comparing groovy & jruby *

** please check all knives, guns, pitchforks, and torches at the door*

NEAL FORD software architect / meme wrangler

ThoughtWorks

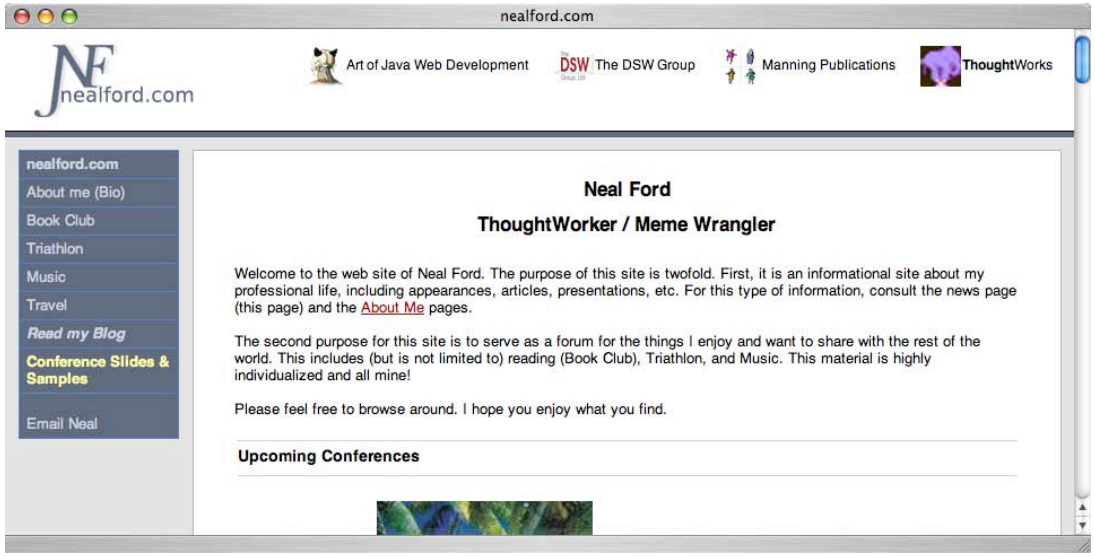
nford@thoughtworks.com
3003 Summit Boulevard, Atlanta, GA 30319
www.nealford.com
www.thoughtworks.com
blog: memeagora.blogspot.com
twitter: neal4d

housekeeping

ask questions at the end

(insane amount of material!)

download slides from nealford.com



download samples from `github.com/nealford`

agenda:

