ThoughtWorks

introduction to jruby

NEAL FORD software architect / meme wrangler

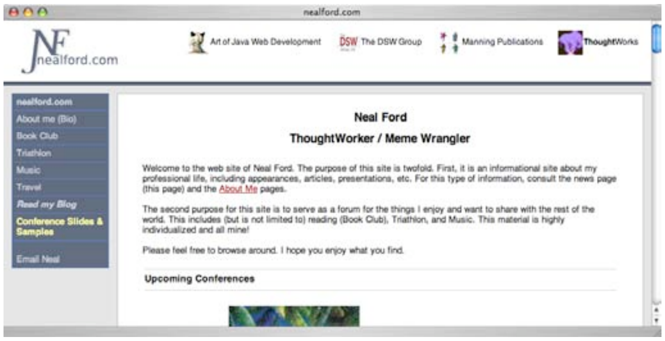
ThoughtWorks

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

housekeeping

ask questions anytime

download slides from nealford.com



Neal Ford

ThoughtWorker / Meme Wrangler

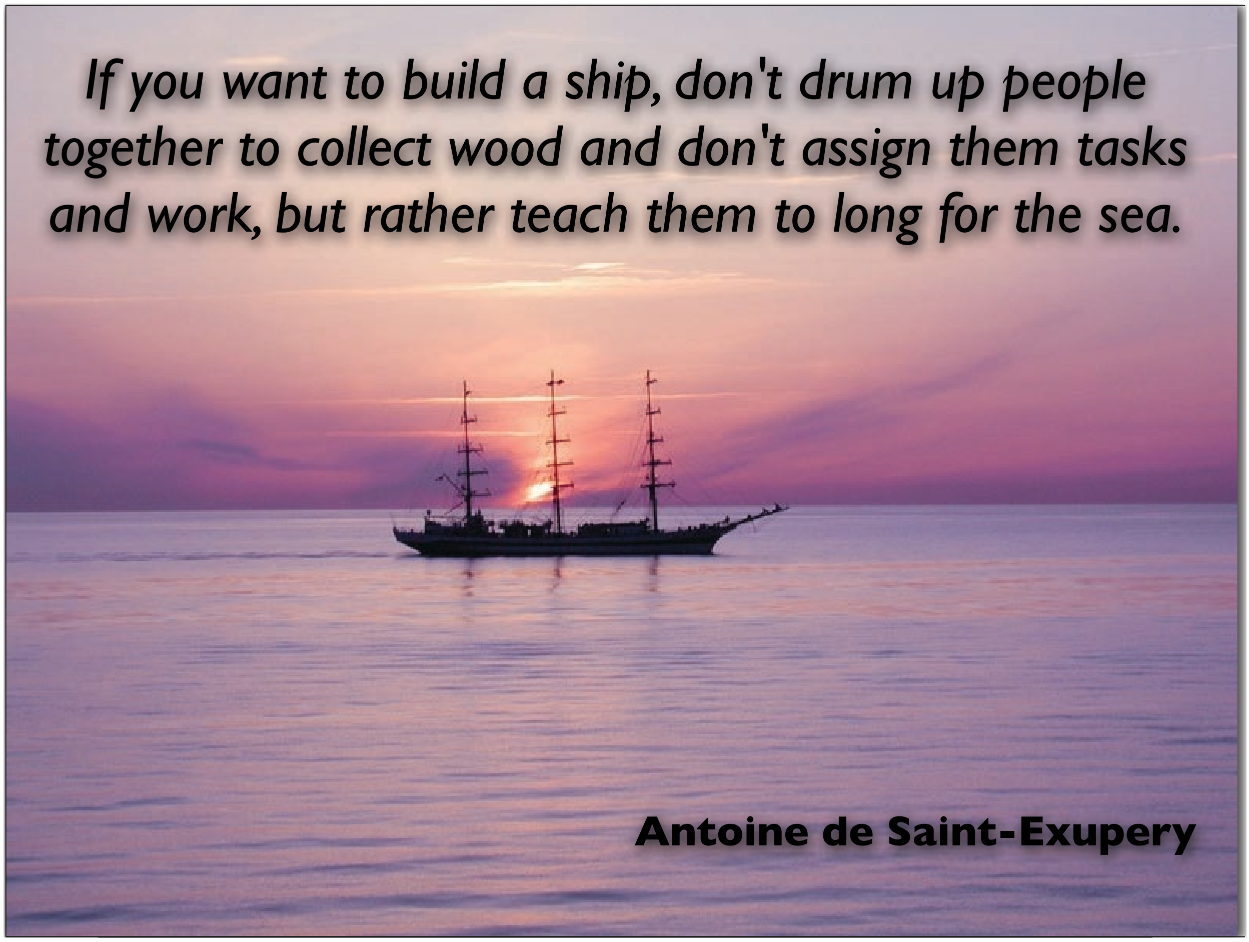
Welcome to the web site of Neal Ford. The purpose of this site is twofold. First, it is an informational site about my professional life, including appearances, articles, presentations, etc. For this type of information, consult the news page (this page) and the About Me pages.

The second purpose for this site is to serve as a forum for the things I enjoy and want to share with the rest of the world. This includes (but is not limited to) reading (Book Club), Triathlon, and Music. This material is highly individualized and all mine!

Please feel free to browse around. I hope you enjoy what you find.

Upcoming Conferences

download samples from **`github.com/nealford`**



If you want to build a ship, don't drum up people together to collect wood and don't assign them tasks and work, but rather teach them to long for the sea.

Antoine de Saint-Exupery

why ruby?

