DEVNEXUS™

Groovy - getting started and practical

in hours*

*if you're a good Java developer

About your speaker

github.com/jbaruch

@jbaruch

linkd.in/jbaruch

stackoverflow.com/users/402053/jbaruch

What Frog?

What Frog?

What Frog?

artifactory

What Frog?

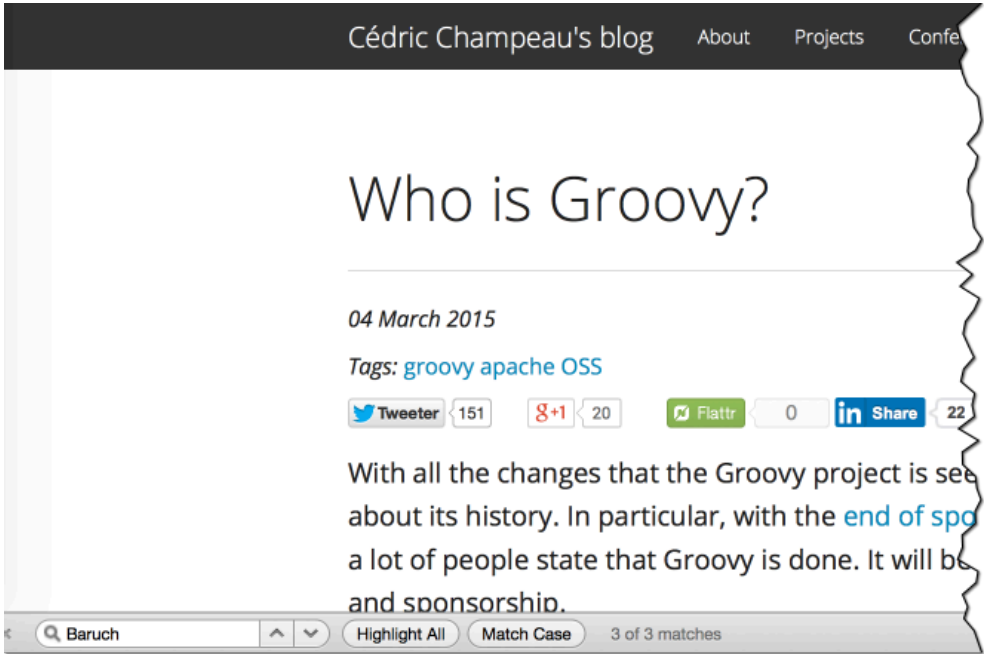
Bintray

artifactory

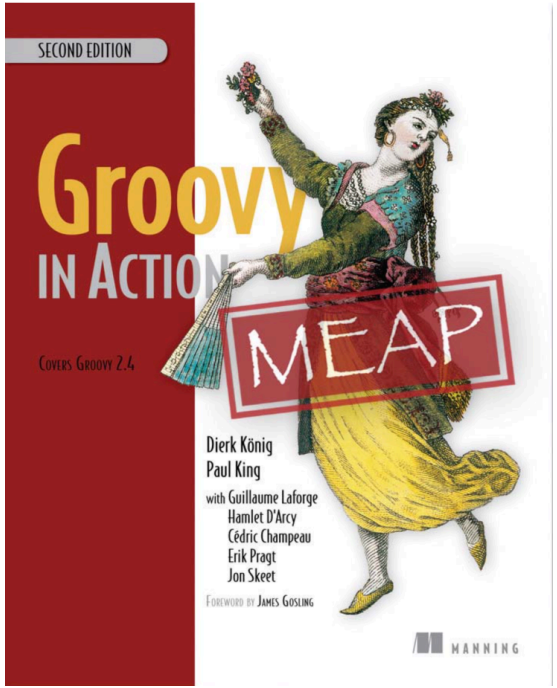
Why me and Groovy?

Bintray

Why me and Groovy?



Just go read this book



No, thank you!

OK, BUT THIS TALK IS STILL TL;DR OF IT.

Teasers and essentials

THE FEEL OF GROOVY

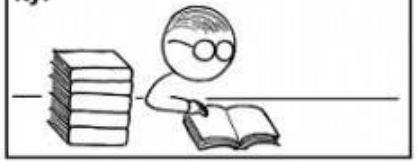
Days 1 - 10
Teach yourself variables, constants, arrays, strings, expressions, statements, functions,...

Days 11 - 21
Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism, ...

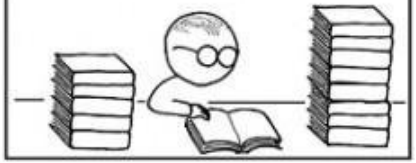
Days 22 - 697
Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.

Days 698 - 3648
Interact with other programmers. Work on programming projects together. Learn from them.

Days 3649 - 7781
Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.