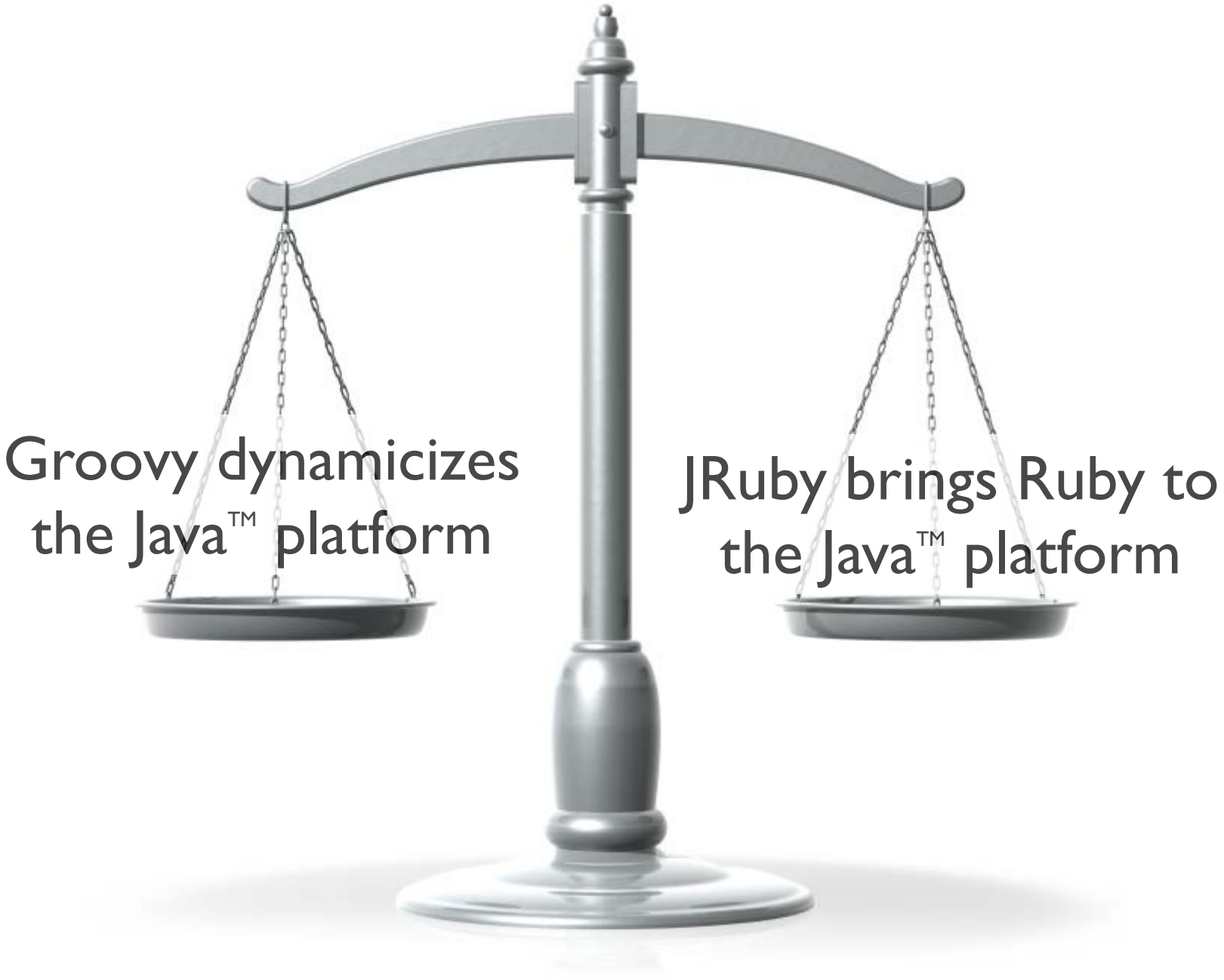
syntax

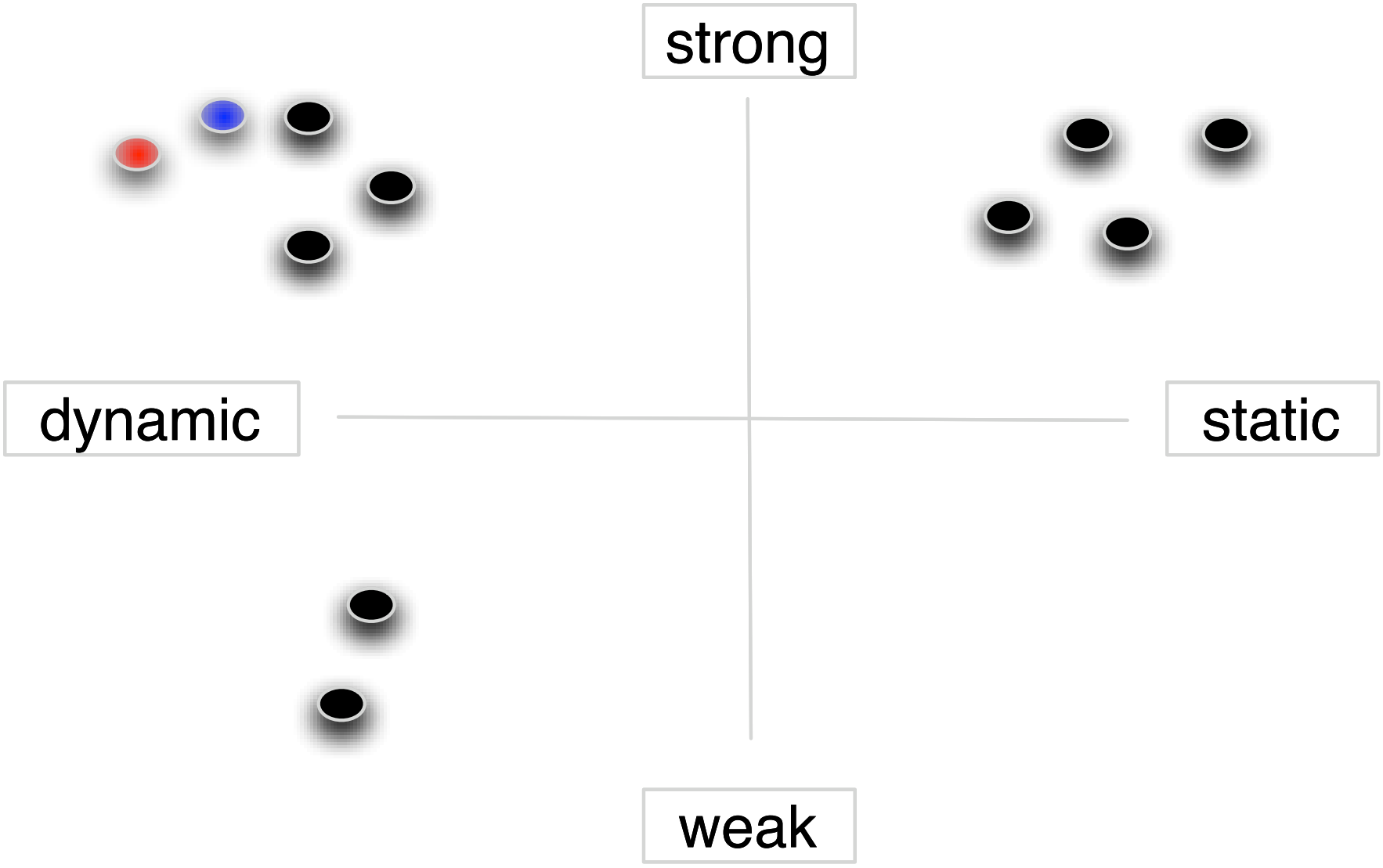
closures

eigenclass approaches

meta-programming

framework approach


Groovy dynamicizes the Java™ platform
JRuby brings Ruby to the Java™ platform


strong
dynamic
static
weak

differences & observations


syntax

sigils


```
this.name
```

vs

```
@name
```

```ruby
class Person
  attr_reader :name
  attr_accessor :age, :salary

  def initialize(name)
    @name = name
  end
end

p = Person.new("homer")
p.age = 45
p.salary = 100.00
puts "#{p.name} is #{p.age} and makes #{p.salary}"

p.instance_eval do
  puts "#{name} is #{age} and makes #{salary}"
end
```


```groovy
class Person {
 def name, age, salary
}

p = new Person(name:'Homer')
p.age = 45
p.salary = 100.00

println "${p.name} is ${p.age} and makes ${p.salary}"

p.identity {
 println "$name is $age and makes $salary"
}
```

auto-encapsulation?

```
class Person {
  def name;
  private salary;

  private getSalary() {
    salary
  }
}
```

```
def getName() {
  this.name
}
```

```
def setName(value) {
  this.name = value
}
```


```
void test_name_access() {
  def Person p = new Person();
  p.name = "James Tiberius"
  assertEquals("James Tiberius", p.name)
}
```

```
void test_that_auto_generated_methods_exist() {
  assertEquals Person.getDeclaredMethod("getName").name, "getName"
  assertEquals Person.getDeclaredMethod("setName", Object.class).name, "setName"
}
```

```
void test_that_you_must_create_your_own_methods_for_encapsulation() {
  def Person p = new Person(salary:1.0);
  shouldFail { Person.getDeclaredMethod("setSalary", BigDecimal.class) }
  assertEquals Person.getDeclaredMethod("getSalary").name, "getSalary"
  def m = Person.getDeclaredMethod("getSalary")
  m.accessible = true
  assertEquals m.invoke(p), 1.0
}
```

```
__FILE__ =~ /(.*)\/(.*)/

puts "path is #{$1} and filename is #{$2}"
```

Perl !!!!!!!!!

Ruby has some Perl-isms...

...with significant differences

```ruby
__FILE__ =~ /(.*)\/(.*)/
puts "path is #{$1} and filename is #{$2}"

rx = Regexp.new("(.*)\/(.*)")
m = rx.match(__FILE__)
puts "path is #{m[1]} and filename is #{m[2]}"
```

closures

```ruby
list.each { |p| puts p }
```

```ruby
list.each do |p|
    puts p if p == "red"
end
```


```groovy
list.each { p ->
    if (p == "red")
        println p
}
```


```groovy
def action = [
    { x, y -> x * y},
    { x, y -> x ** y}
]

action.each { a ->
    println a(2, 4)
}

println action[0].class
```

```
8
16
class closure_syntax$_run_closure1
```

```ruby
list_of_procs = [
  Proc.new { |x, y| x * y },
  Proc.new { |x, y| x ** y}
]

list_of_procs.each { |p| puts p.call(2, 4) }



list_of_procs2 = [
  proc { |x, y| x * y },
  proc { |x, y| x ** y}
]

list_of_procs.each { |p| puts p.call(6, 9) }
```

```ruby
list_of_lambdas = [
  lambda { |x, y| x * y },
  lambda { |x, y| x ** y}
]

list_of_lambdas.each { |l| puts l.call(3, 4)}
```

passing closures

```
def doSteps(step1, step2) {
  step1()
  step2()
}

doSteps({
  sum = 0
  1.upto(10) { n -> sum += n }
  println sum
}, {println "hello, world"})
```

auto-blocks

```
def safe_math x, y
   yield x, y if (x != 0 && y != 0)
end

puts safe_math(2, 4) {|x, y| x ** y}
```

passing closures

```
def do_steps(step1, step2)
  step1.call
  step2.call
end

do_steps proc {
  sum = 0
  1.upto(10) { |n| sum += n}
  puts sum
}, lambda {puts "hello, world"}
```

everything except...

in Java™, everything is an object...

...except primitives and nulls

in Groovy, everything is an object...

except null

in JRuby, *everything* is an object

`null` is null

`?.` handles protected reference

```
person?.address?.street
```

`nil` is an instance of

`NilClass`

```ruby
class String
  def blank?
    empty? || strip.empty?
  end
end
```

```ruby
def test_blank
  assert "".blank?
  assert " ".blank?
  assert nil.to_s.blank?
  assert ! "x".blank?
end
```


switch/case statement

```ruby
class Grade
  class << self
    def for_score_of(grade)
      case grade
        when 90..100: 'A'
        when 80..90 : 'B'
        when 70..80 : 'C'
        when 60..70 : 'D'
        when Integer: 'F'
        when /[A-D]/, /[F]/ : grade
        else raise "Not a grade: #{grade}"
      end
    end
  end
end
```

```groovy
class Grader {
  static def grade(score) {
    def grade
    switch(score) {
    case 90..100:
      grade = "A"
      break
    case 80..90:
      grade = "B"
      break
    case 70..80:
      grade = "C"
      break
    case 60..70:
      grade = "D"
      break
    case 0..60:
      grade = "F"
      break
    case 'A'..'D':
    case 'F':
      grade = score
      break
    case Integer:
      grade = "F"
      break
    default:
      throw new RuntimeException("Not a grade value")
    }
    grade
  }
}
```

```ruby
class Grade
  class << self
    def for_score_of(grade)
      case grade
        when 90..100: 'A'
        when 80..90 : 'B'
        when 70..80 : 'C'
        when 60..70 : 'D'
        when Integer: 'F'
        when /[A-D]/, /[F]/ : grade
        else raise "Not a grade: #{grade}"
      end
    end
  end
end
```

===

```groovy
class Grader {
  static def grade(score) {
    def grade
    switch(score) {
    case 90..100:
      grade = "A"
      break
    case 80..90:
      grade = "B"
      break
    case 70..80:
      grade = "C"
      break
    case 60..70:
      grade = "D"
      break
    case 0..60:
      grade = "F"
      break
    case 'A'..'D':
    case 'F':
      grade = score
      break
    case Integer:
      grade = "F"
      break
    default:
      throw new RuntimeException("Not a grade value")
    }
    grade
  }
}
```

isCase()

“spread” operator

```
def strings = []
strings << "one" << "two" << "three"

s1 = strings.collect { s -> s.toUpperCase() }


s2 = strings*.toUpperCase()
```