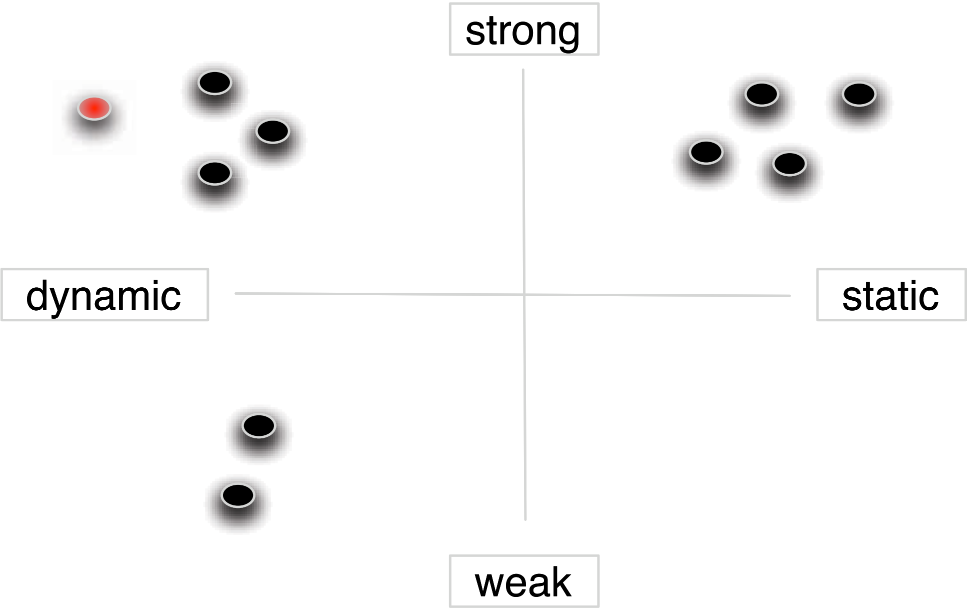
purely object-oriented

compact syntax

advanced language features

rails!

dynamically typed



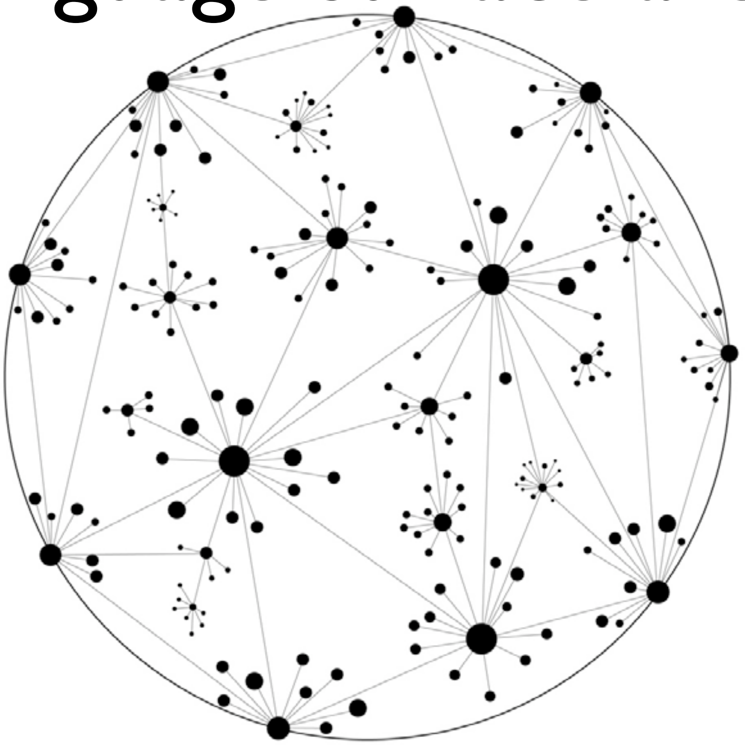
strong

dynamic

static

weak

language surface area



what is jruby?

sophisticated port of ruby to the java platform

mri: matz reference implementation

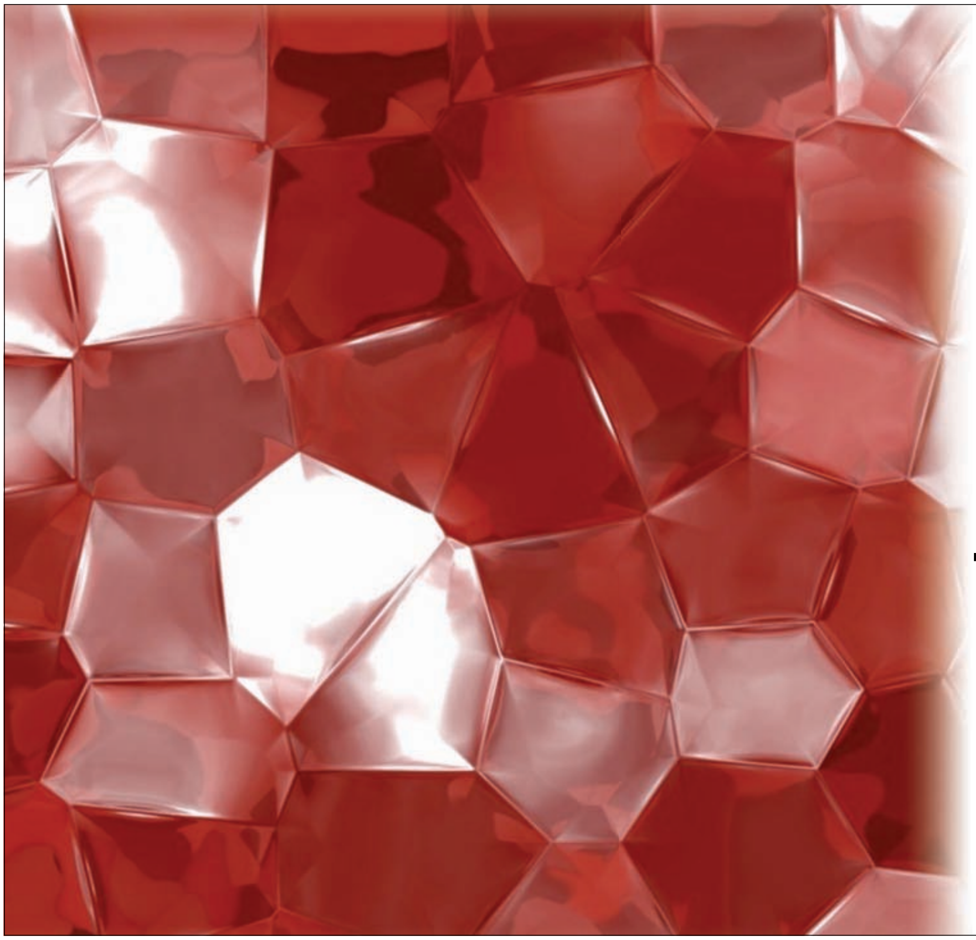
- written in c & ruby
- ported to all major (os) platforms