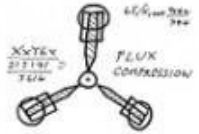


Days 7782 - 14611
Teach yourself biochemistry, molecular biology, genetics,...



Day 14611
Use knowledge of biology to make an age-reversing potion.

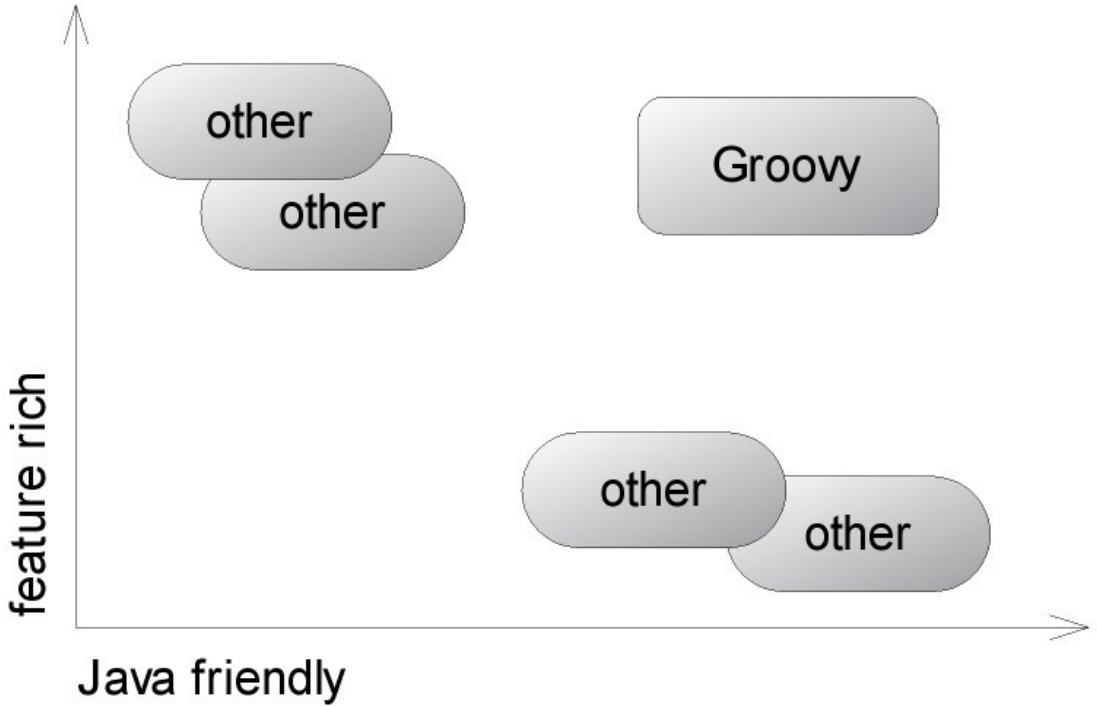
Day 14611
Use knowledge of physics to build flux capacitor and go back in time to day 21.



Day 21
Replace younger self.

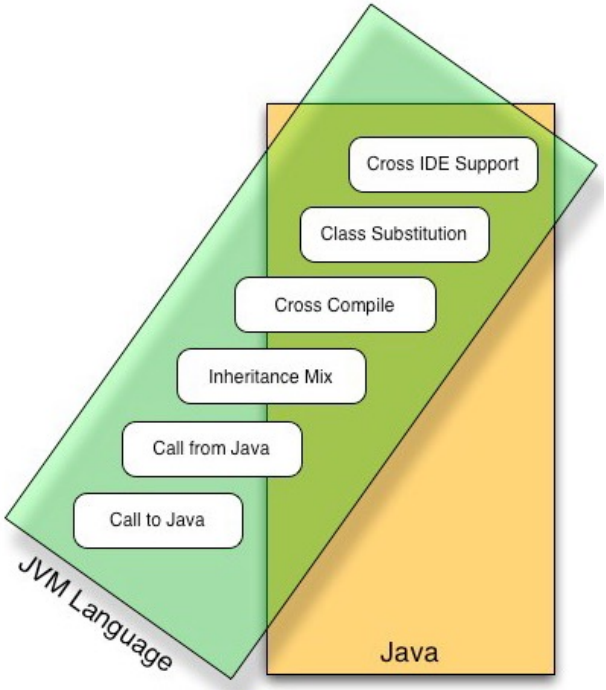
As far as I know, this is the easiest way to "Teach Yourself C++ in 21 Days".