“elvis” operator

```groovy
def gender = user.male ? "male" : "female"
```

```groovy
def displayName = user.name ?: "Anonymous"
```

side effects of encapsulating Java™

things added to `Object`

Return Value	Method	Description
Boolean	any {closure}	returns `true` if the closure returns true for any item
List	collect {closure}	returns a list of all items that were returned from the closure
Collection	collect(Collection collection) {closure}	same as above, but adds each item to the given collection
void	each {closure}	simply executes the closure for each item
void	eachWithIndex {closure}	same as each{} except it passes two aruments: the item and the index
Boolean	every {closure}	returns `true` if the closure returns `true` for *all* items
Object	find {closure}	returns the first item that matches the closure expression
List	findAll {closure}	returns all items that match the closure expression
Integer	findIndexOf {closure}	returns the index of the first item that matched the given expression

```groovy
class TestStuffAddedToObject extends GroovyTestCase {
  def numbers = [ 5, 7, 9, 12 ]

  void test_any() {
    assert numbers.any { it % 2 == 0 }
    assertFalse numbers.any { it > 15 }
  }

  void test_every() {
    assert numbers.every { it > 4 }
    assertFalse numbers.every { it > 5 }
  }

  void test_findAll() {
    assert numbers.findAll { it in 6..10 } == [7,9]
    assert numbers.findAll { it in 15..20} == []
  }
}
```

really? really!

```
void test_some_of_the_wacky_stuff_added_to_Object() {
  def Person p = new Person(name:"Kirk")
  assertFalse p.any { name == "Picard"}
  assertFalse p.any { name == "Kirk"}
}
```

JRuby additions

JRuby adds artifacts to “Rubify” Java™

```
class TestJavaAdditions < Test::Unit::TestCase
  def test_additions_to_ArrayList()
    a = java.util.ArrayList.new
    a << 2 << 4 << 6 << 9 << 14
    assert a.any? {|i| i % 2 != 0 }
    assert a.find { |i| i.between?(a[2], a[4]) }
  end
```

ArrayList

Fixnum

...but doesn’t add collection stuff to Object

Groovy is optionally typed

```
import java.util.*;

class CollectionFactory {
    def List getCollection(description) {
        if (description == "Array-like")
            return new ArrayList()
        else if (description == "Stack-like")
            return new Stack()
    }
}
```

```groovy
@Test void get_an_array() {
    def l = f.getCollection("Array-like")
    assertTrue l instanceof java.util.ArrayList
    l.add("foo")
    assertThat l.get(0), is("foo")
    l.removeAll(l)
    assertThat l.size(), is(0)
}

@Test void get_a_stack() {
    def s = f.getCollection("Stack-like")
    assertTrue s instanceof java.util.Stack
    s.add("foo")
    assertThat s.get(0), is("foo")
    s.removeAll(s)
    assertThat s.size(), is(0)
    s.push("bar")
    assertThat s.size(), is(1)
    def r = s.pop()
    assertThat r, is("bar")
    assertThat s.size(), is(0)
}
```


```groovy
@Test
void test_search() {
    List l = f.getCollection("Stack-like")
    assertTrue l instanceof java.util.Stack
    l.push("foo")
    assertThat l.size(), is(1)
    def r = l.search("foo")
}
```

search exists on `Stack`

```groovy
@Test(expected=groovy.lang.MissingMethodException.class)
void verify_that_typing_does_not_help() {
    List l = f.getCollection("Array-like")
    assertTrue l instanceof java.util.ArrayList
    l.add("foo")
    assertThat l.size(), is(1)
    def r = l.search("foo")
}
```

but not on `ArrayList`

or `List`!

interfaces & mixins

Groovy & interfaces

allows you to interact with Java™ interfaces

can wrap proxies around them if needed

"fake out" interfaces via closures using **as**


```groovy
new Thread(
  {println "running"} as Runnable
).start()


interface X {
  void f()
  void g(int n)
  void h(String s, int n)
}

x = {Object[] args -> println "method called with $args"} as X
x.f()
x.g(1)
x.h("hello",2)
```


```groovy
impl = [
  i: 10,
  hasNext: { impl.i > 0 },
  next: { impl.i-- },
]

iter = impl as Iterator
while (iter.hasNext()) {
  println iter.next()
}
```

be careful when implementing with a map:
`NullPointerException` if you forget a method

Ruby has no interfaces

mixin

first appeared in symbolic logic's *flavors* system

named after ice cream mix-ins

not a form of specialization

a way to collect functionality

```ruby
module Debug
  def who_am_i
    "#{self.class.name} (\##{self.object_id}): #{self.to_s}"
  end
end


class Person
  include Debug
end

class Employee < Person
end

homer = Person.new
monty = Employee.new

puts homer.who_am_i
puts monty.who_am_i
```

comparisons

```
class Employee
  include Comparable

  def <=>(other)
      name <=> other.name
  end
end

list = Array.new
list << Employee.new("Monty", 10000)
list << Employee.new("Homer", 50000)
list << Employee.new("Bart", 5000)
```

comparisons

```
puts list

list.sort!

puts list

puts "Monty vs. Homer", list[0] < list[1]
puts "Homer vs. Monty", list[0] > list[1]

puts "Homer is between Bart and Monty?",
    list[1].between?(list[0], list[2])
```


CONTRACT

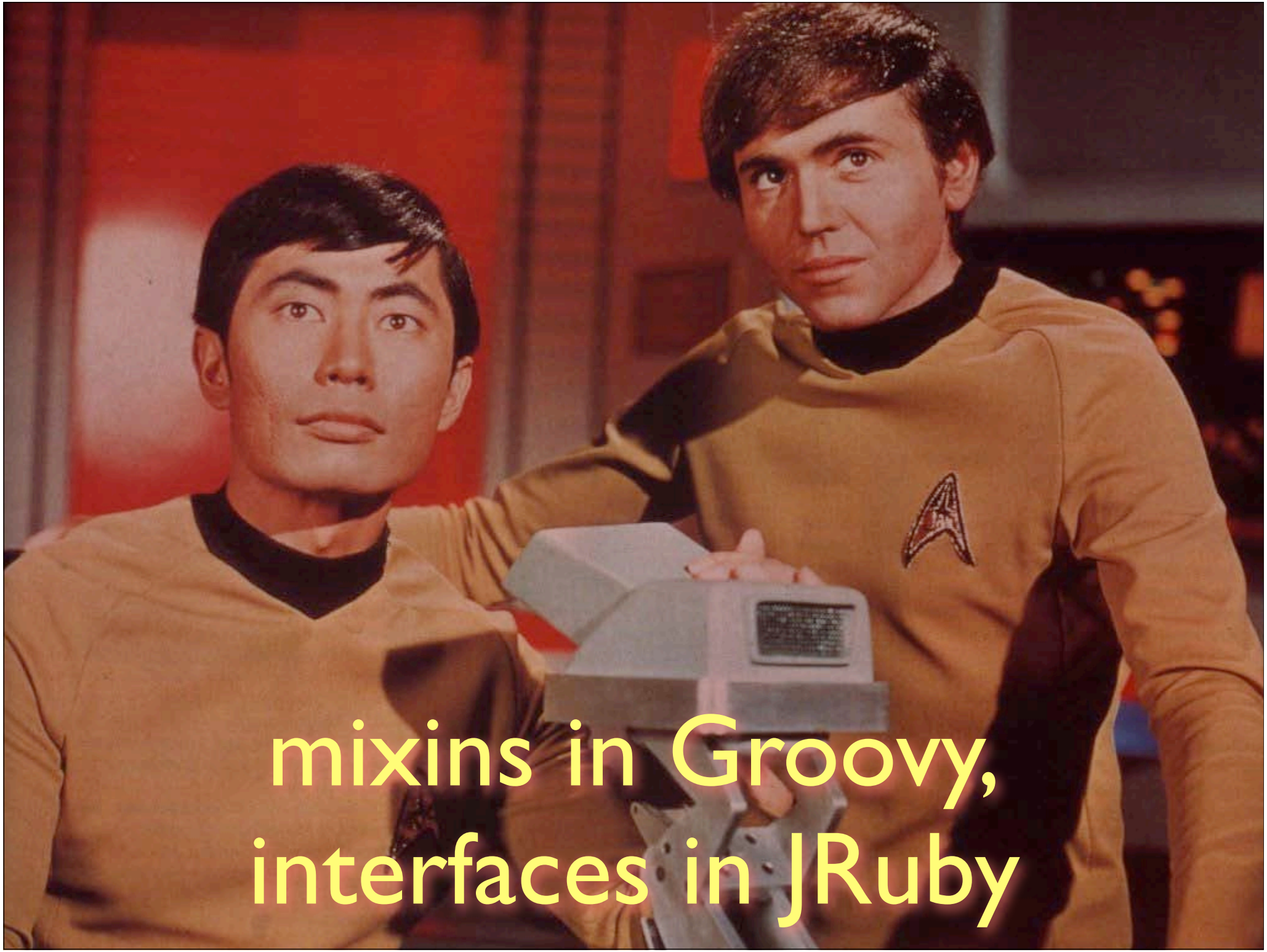
comparisons

```
[nealford| ~/docs/dev/ruby/conf_jruby/10.mixins ]=> ruby comparisons.rb
Name: Monty, salary: 10000, hire year: 2007
Name: Homer, salary: 50000, hire year: 2007
Name: Bart, salary: 5000, hire year: 2007
Name: Bart, salary: 5000, hire year: 2007
Name: Homer, salary: 50000, hire year: 2007
Name: Monty, salary: 10000, hire year: 2007
Monty vs. Homer
true
Homer vs. Monty
false
Homer is between Bart and Monty?
true
```

