ie. “Noise reduction”

Default Imports

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

Optional Semicolons

Semicolons are optional at end-of-line.

(They are still required between statements on the same line.)

Embedded Quotes & Heredocs

```
String a = "'quoted'"
String a = '"quoted"'
String a = '\'quoted\''
String a = "\"quoted\""

String html = """
  <p style="color:'green'">
"""
```

Optional **Return** keyword

```
int sum(int a, int b) {
    return a+b
}
```

$$\Downarrow$$

```
int sum(int a, int b) {
    a+b
}
```

Optional Parens

```
assertEquals(12,lines.count())
```

$$\Downarrow$$

```
assertEquals 12,lines.count()
```

Optional Parens

```
colors.each({println it})
```

$\Downarrow$

```
colors.each{println it}
```

Operator Overloading

```
List languages = new ArrayList()

languages.add("Java")
```

$\Downarrow$

```
languages << "Groovy"
```

Operator Overloading

The man behind the curtain...

```
List {
  pubic void leftShift(a) {
    this.add(a)
  }
}
```

Operator Overloading

```
==  !=  equals(a)
+ -     plus(a)     minus(a)
* /     multiply(a)  div(a)
%       mod(a)
++ --   next()  previous()
& |     and(a)  or(a)
a[b]    getAt(b)
a[b] = c  putAt(b)
<< >>   leftShift(a) rightShift(a)
< > <= >=   compareTo(a)
```

Operator Overloading

```
**     power(a)
^      xor(a)
~      bitwiseNegate()
+, -   (unary) positive(), negative()
>>>    rightShiftUnsigned(a)
as     asType(A)
```

List Notation

```
List colors = new ArrayList()
colors.add('red')
colors.add('green')
colors.add('blue')
```

$\Downarrow$

```
List colors = ['red','green','blue']
```

List Notation

```
List colors = new ArrayList()
```

$$\Downarrow$$

```
List colors = []
```

Hash Map Notation

```
Map params = new HashMap()
params.put('filename','data.txt')
params.put('buffersize',512)
```

$\Downarrow$

```
Map params = ['filename':'data.txt',
    'buffersize':512]
```

$\Downarrow$

```
Map params = [filename:'data.txt',
    buffersize:512]
```

(Simulated) Named Parameters

Optional brackets when passed as argument

```
parser.parse(['file':'myscript.txt',
    'eol':'unix'])
```

$\Downarrow$

```
parser.parse(file:'myscript.txt',
     eol:'unix')
```

Autoboxing

```
int i = 2
> println i.class
java.lang.Integer

> 2.class
java.lang.Integer

> 3.times{println “Hi”}
Hi
Hi
Hi
```

A Few Other Differences

- You can use the `this` keyword inside static methods (which refers to this class).
- Methods and classes are public by default.
- Inner classes are not supported (use closures instead).
- The `throws` clause is ignored

Optional Class Structure

// echo.java

```
Package com.example.stuff
import System.out.*
public class Echo {
    public static void main(
        String[] args) {
        for (String s: args) {
            println(s)
} } }
```

$$\Downarrow$$

// echo.groovy (or just **echo)**

```
for (s in this.args) { println(s) }
```

Running a Script

```
-$ java Echo red green blue
```

$\Downarrow$

```
-$ groovy Echo red green blue
```

$\Downarrow$

```
-$ echo red green blue
```

```
#!/usr/bin/env groovy
```

GStrings

```
String villain = 'Dr Chaotica'
int dimension = 5
> "Bow to the will of ${villain}
from the ${dimension}th dimension"
Bow to the will of Dr Chaotica from
the 5th dimension
```

GStrings

```
String villain = 'Dr Chaotica'
int dimension = 1
> "Bow to the will of ${villain}
from the ${dimension}${dimension==1?
'st':'th'} dimension"
Bow to the will of Dr Chaotica from
the 1st dimension
```

GStrings

```
String name = 'World'
String s = "Hello, ${name}!"
name = 'Universe'
> println s
Hello, World!
```

GStrings

```
String name = 'World'
def    s = "Hello, ${name}!"
name = 'Universe'
> println s
Hello, World!      // this?
Hello, Universe!  // or this?
```

GStrings

```
String name = 'World'
def    s = "Hello, ${name}!"
s.values[0] = 'Universe'
> println s
Hello, World!      // this?
Hello, Universe!   // or this?
```

Beans

```
Class Buffer {
  int getPosition() {
    return absolutePosition-offset
  }
}

int index = buffer.getPosition()
                          ⇓
int index = buffer.position
```