Not with Groovy



Java and Groovy – bros forever

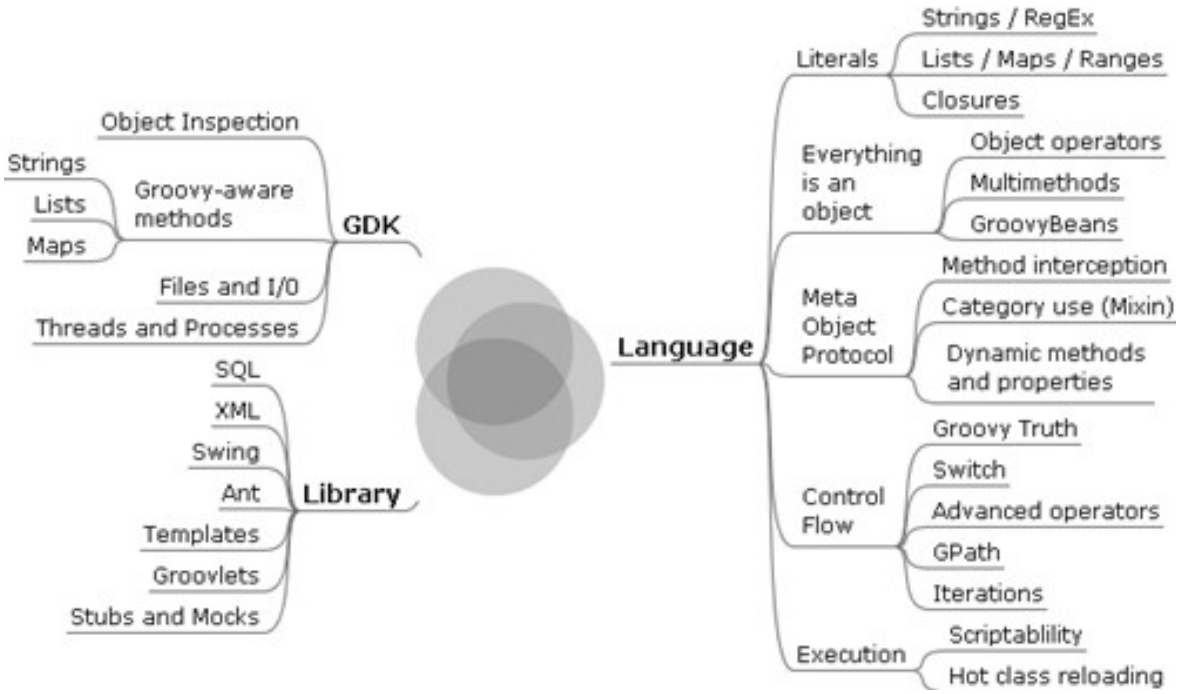
Powerful as Scala, Java-friendly as… Java



Similar syntax

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

today = new Date() // Groovy

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


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

(import '(java.util Date)) ; Clojure
        (def today (new Date)) ; Clojure
        (def today (Date.)) ; Clojure alternative
```

Much more powerful



Teasers: Closures and IO

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

Output:

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

Teasers: Collections 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)
```

XML gotta hurt. Or not.

Given:

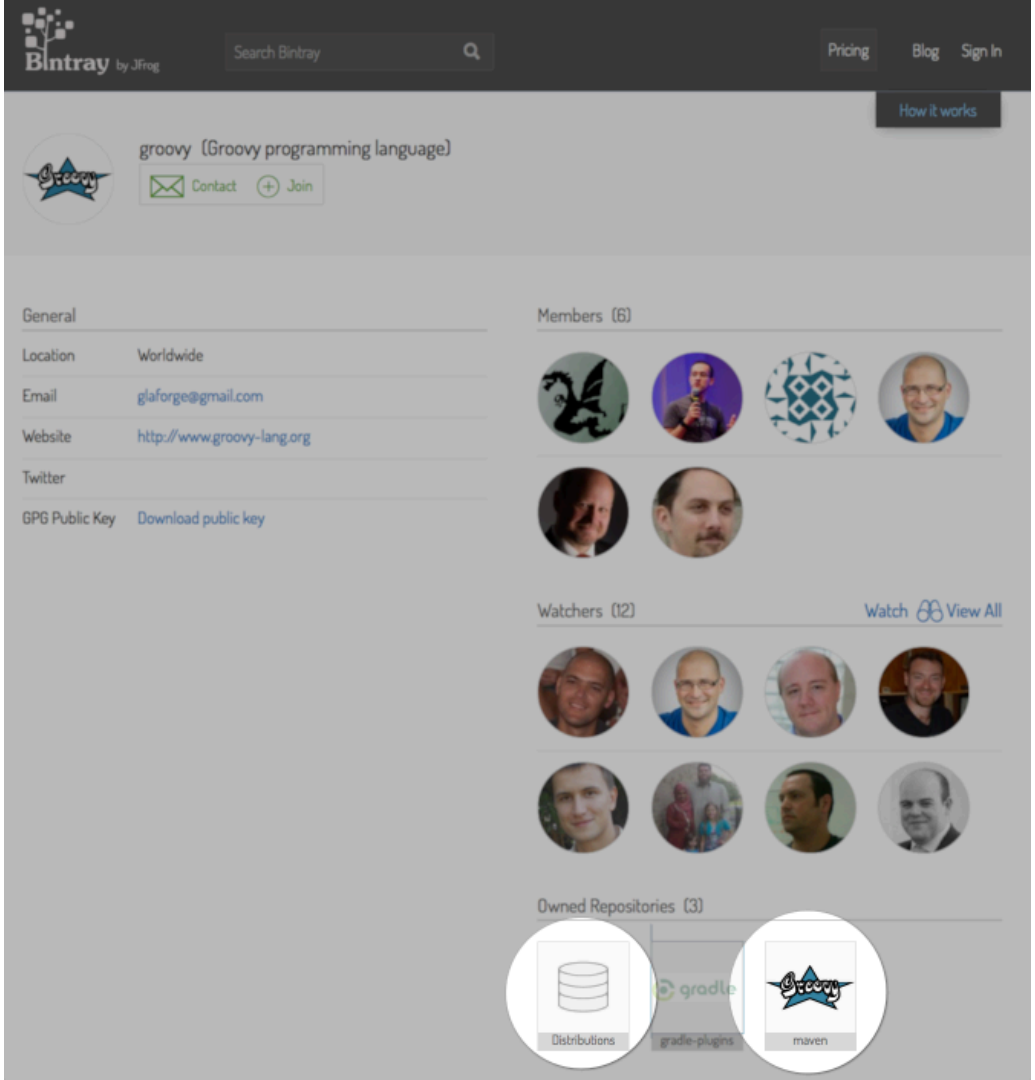
```
<?xml version="1.0" ?>
<customers>
    <corporate>
        <customer name="Satya Nadella" company="Microsoft" />
        <customer name="Tim Cook" company="Apple" />
        <customer name="Larry Ellison" company="Oracle" />
    </corporate>
    <consumer>
        <customer name="John Doe" />
  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

XML gotta hurt. Or not.

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

Output:

```
Satya Nadella works for Microsoft
Tim Cook works for Apple
Larry Ellison works for Oracle
```


Bintray by JFrog
Search Bintray
Pricing
Blog
Sign In
How it works
groovy (Groovy programming language)
Contact
Join
General
Location
Worldwide
Email
glaforge@gmail.com
Website
http://www.groovy-lang.org
Twitter
GPG Public Key
Download public key
Members (6)
Watchers (12)
Watch
View All
Owned Repositories (3)
Distributions
gradle
gradle-plugins
maven

Groovy executables in dist