violating handshakes

```
Name: Monty, salary: 10000, hire year: 2007
Name: Homer, salary: 50000, hire year: 2007
Name: Bart, salary: 5000, hire year: 2007
comparisons.rb:19:in `sort!': undefined method `<=>' for #<Employee:0x27650
@hire_year=2007, @salary=10000, @name="Monty"> (NoMethodError)
        from comparisons.rb:19
```

```ruby
puts list

list.sort!

puts list

puts "Monty vs. Homer", list[0] < list[1]
puts "Homer vs. Monty", list[0] > list[1]

puts "Homer is between Bart and Monty?",
    list[1].between?(list[0], list[2])
```


mixins in Groovy,
interfaces in JRuby


```groovy
class ListUtils {
  static randomize(List list) {
    Collections.shuffle(list)
    list
  }
}
List.metaClass.mixin ListUtils

println "Random list = " + [1, 2, 3, 4, 5].randomize()
```

interfaces in JRuby?

```
module Iterator
  def initialize
    %w(hasNext next).each do |m|
      unless self.class.public_method_defined? m
        raise NoMethodError
      end
    end
  end
end
```

```ruby
class TestInterfaceDemo < Test::Unit::TestCase

  class Foo; include Iterator; end

  class Foo2; include Iterator; def hasNext; end; end

  class Foo3; include Iterator; def hasNext; end; def next; end
  end

  def test_methods_exist_when_imposed
    assert_raise(NoMethodError) {
      Foo.new
    }
  end

  def test_interface_imposition_fails_when_only_1_method_present
    assert_raise(NoMethodError) {
      Foo2.new
    }
  end

  def test_interface_works_when_interfaces_implemented
    f = Foo3.new
    assert f.class.public_method_defined? :hasNext
    assert f.class.public_method_defined? :next
  end

end
```


meta-programming

mostly parity (functionally)

very different implementations

executable declarations

much of Ruby's infrastructure is meta-programming

meta-programming methods:

```
attr_reader, attr_writer,
attr_accessor

include


private, protected, public
```

AST transformations

```
@WithLogging
def greet() {
  println "Hello World"
}
```

implementation

```
@Retention(RetentionPolicy.SOURCE)
@Target([ElementType.METHOD])
@GroovyASTTransformationClass(
    ["gep.LoggingASTTransformation"])
public @interface WithLogging {
}
```

```groovy
@GroovyASTTransformation(phase=CompilePhase.SEMANTIC_ANALYSIS)
public class LoggingASTTransformation implements ASTTransformation {

  public void visit(ASTNode[] nodes, SourceUnit sourceUnit) {
      List methods = sourceUnit.getAST()?.getMethods()
      // find all methods annotated with @WithLogging
      methods.findAll { MethodNode method ->
          method.getAnnotations(new ClassNode(WithLogging))
      }.each { MethodNode method ->
          Statement startMessage = createPrintlnAst("Starting $method.name")
          Statement endMessage = createPrintlnAst("Ending $method.name")

          List existingStatements = method.getCode().getStatements()
          existingStatements.add(0, startMessage)
          existingStatements.add(endMessage)
      }
  }

  private Statement createPrintlnAst(String message) {
      return new ExpressionStatement(
          new MethodCallExpression(
              new VariableExpression("this"),
              new ConstantExpression("println"),
              new ArgumentListExpression(
                  new ConstantExpression(message)
              )
          )
      )
  }
}
```

optional inclusion

```
class TestCalculator < Test::Unit::TestCase
  def test_complex_calculation
    assert_equal(4, Calculator.new.complex_calculation)
  end
end
```

conditional method

```
class TestCalculator < Test::Unit::TestCase

  if ENV['BUILD'] == 'ACCEPTANCE'
    def test_complex_calculation
       assert_equal(4, Calculator.new.complex_calculation)
    end
  end

end
```

attributes

```
class TestCalculator < Test::Unit::TestCase
  extend TestDirectives

  acceptance_only
  def test_complex_calculation
    assert_equal(4, Calculator.new.complex_calculation)
  end
end
```

hook methods

```
module TestDirectives
  def acceptance_only
    @acceptance_build = ENV['BUILD'] == 'ACCEPTANCE'
  end

  def method_added(method_name)
    remove_method(method_name) unless @acceptance_build
    @acceptance_build = false
  end
end
```

sticky attributes

```
module TestDirectives
  def acceptance_only
    @acceptance_build = ENV['BUILD'] == 'ACCEPTANCE'
  end

  def method_added(method_name)
    remove_method(method_name) unless @acceptance_build

  end
end
```

private, protected, public


eigenclass

adding methods via proxies

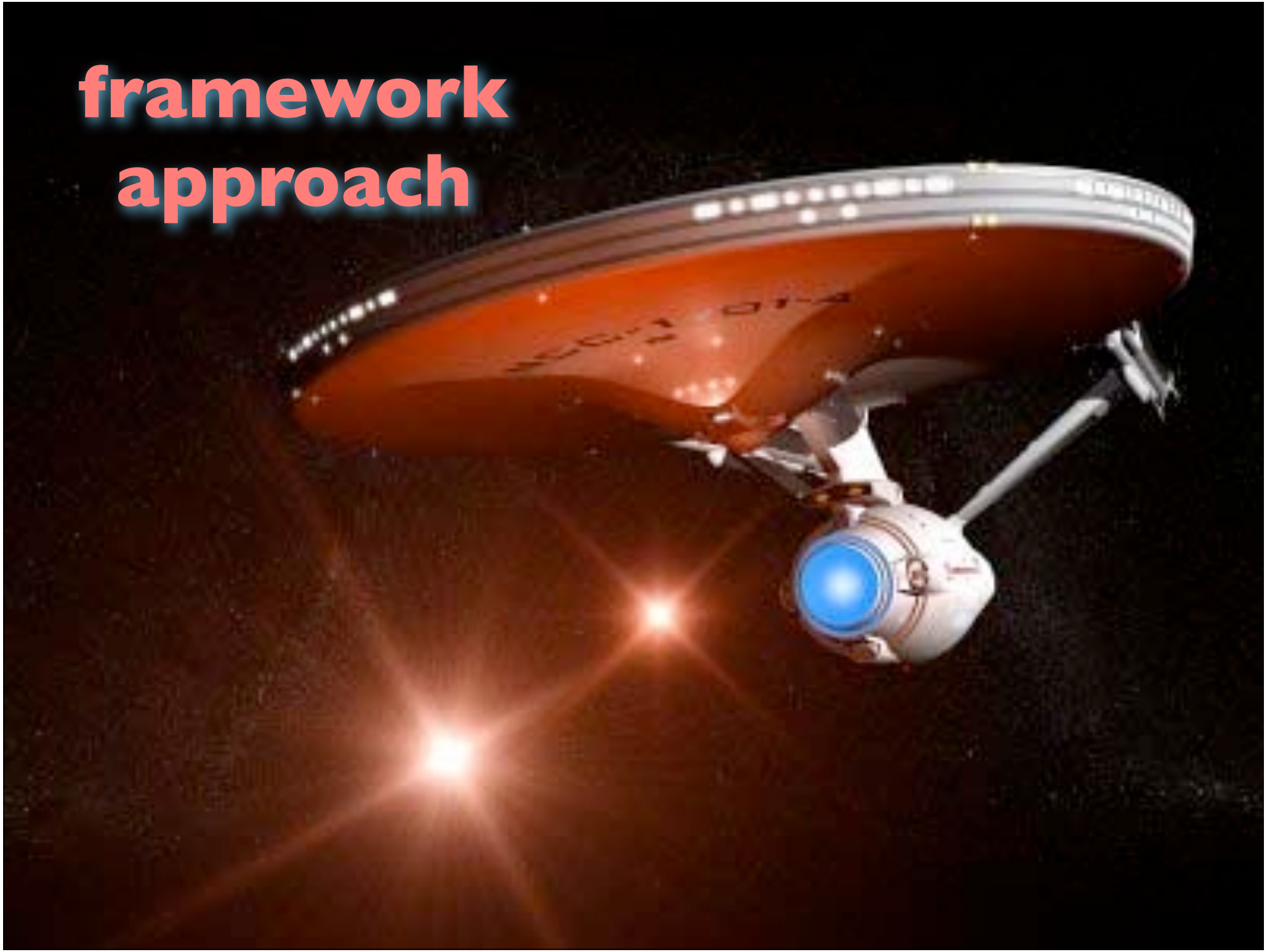
```
require 'java'

include_class 'java.util.ArrayList'

class ArrayList
   def first
     size == 0 ? nil : get(0)
   end
end


def test_added_first_method_is_present
  l = ArrayList.new
  l << 'red' << 'green' << 'blue'
  assert_equal('red', l.first)
end
```