what is jruby?

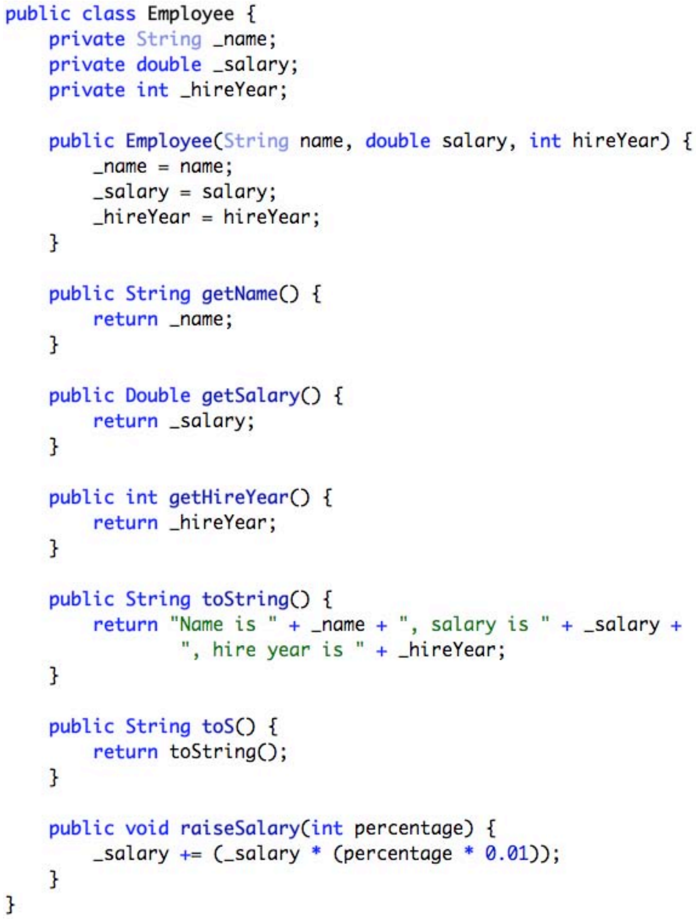
jruby 1.0 ported interpreter to java

jruby 1.1 created a compiler

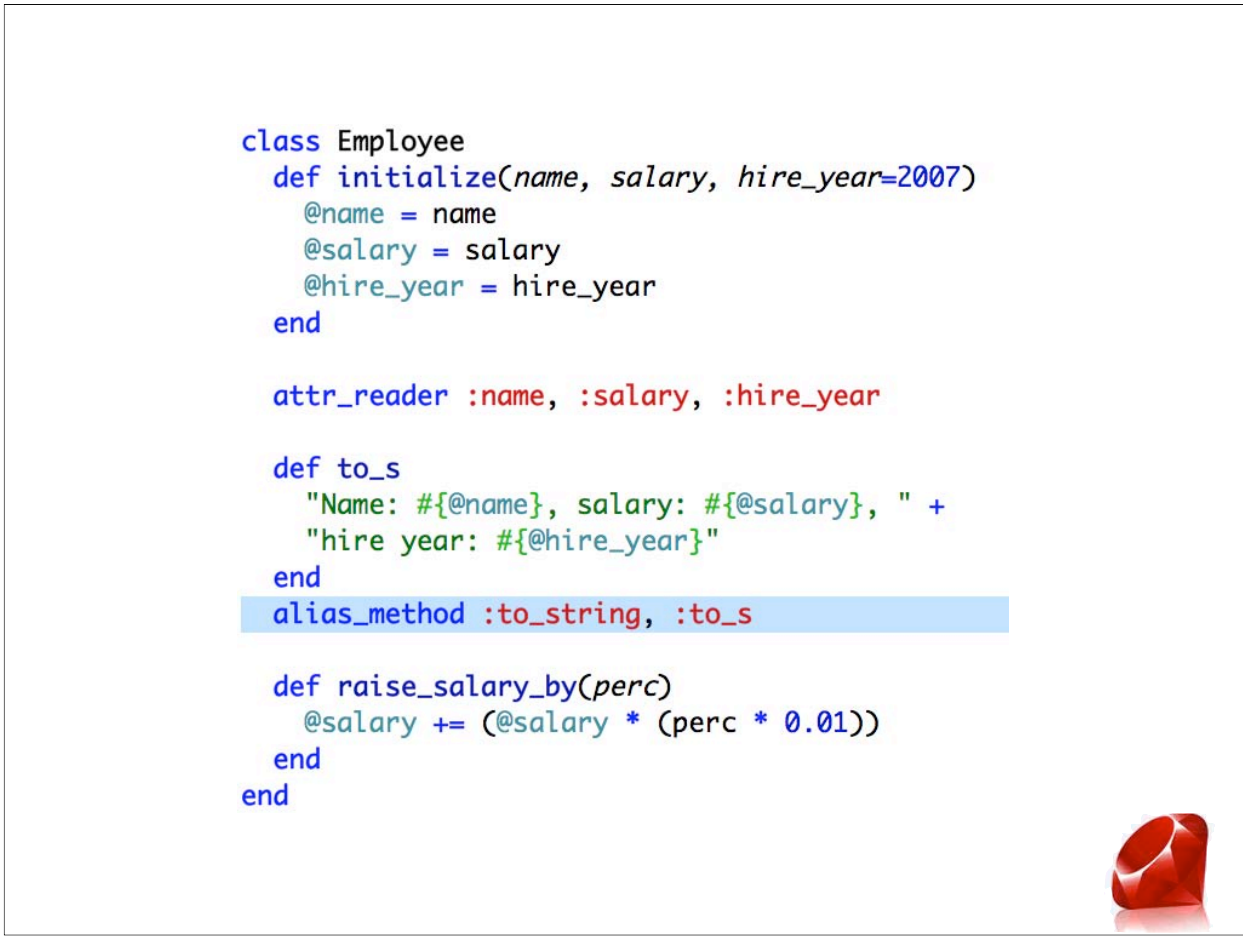
jruby is now the fastest version of ruby