```
jbaruch@smallfry ~/.gvm/groovy/current/bin » ls
grape                  groovydoc
grape.bat              groovydoc.bat
groovy                 groovysh
groovy.bat             groovysh.bat
groovyConsole          java2groovy
groovyConsole.bat      java2groovy.bat
groovyc                startGroovy
groovyc.bat            startGroovy.bat
```

IDE support

IDE support

IDE support

IDE support

How the code looks like, with a dynamic touch

THE BASICS

Looks like Java. Almost.

- Anonymous inner classes (who needs them anyway)
- Multiple variables initialization in for loop (when did you use them last time?)
- Array creation with initialization (supported differently)
- Java 8 syntax (not sure how lambda expressions should be treated)
- Checked exceptions (we hate them too)
- Try with resources (usually not needed)

Get rid of the boilerplate

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
- Sometimes dots before method calls
- Type declarations
- Type casts
- throws

Improvements over Java

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

Output:

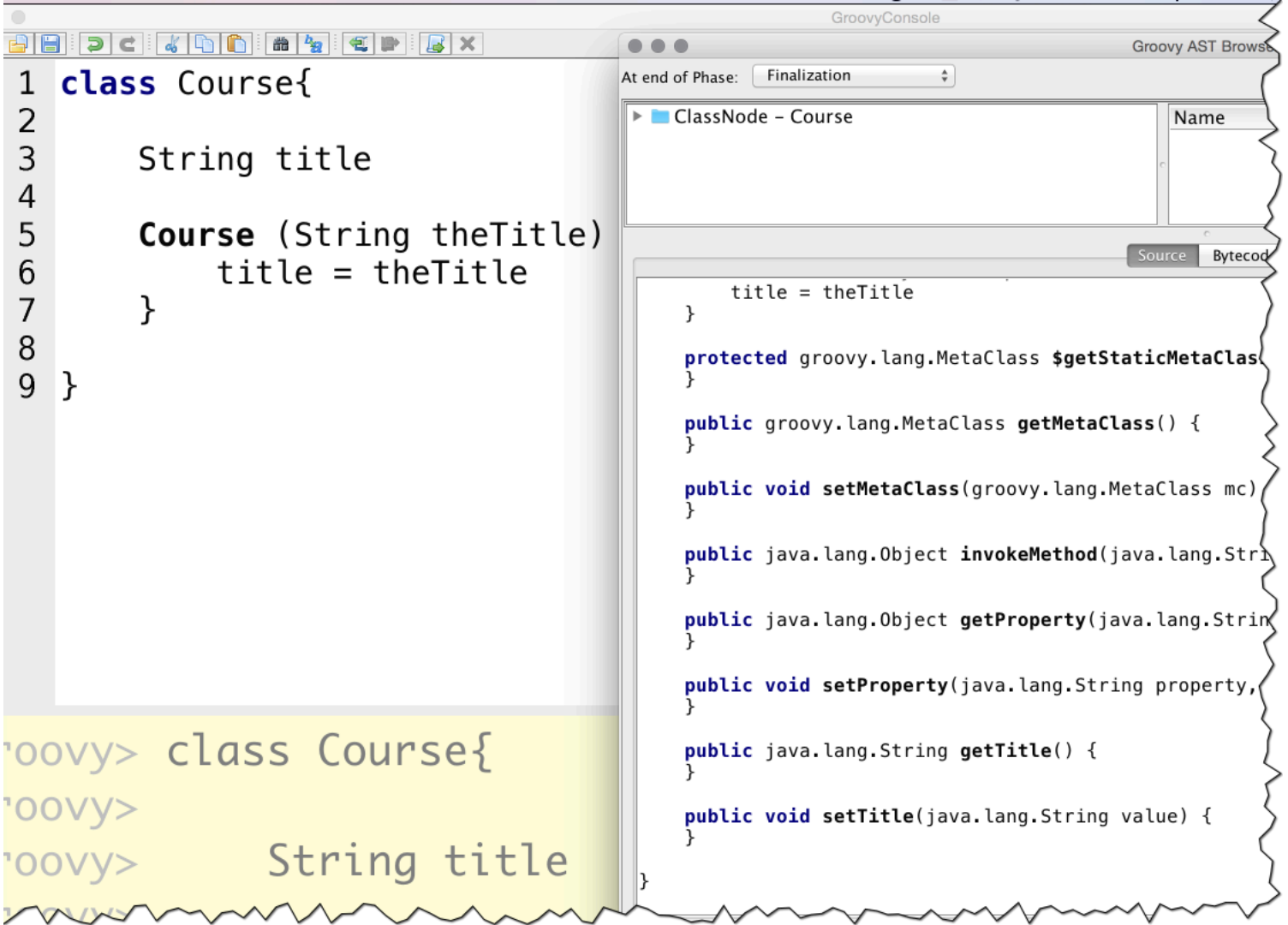
```
Assertion failed:
assert b == a + a
       |  | | | |
       9  | 5 | 5
          |   10
        false
```

Classes

```
class Course{

    private String title

    Course (String theTitle) {
        title = theTitle
    }

    String getTitle(){
        return title
    }
}
```

Properties

```
class Course{

    String title

    Course (String theTitle) {
        title = theTitle
    }

}
```


GroovyConsole
1 class Course{
2
3 String title
4
5 Course (String theTitle)
6 title = theTitle
7 }
8
9 }
roovy> class Course{
roovy>
roovy> String title
Groovy AST Browse
At end of Phase: Finalization
ClassNode - Course
Name
Source
Bytecod
title = theTitle
}
protected groovy.lang.MetaClass $getStaticMetaClas
}
public groovy.lang.MetaClass getMetaClass() {
}
public void setMetaClass(groovy.lang.MetaClass mc)
}
public java.lang.Object invokeMethod(java.lang.Stri
}
public java.lang.Object getProperty(java.lang.Strin
}
public void setProperty(java.lang.String property,
}
public java.lang.String getTitle() {
}
public void setTitle(java.lang.String value) {
}
}

Scripts

- Creates a class
- Wraps top level script code in a main method
- Existing methods become members
- Existing classes used outside of the script

```groovy
class Course {
    String title
}
```

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

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

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

Annotations

- Same concept, much more powerful usage
- Can appear at unexpected places

AST Transformations

```groovy
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'"
}
```

AST Transformations

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

AST Transformations

- @Delegate
- @Singleton
- @Immutable
- Loggers
  - @Log
  - @Log4j
  - @Slf4j
  - @Commons

AST Transformations

- @Synchronized
- @WithReadLock
- @WithWriteLock

- Or just use GPars

AST Transformations

- @AutoClone
  - Classic
  - Copy constructor
  - Serialization
- @AutoExternalize

- @PackageScope
- @Category
- @Mixin

Grapes: poor man's dependency manager

```
@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 (SFW!)

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

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

Great support for regular expressions

- No, I won’t show any examples here, we only have 75 minutes.