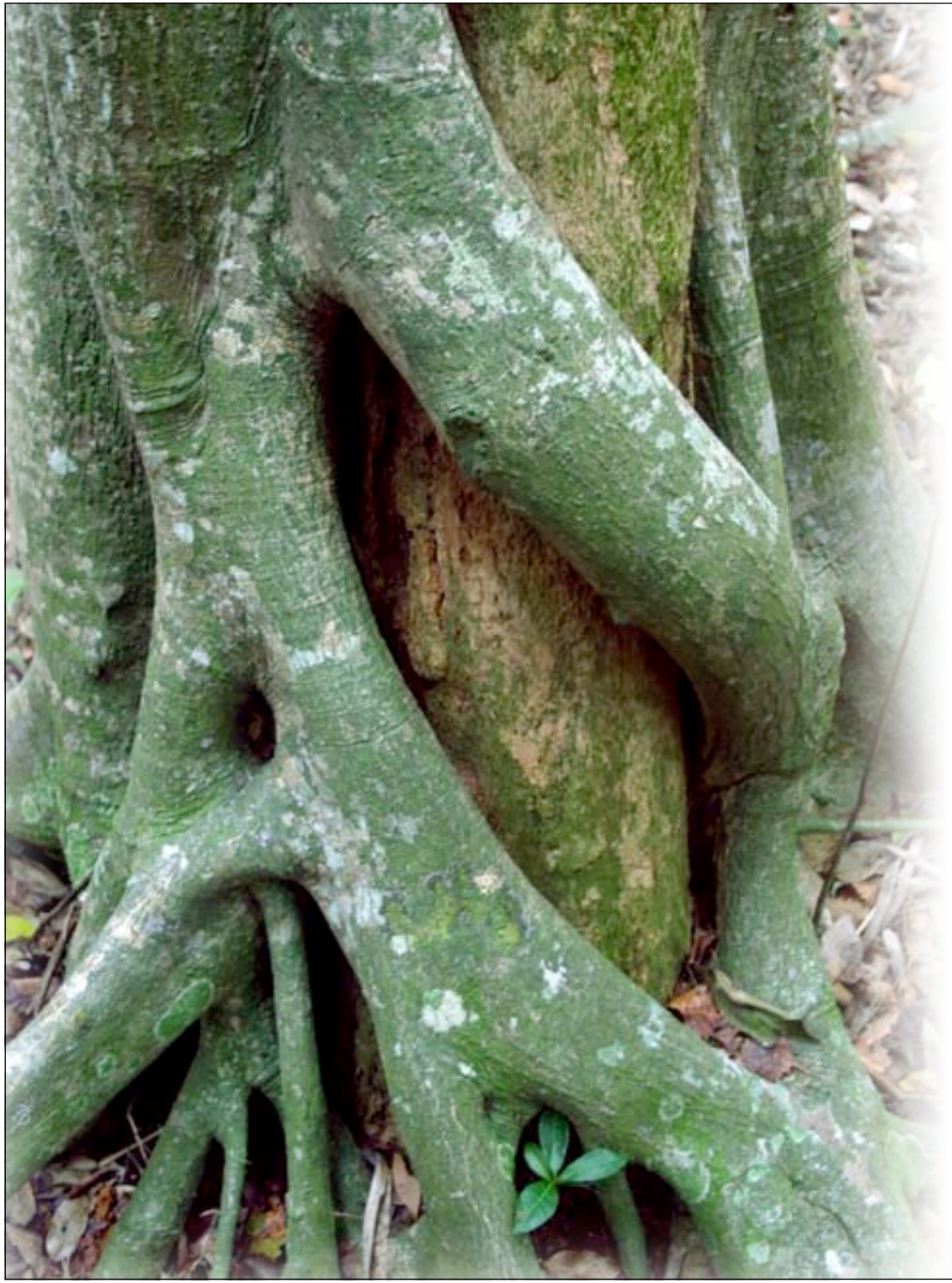
```ruby
def setup
  @l = ArrayList.new
  def @l.last
     size == 0 ? nil : get(size - 1)
  end
  @l << 'red' << 'green' << 'blue'
end

def test_adding_method_via_eigenclass
  assert_equal('blue', @l.last)
end

def test_that_last_only_exists_on_eigenclass
  l = ArrayList.new
  l << 'red' << 'green' << 'blue'
  assert @l.respond_to? :last
  assert ! (l.respond_to? :last)
  assert_raise(NoMethodError) {
    l.last
  }
end
```

Groovy's eigenclass

```
def list = new ArrayList()
list.metaClass.randomOrder { ->
    Collections.shuffle(delegate)
    delegate
}
list << 1 << 2 << 3 << 4
println list.randomOrder()
```