Beans

```
Class Buffer {
  boolean isAtEnd() {
    return absolutePosition
      == bufferSize
  }
}

int index = buffer.isAtEnd()
```

$$\Downarrow$$

```
int index = buffer.atEnd
```

Beans

```
Class Buffer {
  void setPosition(int index)
}

buffer.setPosition(128)
```

$\Downarrow$

```
buffer.position = 128
```

Beans

```
Class Buffer {
  int position
  int getPosition()               // implied
  void setPosition(int ix) // implied
}

buffer.setPosition(128)
buffer.position = 128   // implied
```

Beans

```
Class Buffer {
  int position
  int getPosition()              // implied
  void setPosition(int ix) // implied
}

buffer.position = 128   // (property)
buffer.setPosition(128)    // implied
buffer.position = 128   // implied (field)
```

Groovy Logic

False Values in Groovy

false	(boolean)
null	
''	(empty string)
0	(zero)
[]	(empty List)
[:]	(empty Map)

Groovy Logic

```
if (a != null) && (a != 0) { ... }
```

⇓

```
if (a) { ... }
```

Safe Navigation

```
if (a != null)
    { println a.height }
else
    { println a }
```

$\Downarrow$

```
println a?.height
```

Ternary Operator Shortcut

a.k.a. the Elvis Operator

```
println a ? a : 'sensible default'
```

$\Downarrow$

```
println a ?: 'sensible default'
```

20% Mark

Go! Be Groovy!

Regular Expressions

Library Code and Special Syntax

```
def s = 'A 3rd fox at Oxford.'
assertTrue s =~ /ox/
assertTrue s =~ /\d+/
assertFalse s =~ /^\d+/
```

(Uses the Pattern and Matcher classes.)

List Helper Methods

```
getAt(int)           list[i]
leftShift(obj)       list << a
list.each{println it}
eachWithIndex{it,index -> ...}
sort()
reverse()
pop()
list.find{expr}
list.findAll{expr}
Concatenation      list += list2
                   list -= list2
```

List Helper Methods

```
scores.max()
scores.min()
scores.sum()
list.collect{it.toUpper()}
flatten()
```

Spread operator `*list`

Spread-dot operator `list*.toUpper()`

Map Syntax

```
Map options =
[audio:'XFM',tailgate:'barn doors']

s = options.get('audio')
s = options['audio']      // array like
s = options.audio         // property like

options.put('audio','CD Changer')
options['audio'] = 'CD Changer'
options.audio = 'CD Changer'
```

Property Constructors

```
class Person {
  String first
  String last
}

Person p = new Person(first:'John',
last:'Smith')
```

Ranges

```
Range r = 2..4
println r.class
===> groovy.lang.IntRange

r.each{println it}
===> 2
3
4
```

Ranges

Works on any class that implements the **`Comparable`** interface and has **`next()`** and **`previous()`** methods.

```
Date today = new Date()
Date nextWeek = new Date() + 7
(today..nextWeek).each{
                checkBusyCalendar(it) }
```

Ranges

```
Range r = 2..4
println r.size()
===> 3

println r.from    // i.e. getFrom()
===> 2

println r.to      // i.e. getTo()
===> 4

println r.contains(5)
===> false
```

Ranges

```
Range r = 2..4
for (i in r) {println i}
===> 2
3
4

for (i in r.reverse()) {println i}
===> 4
3
2
```

Other Helpers

- String access via array indexing

```
def foo = 'A fox at Oxford.'
> println foo[2]
f

> println foo[-1]
.

> println foo[-7..-2]
Oxford
```

40% Mark

Go! Be Groovier!

Expando class

```
def player = new Expando()
player.name = 'John'
player.greeting = { "Hello, my name
is $name" }

> println player.greeting()
Hello, my name is John

player.name = "Susan"
> println player.greeting()
Hello, my name is Susan
```

The MethodMissing Method

A catch-all method for making up new method calls on the fly (and/or new properties).

- Supports GORM
- Supports DSL's & Builders

GORM

```
Class Customer { // (potential) customers
   String name
   String contact
   int pipeline // 0 = lead, 1 = prospect, 2 = customer
   String zipcode
   String region
}

List<Customer> prospects =
   Customer.FindAllByPipeline(1);
```

“With Great Power...”

```
def s={d->c={b=[it%9]
8.times{b<<b[-1]+9}
int y=it/9
d[y*9..<y*9+9]+d[b]}
def p=d.indexOf('0')
if(p<0)
  print d
else
  ('1'..'9').each{if(!
  (c(p)=~it))s((d<<"")
  .replace(p,p+1,it))}}
```

Putting It All Together

Rule #1: Be consistent with the optional syntax – especially with your team mates – but also the rest of the Groovy community. Keep asking, “is this idiomatic Groovy?”