More object-oriented than Java

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

How that happened?

1. Everything is an object
2. Operators are aliases for methods

Method aliases

Operator	Method	Already overloaded in
a + b	a.plus(b)	Number, String, StringBuffer, Collection, Map, Date, Duration
a[b]	a.getAt(b)	Object, List, Map, CharSequence, Matcher, and much more
a << b	a.leftShift(b)	Usually used as ‘append’ or ‘add’

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

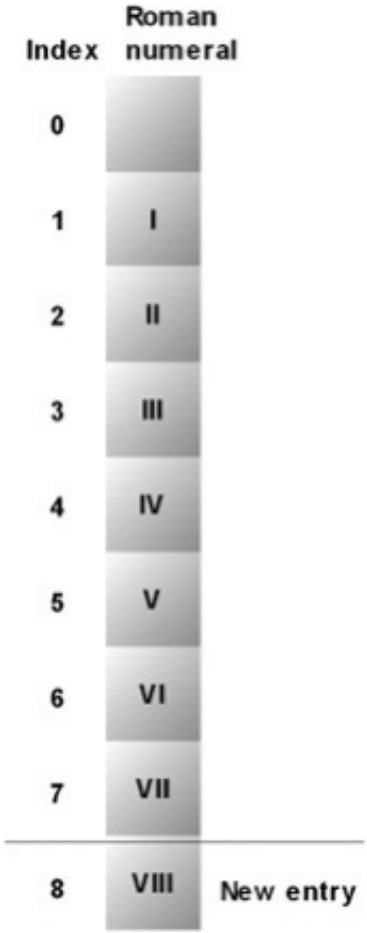
a == b	a.equals(b)	Yes, in Groovy == runs equals
a <=> b	a.compareTo(b)	java.lang.Comparable
a as type	a.asType(typeClass)	everywhere

Powerful as List and Map, comfortable as Array

DATA STRUCTURES IN GROOVY

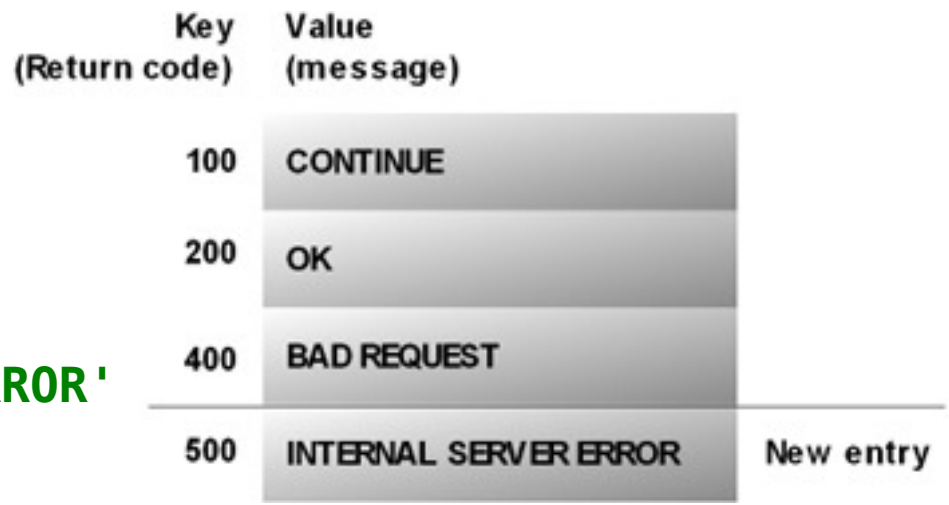
Lists

```
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
```



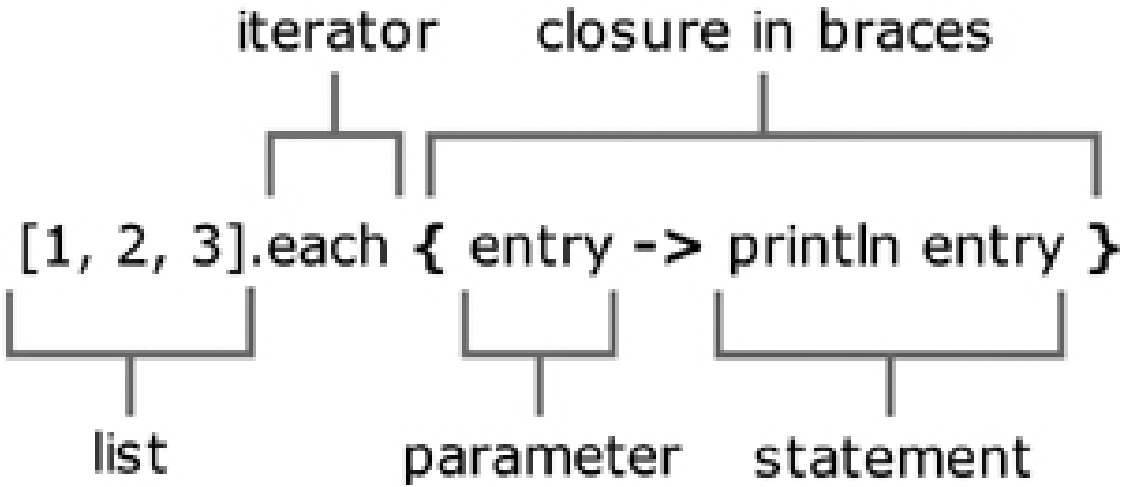
Ranges

```
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
```

Hell of a feature

CLOSURES

Closures


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

Why Closures?