framework approach





encapsulate
& extend

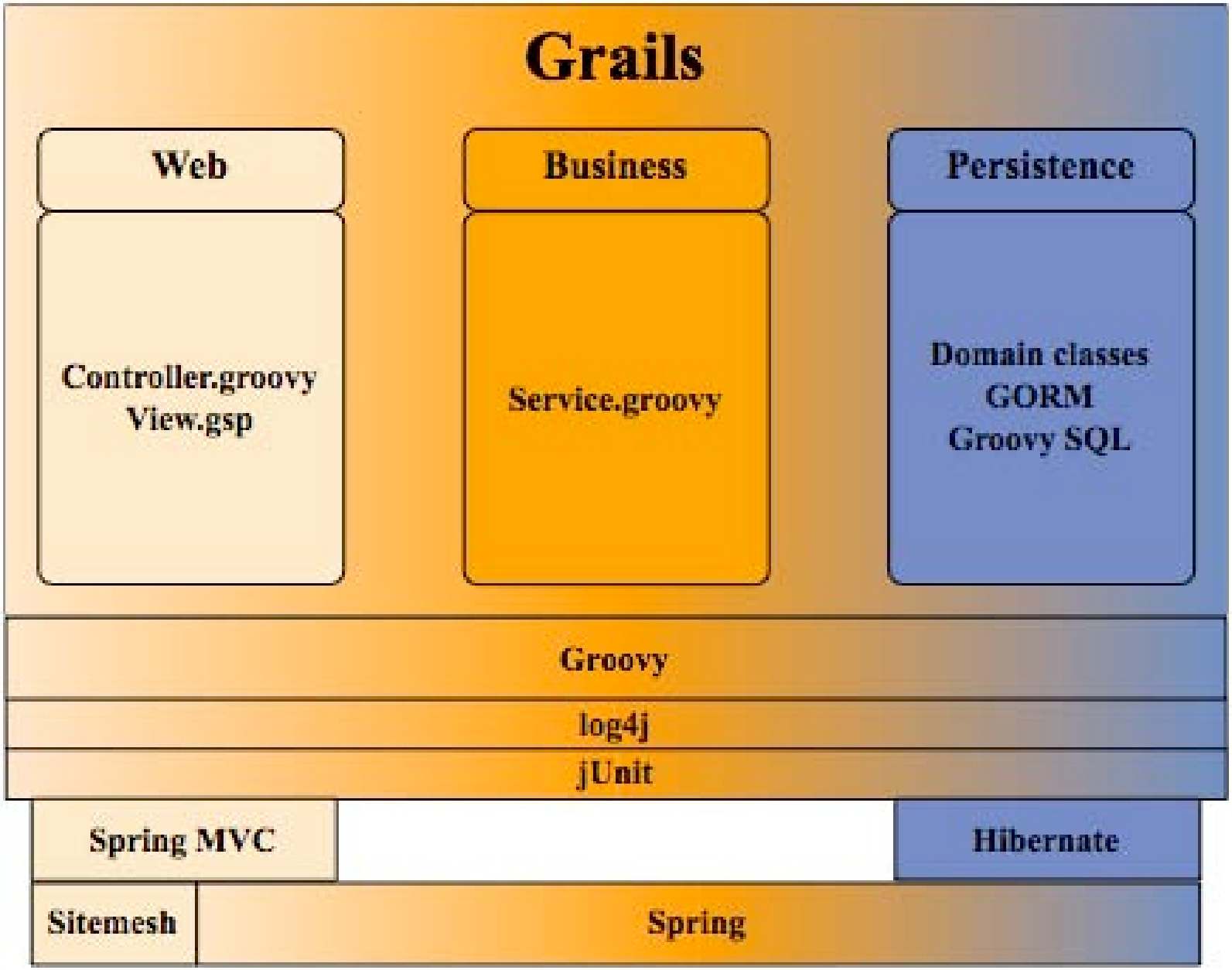
Groovy & Grails

leverage the existing industrial strength Java™ stack

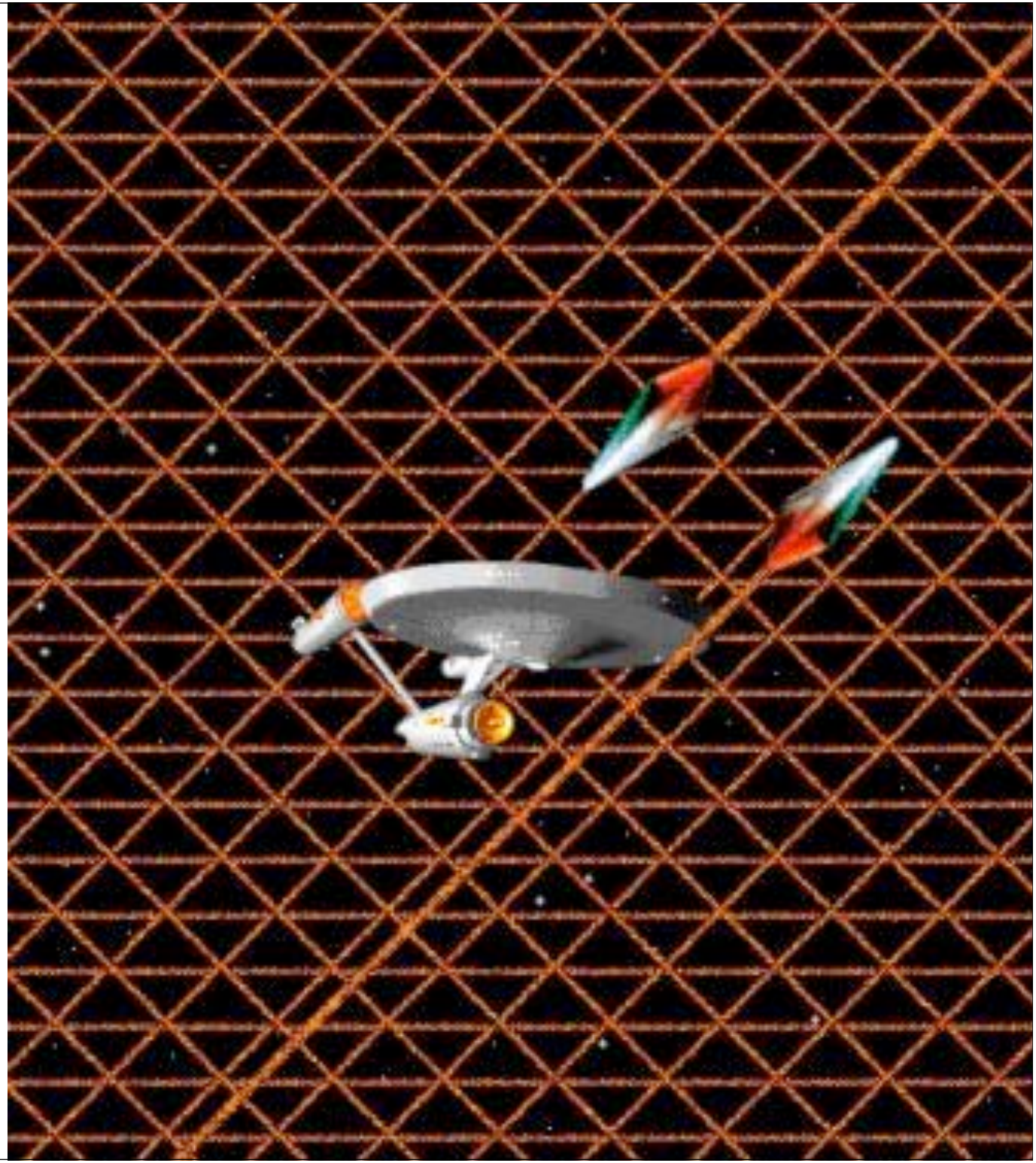
encapsulate best-of-breed frameworks

use Groovy to simplify and humanize them

encapsulate, then build up


Grails
Web
Business
Persistence
Controller.groovy
View.gsp
Service.groovy
Domain classes
GORM
Groovy SQL
Groovy
log4j
jUnit
Spring MVC
Hibernate
Sitemesh
Spring

abstraction
layers
via
meta || dsl



Ruby

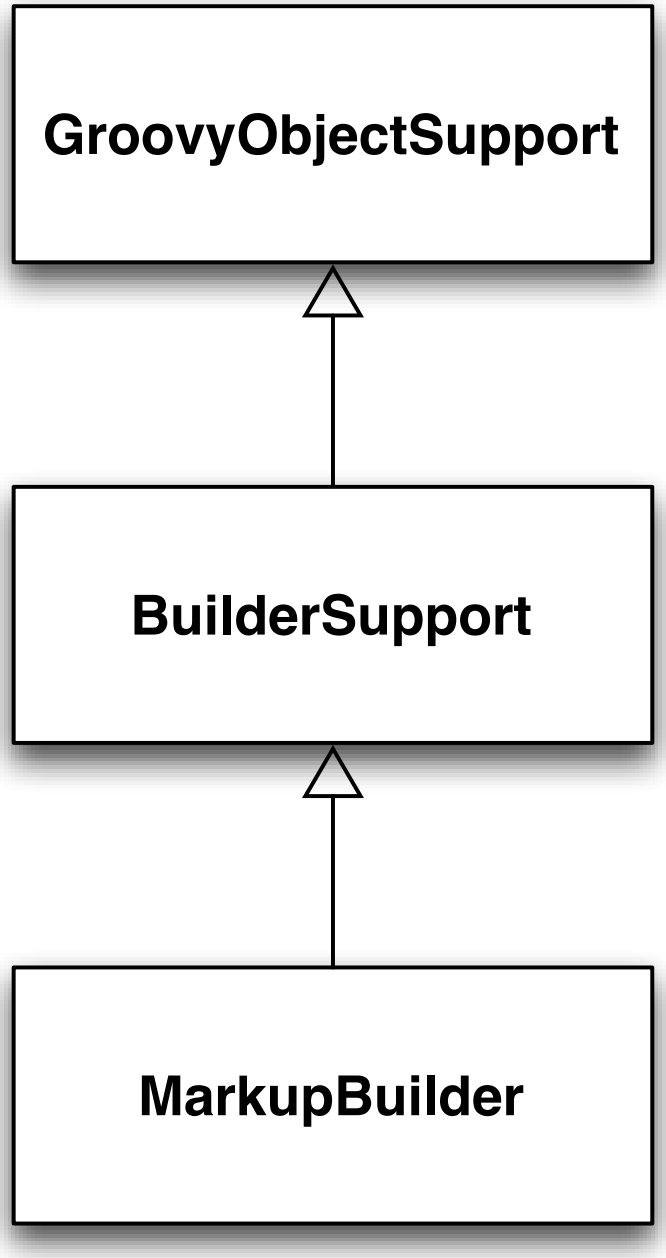
tends to build things as abstraction layers atop Ruby using meta-programming & dsl techniques