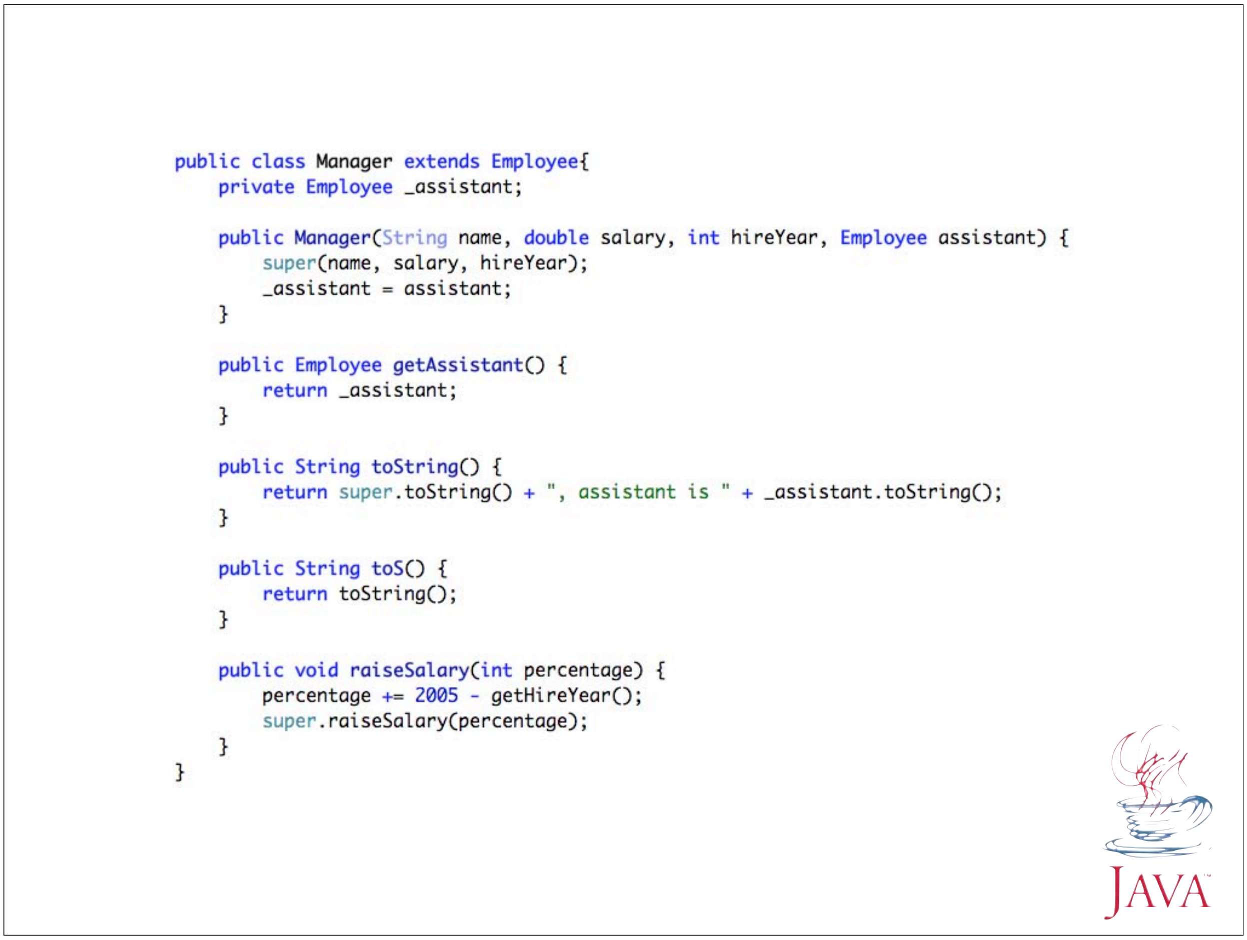
ruby facets



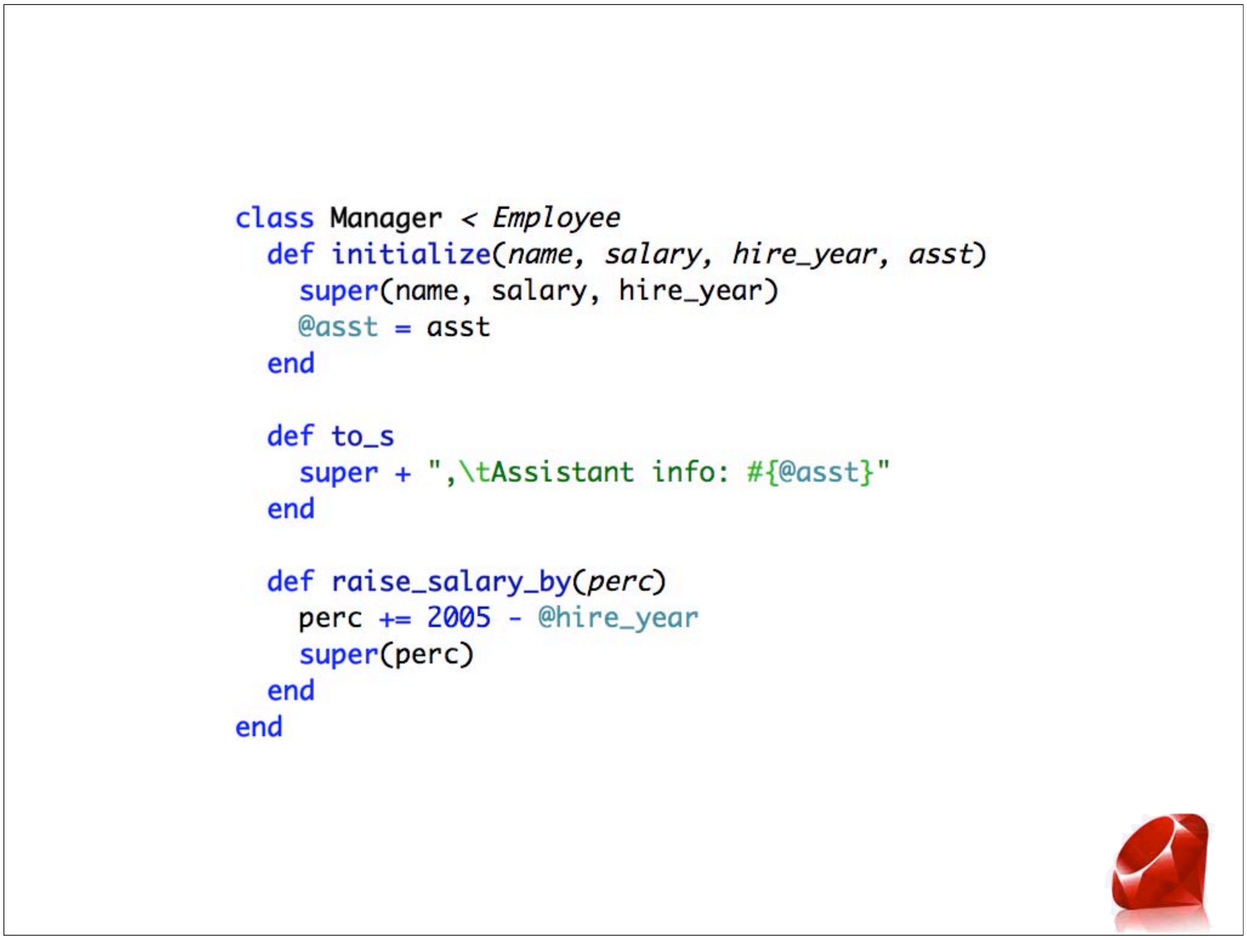
```
public class Employee {
    private String _name;
    private double _salary;
    private int _hireYear;

    public Employee(String name, double salary, int hireYear) {
        _name = name;
        _salary = salary;
        _hireYear = hireYear;
    }

    public String getName() {
        return _name;
    }

    public Double getSalary() {
        return _salary;
    }

    public int getHireYear() {
        return _hireYear;
    }

    public String toString() {
        return "Name is " + _name + ", salary is " + _salary +
                ", hire year is " + _hireYear;
    }

    public String toS() {
        return toString();
    }

    public void raiseSalary(int percentage) {
        _salary += (_salary * (percentage * 0.01));
    }
}
```