Corollary: Think like a speed reader.

Idiomatic Groovy

- Declare all variables except Closure, unless truly being dynamic

```
List colors = ['red','green']

int index = 0

def action = {whistle.toot()}

def item = getArbitraryItem()
engine.parse(item) // overloaded
```

Idiomatic Groovy

- Use parens with method calls except when passing Closures, or when the method call acts like a language “command”

```
... colors.collect{it.toUpper()}


println results.join('\n')

assertEquals 10,results.size()

log.debug “Parsed with ${option}”
```

Idiomatic Groovy

- Use GStrings instead of string concatenation

```
'Total of '+errors.count()+'
errors.'
```

$$\Downarrow$$

```
“Total of ${errors.count()} errors.”
```

Idiomatic Groovy

- If Statement Order of Preference...

1) `myString ?: “default”` // Elvis Operator
2) `s = myBool ? 'yes' : 'no'` // Ternary
3) `if (myBool) {s = 'yes'} else {s = 'no'}`

Idiomatic Groovy

- For-Loop Order of Preference...

1) `myList.each{...}` // Closure
2) `for (i in [0..9])` // “in” syntax
3) `for (i=0; i<=9; i++)` // C++ style

e.g. SHA1 for File Contents

```
-$ groovy sha1.groovy myLargeFile.iso

import java.security.MessageDigest
final int MB = 1024**2
File file = new File(args[0])
if (!file.exists() || !file.isFile()) {...}
def digester = MessageDigest.getInstance("SHA1")
file.eachByte(MB) { byte[] buf, int bytesRead ->
  digester.update(buf, 0, bytesRead);
}
def sha1Hex = new BigInteger(1,
 digester.digest()).toString(16).padLeft(40,'0')
println sha1Hex
```

e.g. Export SQL as XML

```
SELECT id, name, contactname, region FROM
customers
WHERE pipeline=1
```

```
<prospects>
  <customer id='870872'>
    <name>ABTI</name>
    <contact>Jean</contact>
    <region>West</region>
  </customer>
...
</prospects>
```

e.g. Export SQL as XML

```
import groovy.sql.Sql
def schema = "PROD"
def sql =
Sql.newInstance("jdbc:oracle:thin:@hostname:1526
:${schema}", "user", "secret",
"oracle.jdbc.driver.OracleDriver")

def query = """SELECT id, name, contactname,
region FROM ${schema}.customers
WHERE pipeline=1
"""
```

e.g. Export SQL as XML

```
import groovy.xml.MarkupBuilder
def out = new File('out.xml')
def writer = new FileWriter( out )
def xml = new MarkupBuilder( writer )

xml.prospects {
  sql.eachRow( query as String ) { row ->
    xml.customer(id:row.id) {
      name( row.name )
      contact( row.contactname )
      region( row.region )
    }
  }
}
```

e.g Greedy Coin Changer

```
enum UsCoin {
  quarter(25), dime(10), nickel(5), penny(1)
  UsCoin(v) { value = v }
  final value
}
def plural(word, count) {
  if (count == 1) return word
  word[-1]=='y'?word[0..-2]+'ies':word+'s'
}
def change(currency, amount) {
  currency.values().collect{ coin ->
    int count = amount / coin.value
    amount = amount % coin.value
    "$count ${plural(coin.name(),count)}"
} }
```

Groovy++

```
@Typed package mypackage

["Hello, ", "World!"].each {
    print it.toLowerCase()
}
println()
```

Groovy++

- Compile-time checking
- High performance (== Java)
- Statically and/or dynamically typed code
- Powerful type inference
- Tail recursion (optimization that supports functional programming)
- Traits (interfaces with default implementation)
- Extension methods (compile time categories)