Ruby on Rails = software factories + dsl's

Ruby language is always 1 abstraction layer down

lightweight abstraction layers

case study: builders


GroovyObjectSupport
BuilderSupport
MarkupBuilder


JAVA


```groovy
public Object invokeMethod(String methodName, Object args) {
    Object name = getName(methodName);
    return doInvokeMethod(methodName, name, args);
}

protected Object doInvokeMethod(String methodName, Object name, Object args) {
    Object node = null;
    Closure closure = null;
    List list = InvokerHelper.asList(args);

    switch (list.size()) {
            case 0:
                    node = proxyBuilder.createNode(name);
                break;
                case 1:
                {
                        Object object = list.get(0);
                        if (object instanceof Map) {
                            node = proxyBuilder.createNode(name, (Map) object);
                        } else if (object instanceof Closure) {
                            closure = (Closure) object;
                            node = proxyBuilder.createNode(name);
                        } else {
```

builder

```ruby
require 'builder'

xml = Builder::XmlMarkup.new(:indent => 2)
xml.person {
  xml.name("Neo")
  xml.catch_phrase("Whoa")
}
puts xml.target!
```

builder works via method_missing

but what about this?

```ruby
require 'builder'

xml = Builder::XmlMarkup.new(:indent => 2)
xml.person {
  xml.name("Neo")
  xml.catch_phrase("Whoa")
  xml.class("pod-born")
}
puts xml.target!
```

```
simple_builder.rb:32: in 'class': wrong number of arguments (1 for 0) (ArgumentError)
   from simple_builder.rb:32
   from simple_builder.rb:14: in 'method_missing'
   from simple_builder.rb:25
```

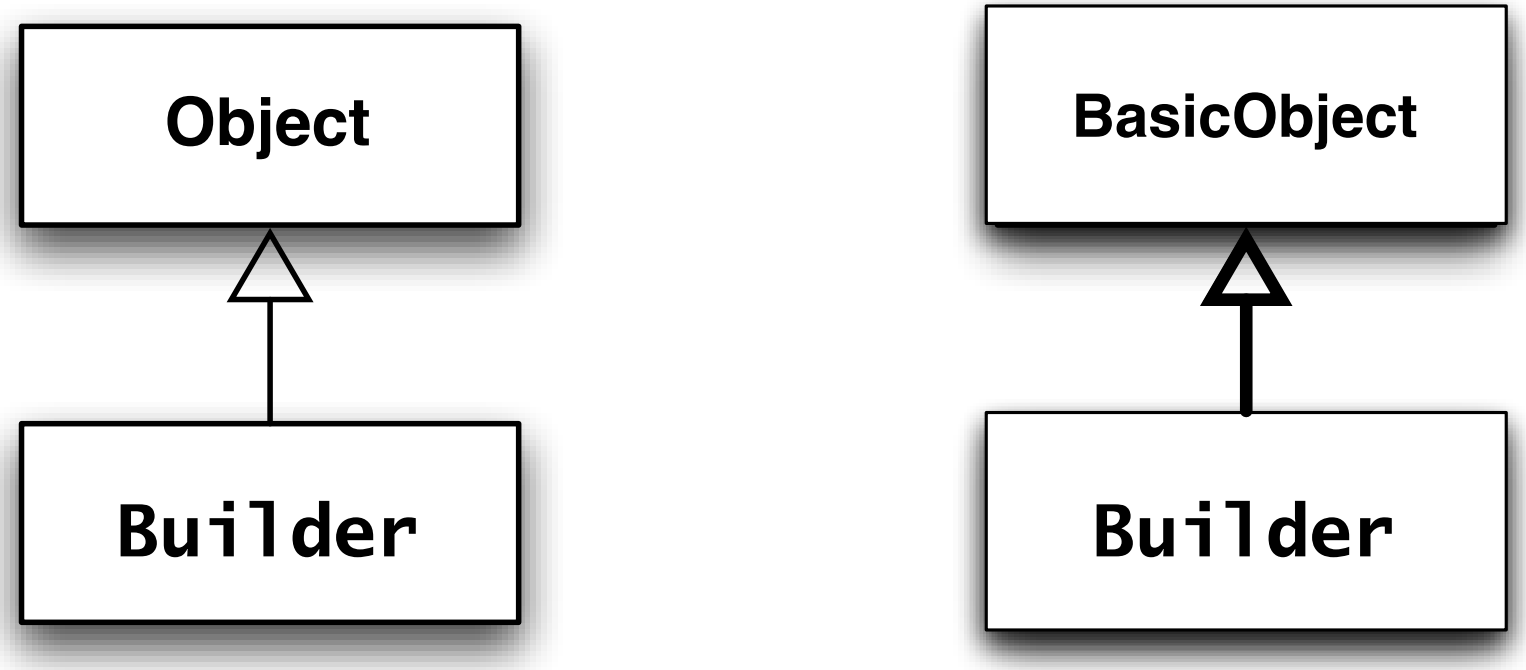
the problem:

how do you inherit from **`object`**...

...with inheriting from **`object`**?

1.8

Jim Weirich's
BlankSlate



1.9

BasicObject

```
class XmlBuilder
  def method_missing(sym, *args, &block)
    ...
  end
end
class XmlBuilder < BlankSlate
  def method_missing(sym, *args, &block)
    ...
  end
end
```