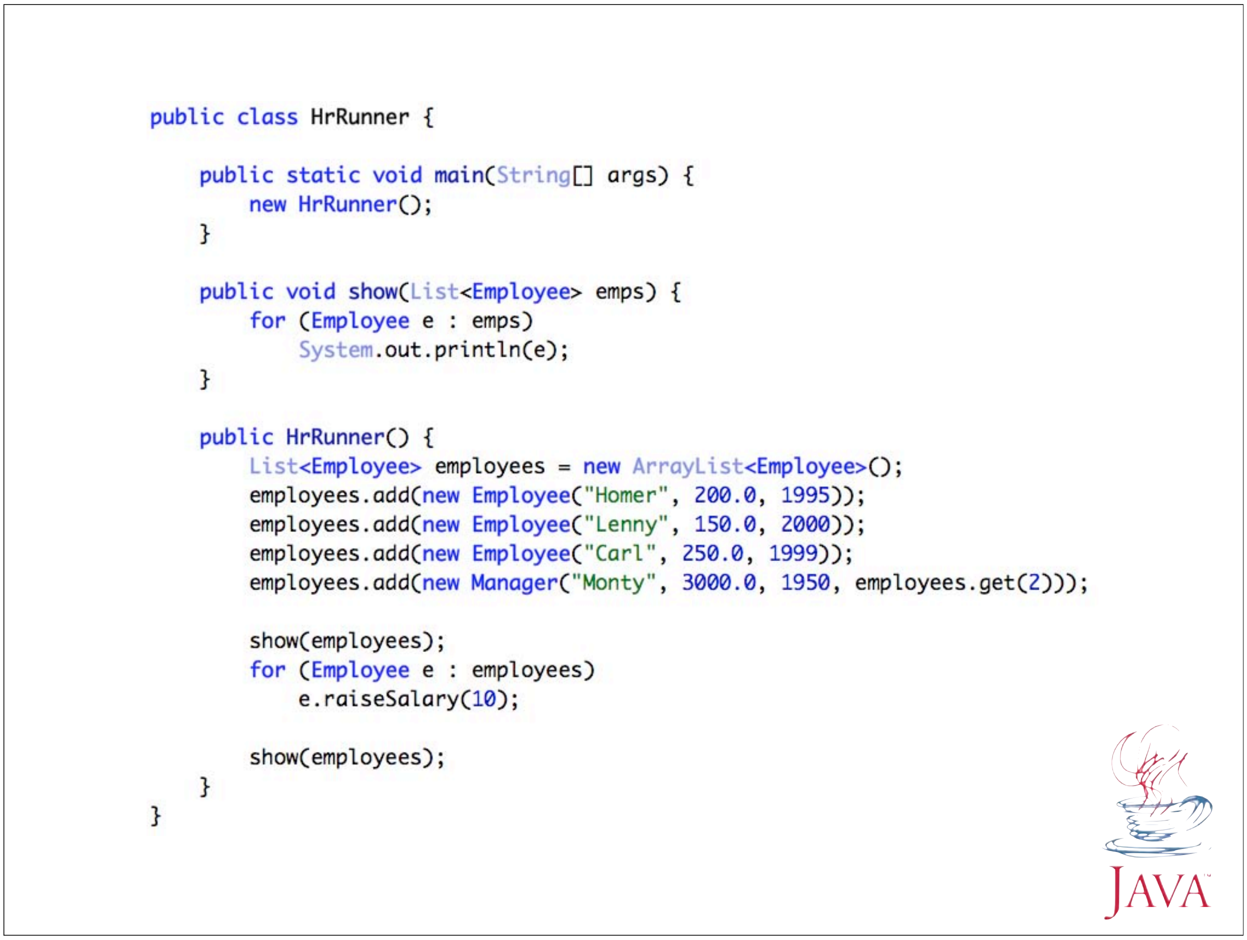
```ruby
class Employee
  def initialize(name, salary, hire_year=2007)
    @name = name
    @salary = salary
    @hire_year = hire_year
  end

  attr_reader :name, :salary, :hire_year

  def to_s
    "Name: #{@name}, salary: #{@salary}, " +
    "hire year: #{@hire_year}"
  end
  alias_method :to_string, :to_s

  def raise_salary_by(perc)
    @salary += (@salary * (perc * 0.01))
  end
end
```



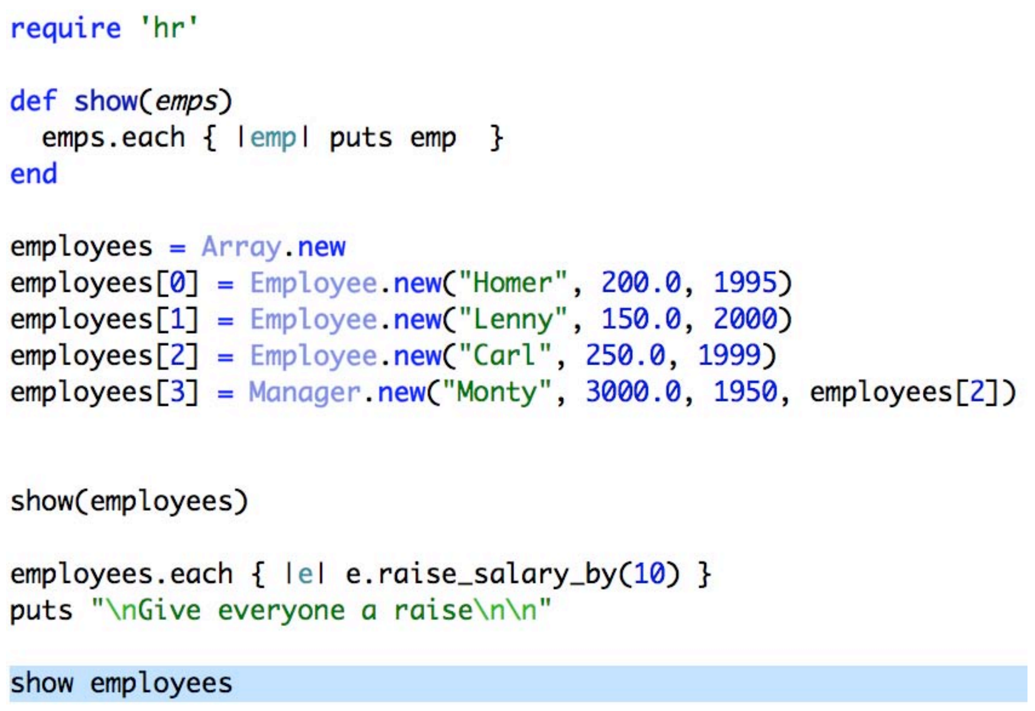
```java
public class Manager extends Employee{
    private Employee _assistant;

    public Manager(String name, double salary, int hireYear, Employee assistant) {
        super(name, salary, hireYear);
        _assistant = assistant;
    }

    public Employee getAssistant() {
        return _assistant;
    }

    public String toString() {
        return super.toString() + ", assistant is " + _assistant.toString();
    }

    public String toS() {
        return toString();
    }

    public void raiseSalary(int percentage) {
        percentage += 2005 - getHireYear();
        super.raiseSalary(percentage);
    }
}
```



```ruby
class Manager < Employee
  def initialize(name, salary, hire_year, asst)
    super(name, salary, hire_year)
    @asst = asst
  end

  def to_s