- Reference to block of code: Handlers, listeners, callbacks
- Allows encapsulation of the control structures logic (WAT?)
- Access to the enclosing scope
- Not ugly

Anonymous inner classes, anyone?

- Pass the logic without knowing it in advance

```
JButton.addActionListener(ActionListene
r a)
```

- And then somewhere deep inside

```
a.actionPerformed(e)
```

- Closures serve the same target (but better)

```
JButton.addActionListener(Closure c)
```

- And then somewhere deep inside

```
c.call(e)
```

But it’s almost the same!

```
jbutton.addActionListener(new ActionListener() {

    @Override
    void actionPerformed(ActionEvent e) {
        //can't do much except of playing with e

    }
});
```

VS.

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

What just happened?!

Let’s take it easy

- Closure is an object

```
Closure action = {println ‘groovy’}
```

- We can declare a method that receives a closure

```
JButton#addActionListener(Closure action)
```

- We can pass a closure as an argument for this method

```
jbutton.addActionListener(action)
```

- We can skip the reference

```
jbutton.addActionListener({println ‘groovy’})
```

- If closure is a last argiment it can be placed outside the parentheses

```
jbutton.addActionListener(){
   println ‘groovy’
}
```

- Parentheses are optional

```
jbutton.addActionListener{
   println ‘groovy’
}
```

Cheater, addActionListener doesn't accept Closure!

- Closure can be cast to any type with corresponsding method

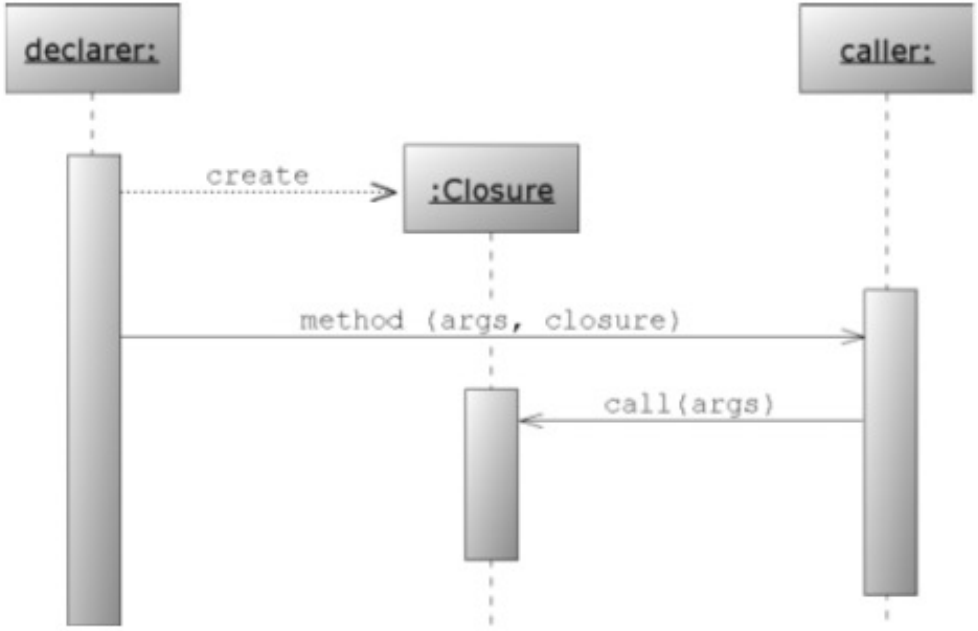
```
Closure c = {
  ActionEvent e -> prinltn e
} as ActionListener
```

Looks exactly like SAMs in Java 8!

Closures vs. Java 8 Lambdas

- Doesn’t have to backed up by SAM
- Can be cast to any interface (not only SAM)
- Can access any enclosing variables (not only final)
- No parentheses, no semicolons!

Making sense of closures – who calls what



Not your granddad’s ifs

CONTROL STATEMENTS

Control statements, BTW

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

```
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
}
```

Truth Power

Type	true
boolean	Well, true
Matcher	Match found
Collection, map	Not empty
String, GString	Not empty
Number	Not zero
Object reference	Not null
Everything	Whatever asBoolean method returns

Or not

DYNAMIC GROOVY

What “Dynamic” in Groovy means?