different philosophies

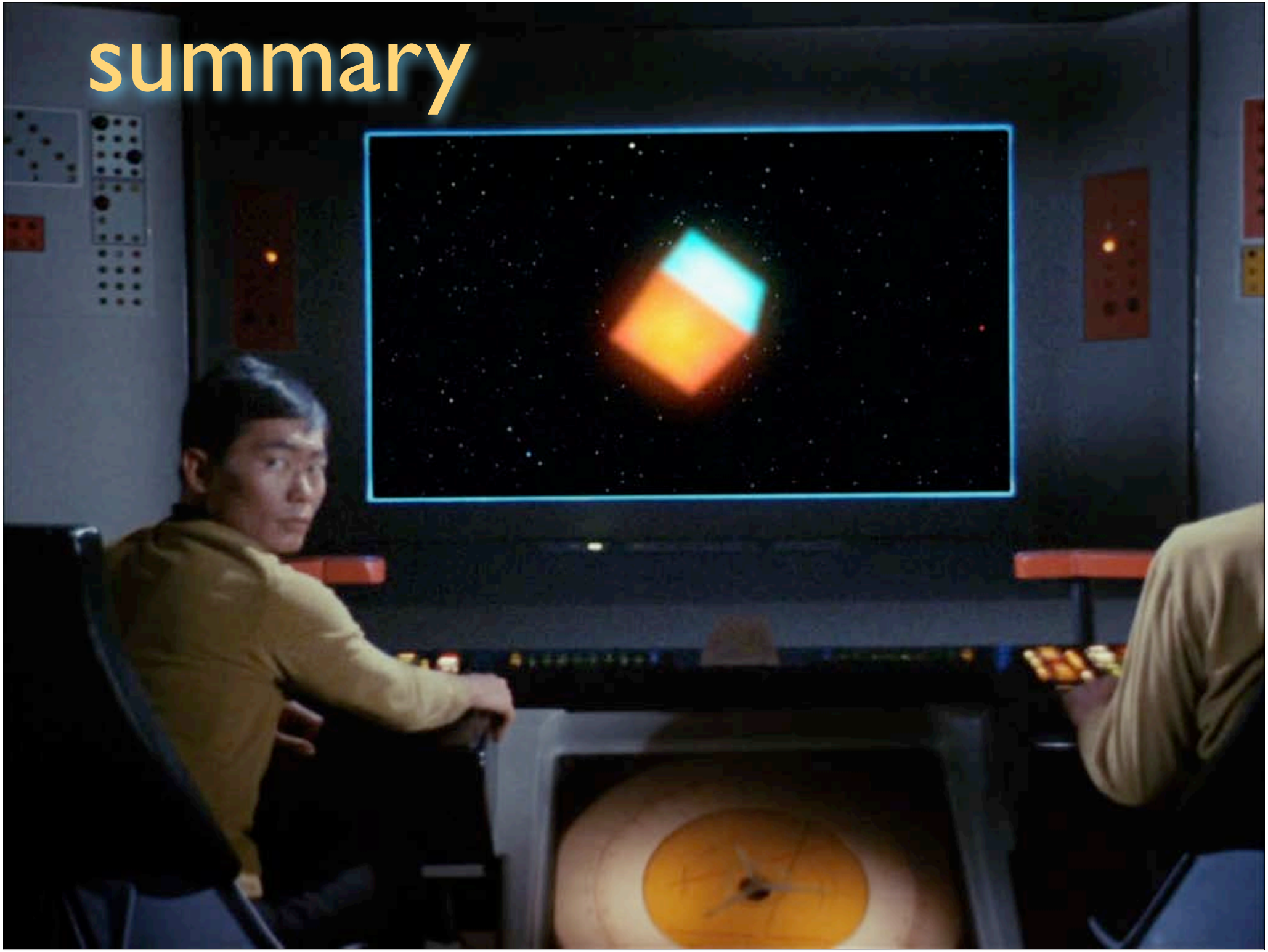
Groovy encapsulates and builds up

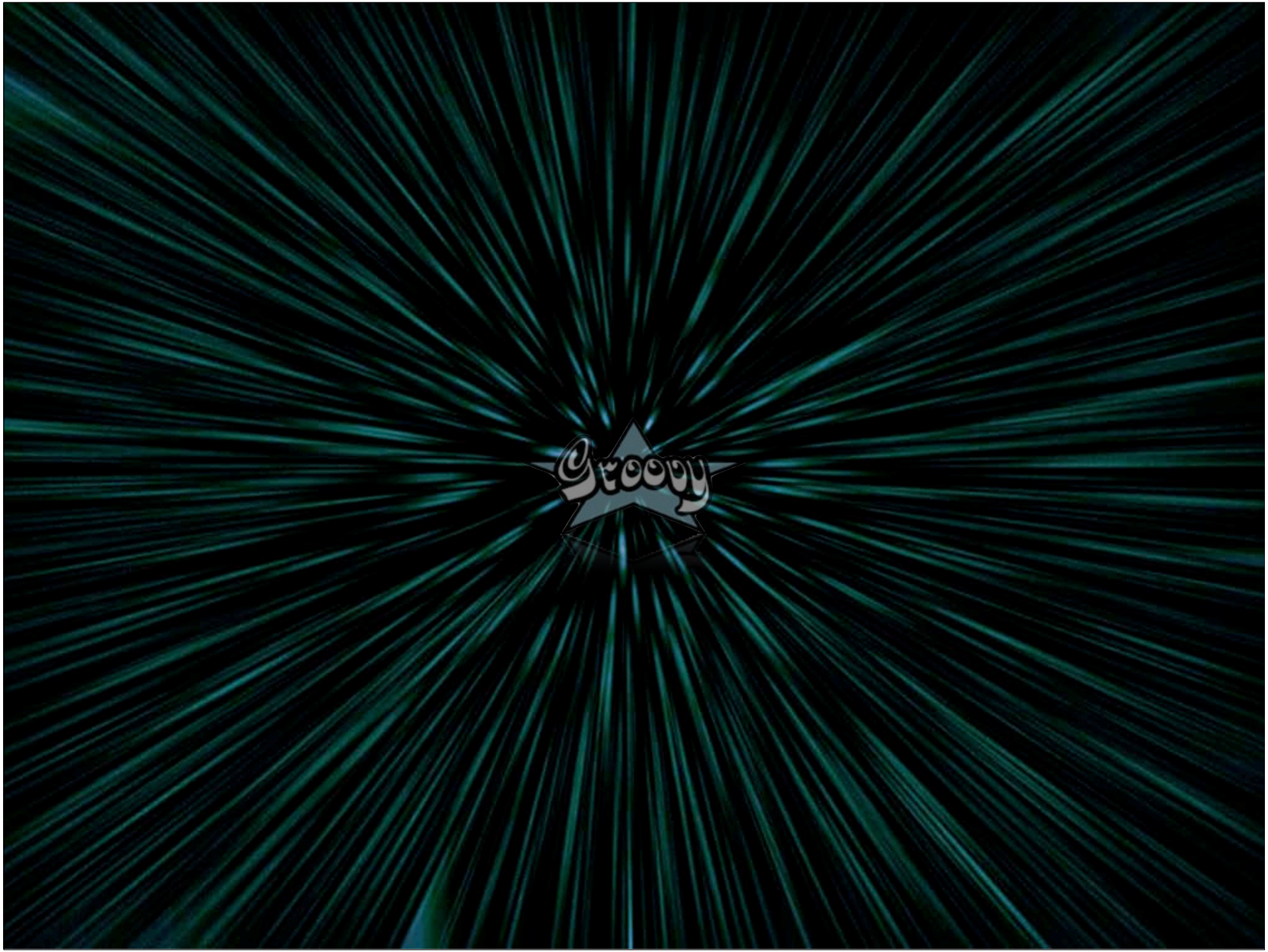
Ruby either builds

abstractions with dsl's

via meta-programming

summary




Groovy

the good

unparalleled Java™ integration

willing to evolve the language

the real JDK™ 2.0!

building up from best-of-breed

the bad

fast changing at the core

single runtime platform

never spread out of the Java™ community

framework approach?


def name
public String getName()
public void setName(String name)

```
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:585)
at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:95)
at org.codehaus.groovy.runtime.MetaClassHelper.doMethodInvoke(MetaClassHelper.java:599)
at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:904)
at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:740)
at org.codehaus.groovy.runtime.InvokerHelper.invokePogoMethod(InvokerHelper.java:773)
at org.codehaus.groovy.runtime.InvokerHelper.invokeMethod(InvokerHelper.java:753)
at org.codehaus.groovy.runtime.InvokerHelper.runScript(InvokerHelper.java:402)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:585)
at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:95)
at org.codehaus.groovy.runtime.MetaClassHelper.doMethodInvoke(MetaClassHelper.java:599)
at groovy.lang.MetaClassImpl.invokeStaticMethod(MetaClassImpl.java:1077)
at org.codehaus.groovy.runtime.InvokerHelper.invokeMethod(InvokerHelper.java:744)
at org.codehaus.groovy.runtime.ScriptBytecodeAdapter.invokeMethodN(ScriptBytecodeAdapter.java:167)
at recorder_test.main(recorder_test.groovy)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:585)
at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:95)
at org.codehaus.groovy.runtime.MetaClassHelper.doMethodInvoke(MetaClassHelper.java:599)
at groovy.lang.MetaClassImpl.invokeStaticMethod(MetaClassImpl.java:1077)
at org.codehaus.groovy.runtime.InvokerHelper.invokeMethod(InvokerHelper.java:744)
at groovy.lang.GroovyShell.runMainOrTestOrRunnable(GroovyShell.java:244)
at groovy.lang.GroovyShell.run(GroovyShell.java:218)
at groovy.lang.GroovyShell.run(GroovyShell.java:147)
at groovy.ui.GroovyMain.processOnce(GroovyMain.java:492)
at groovy.ui.GroovyMain.run(GroovyMain.java:308)
at groovy.ui.GroovyMain.process(GroovyMain.java:294)
at groovy.ui.GroovyMain.processArgs(GroovyMain.java:111)
at groovy.ui.GroovyMain.main(GroovyMain.java:92)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:585)
at org.codehaus.groovy.tools.GroovyStarter.rootLoader(GroovyStarter.java:101)
at org.codehaus.groovy.tools.GroovyStarter.main(GroovyStarter.java:130)
```

the good

well established (older than Java™)

virtually effortless meta-programming

futuristic framework approach

truly cross platform, for the new definition of platform

interpreted, then jit-ted

the bad

some impedance mismatch with Java™ (smaller all the time)

yet another language to learn

unfamiliar frameworks

you have to switch to a new community

you have to learn to *think* like a Ruby-ist

? ' s

please fill out the session evaluations
samples at `github.com/nealford`

NEAL FORD software architect / meme wrangler

ThoughtWorks

nford@thoughtworks.com
3003 Summit Boulevard, Atlanta, GA 30319
www.nealford.com
www.thoughtworks.com
blog: memeagora.blogspot.com
twitter: neal4d

resources

Cow in the Road photo by Sophia Huda, from Flickr, under a Creative Commons license

groovy web site
http://groovy.codehaus.org

jruby site
http://jruby.codehaus.org

ola bini's blog
http://olabini.blogspot.com/

charles nutter's blog
http://headius.blogspot.com/

resources

jeff brown's blog
http://javajeff.blogspot.com/

graeme rocher's blog
http://graemerocher.blogspot.com/

aboutGroovy
http://aboutGroovy.com

venkat subramaniam's blog
http://www.agiledeveloper.com/blog/