    super + ",\tAssistant info: #{@asst}"
  end

  def raise_salary_by(perc)
    perc += 2005 - @hire_year
    super(perc)
  end
end
```



```java
public class HrRunner {

    public static void main(String[] args) {
        new HrRunner();
    }

    public void show(List<Employee> emps) {
        for (Employee e : emps)
            System.out.println(e);
    }

    public HrRunner() {
        List<Employee> employees = new ArrayList<Employee>();
        employees.add(new Employee("Homer", 200.0, 1995));
        employees.add(new Employee("Lenny", 150.0, 2000));
        employees.add(new Employee("Carl", 250.0, 1999));
        employees.add(new Manager("Monty", 3000.0, 1950, employees.get(2)));

        show(employees);
        for (Employee e : employees)
            e.raiseSalary(10);

        show(employees);
    }
}
```



```
require 'hr'

def show(emps)
  emps.each { |emp| puts emp  }
end

employees = Array.new
employees[0] = Employee.new("Homer", 200.0, 1995)
employees[1] = Employee.new("Lenny", 150.0, 2000)
employees[2] = Employee.new("Carl", 250.0, 1999)
employees[3] = Manager.new("Monty", 3000.0, 1950, employees[2])


show(employees)

employees.each { |e| e.raise_salary_by(10) }
puts "\nGive everyone a raise\n\n"

show employees
```