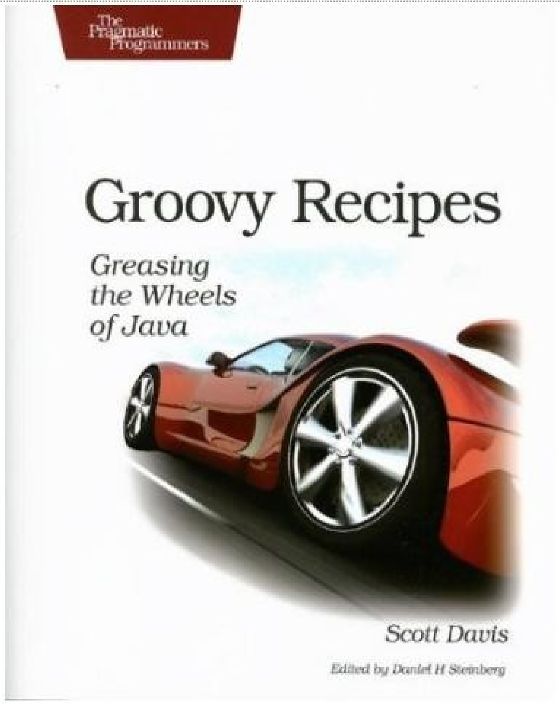
Groovy++

- Utility libraries (pending)
  - Functional programming
  - Concurrency
  - Distributed computing

References

- http://groovy.codehaus.org/
- http://refcardz.dzone.com/refcardz/groovy
- http://grails.org/
- http://griffon.codehaus.org/

Books

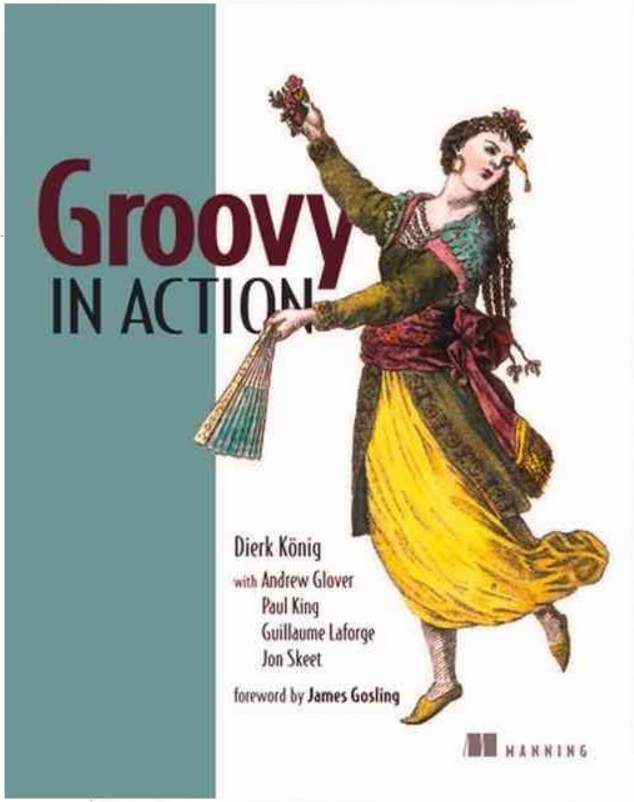

The Pragmatic Programmers

Groovy Recipes

Greasing the Wheels of Java

Scott Davis

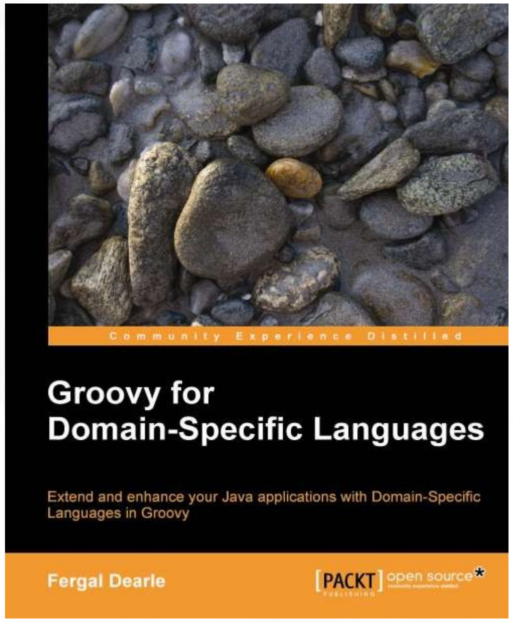
Edited by Daniel H Steinberg


Groovy IN ACTION

Dierk König
with Andrew Glover
Paul King
Guillaume Laforge
Jon Skeet

foreword by James Gosling

MANNING


Community Experience Distilled

Groovy for Domain-Specific Languages

Extend and enhance your Java applications with Domain-Specific Languages in Groovy

Fergal Dearle

[PACKT] open source*

http://www.ChiefSimplicityOfficer.com

If you would like to schedule this, or another of my presentations, for your group, don't hesitate to contact me.

craig@chiefsimplicityofficer.com
714-955-4025