- If Groovy compiles to Java classes, how can it be dynamic?

1. “Dynamic” in a way “can be any type”
2. “Dynamic” in a way method dispatch is not finalized

Optional Typing

- Type declaration is optional
- def means “any type”, means “java.lang.Object”
- It behaves that way!
- Two approaches:

1. Declare only when you have to
2. Don’t declare only when you have to

When you have to omit the type?

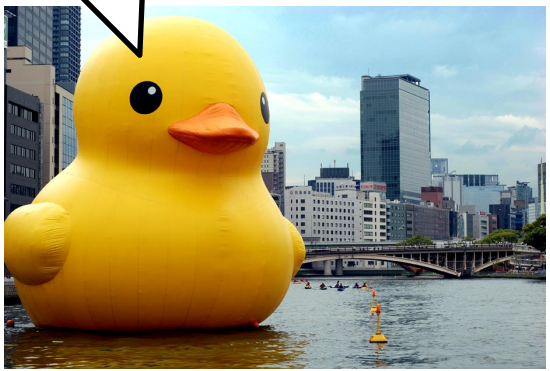
- Type encapsulation
- You actively don’t want to know
- Because you want to treat it like something else

```
List customers = new XmlSlurper().parse(new File('customers.xml'))
def customer = customers.first()
```

Treat like something else?

If I walk like a duck, swim like a duck, fly like a duck and quack like a duck, who cares I am an XML Node!

```
List customers = new XmlSlurper().parse(new File(          omers.xml'))
Customer customer = customers.first() as Customer
```



Dynamic method dispatch

- This achieved by passing all the calls via Meta Object Protocol (MOP)
- Decisions are taken in runtime

```
def customers = new XmlSlurper().parse(new File('customers.xml'))
Customer customer = customers.first() as Customer
customer.buy()
```

Keywords to google:

- MetaClass and ExpandoMetaClass
- methodMissing and propertyMissing
- getMethod and getProperty

Problems with Dynamic

1. Bad assignments not enforces at compile time
2. Absence of methods and properties can’t fail compilation (e.g. typos not being caught)

Turning the magic off

- Dynamic types can be turned off for better confidence
  - @TypeChecked
- Dynamic method dispatch can be turned off for better performance (no MOP == Java performance)
  - @CompileStatic

What Gives?!

- Still tons of syntactic sugar
- Use dynamic stuff when needed
  - Builders, Slurpers, Grails, all the magic
- Win-win!

Random things I love about Groovy

- Private scope is broken
  - Unfortunately targeted to be “fixed” in 3.0
- Methods can have default values
  - def print(String value, String encoding = ‘UTF-8’)
  - Generates overloaded methods in bytecode
- with{} gets in context of the object

Random things I love about Groovy

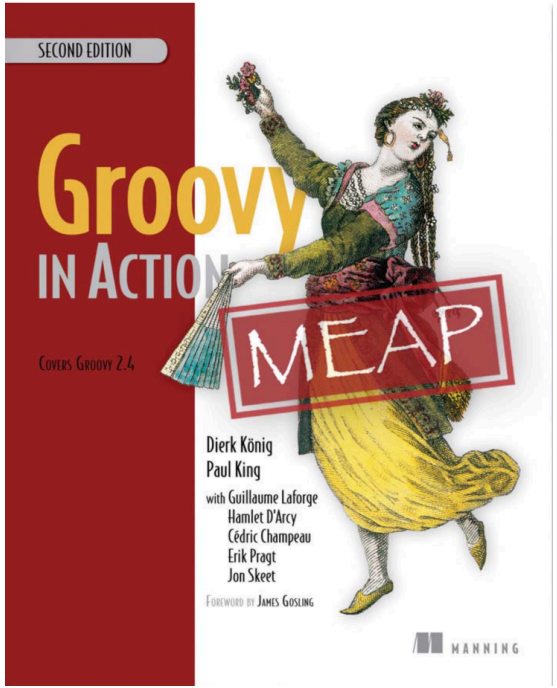
- Import alias
  - import com.somecompany.RobotDefaultImpl as Robot
- Refer to method with String
  - Removes some limitations of Java literals
  - Allows dynamic method calls:

```
def “${name} can name a method with spaces”(){ }
```

What we didn't cover ☹

- Everything in depth
- Builders and Slurpers
- Testing (Spock and Geb)
- Parallel programming with GPars (Hint: everything Scala has and more)
- DSLs (Hint: Groovy is perfect for DSLs)
- Writing AST transformations
- Other projects in the ecosystem (Gradle, Grails, Ratpack, Griffon, Grooscipt, Lazybones…)

Just go read this book



No, thank you!