blocks

delimited with either { } or do..end

both support parameters

closures

```
public class EmployeeList {
    private List<Employee> _employees;

    public List<Employee> getEmployees() {
        return _employees;
    }

    public EmployeeList() {
        _employees = new ArrayList<Employee>();
    }

    public void add(Employee e) {
        _employees.add(e);
    }

    public void deleteFirst() {
        _employees.remove(0);
    }

    public void deleteLast() {
        _employees.remove(_employees.size() - 1);
    }

    public void show() {
        for (Employee e : _employees)
            System.out.println(e.toString());
    }
}
```

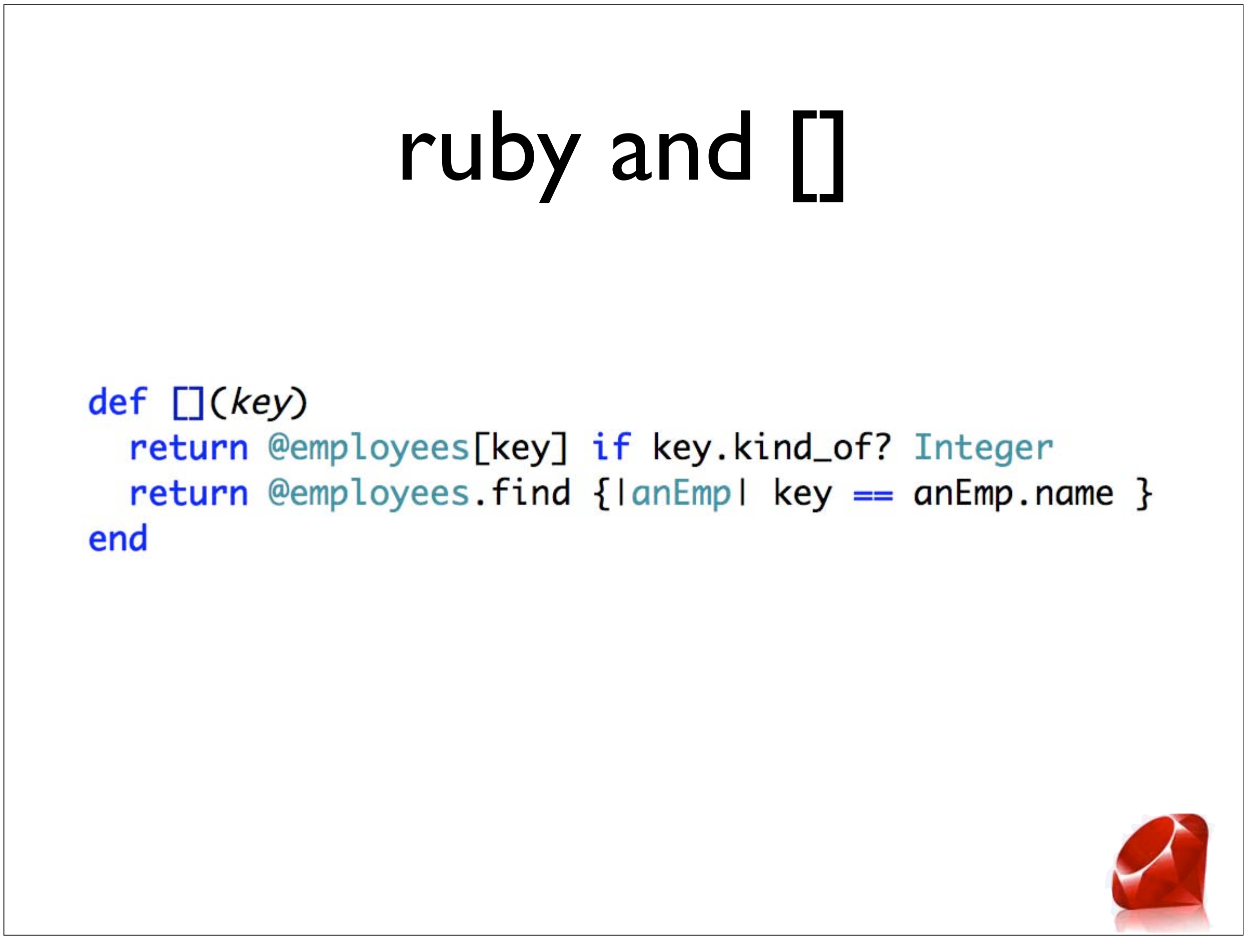
list access

```
public Employee get(int key) {
    return _employees.get(key);
}

public Employee get(String name) {
    for (Employee e : _employees)
        if (e.getName().equals(name))
            return e;
    return null;
}
```

access tests

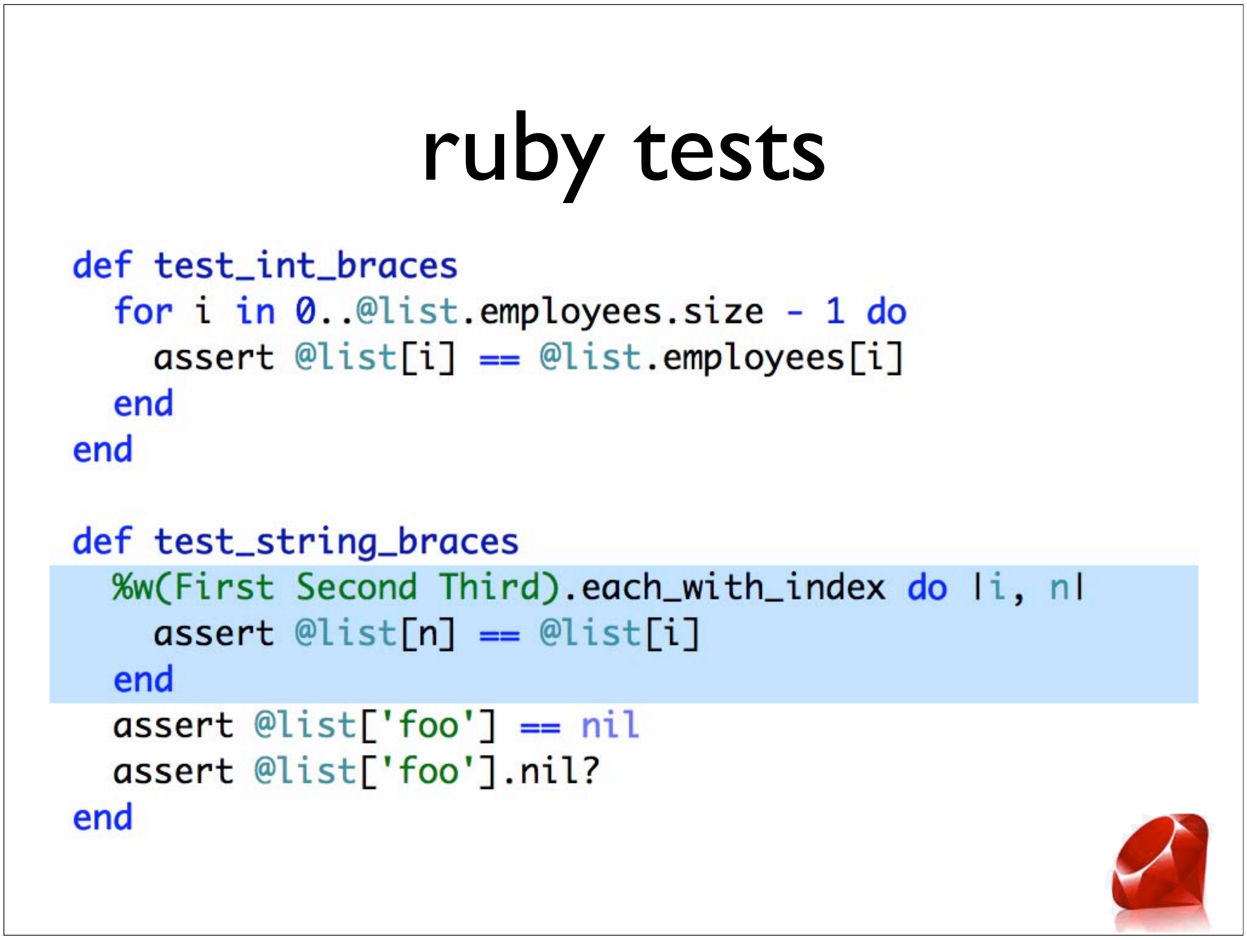
```
@Test public void get_with_int() {
    assertThat(_list.get(0),
            sameInstance(_list.getEmployees().get(0)));
}

@Test public void get_with_string() {
    assertThat(_list.get("Second"),
            sameInstance(_list.getEmployees().get(1)));
    assertNull(_list.get("Homer"));
}
```

```
class EmployeeList
  def initialize
    @employees = Array.new
  end

  attr_reader :employees

  def add(an_employee)
    @employees.push(an_employee)
  end
  alias_method :<<, :add

  def delete_first
    @employees.shift
  end

  def delete_last
    @employees.pop
  end

  def show
    @employees.each { |e| puts e }
  end
```



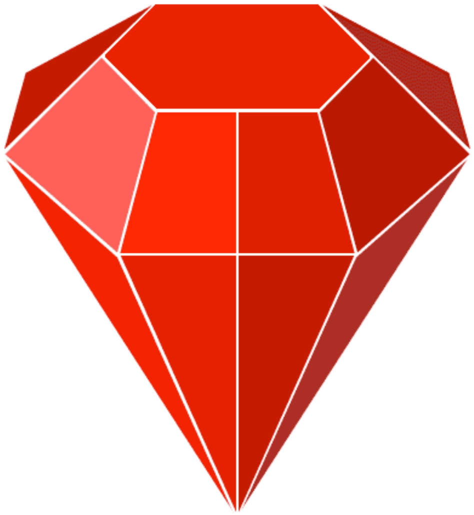
ruby and []

```
def [](key)
  return @employees[key] if key.kind_of? Integer
  return @employees.find {|anEmp| key == anEmp.name }
end
```

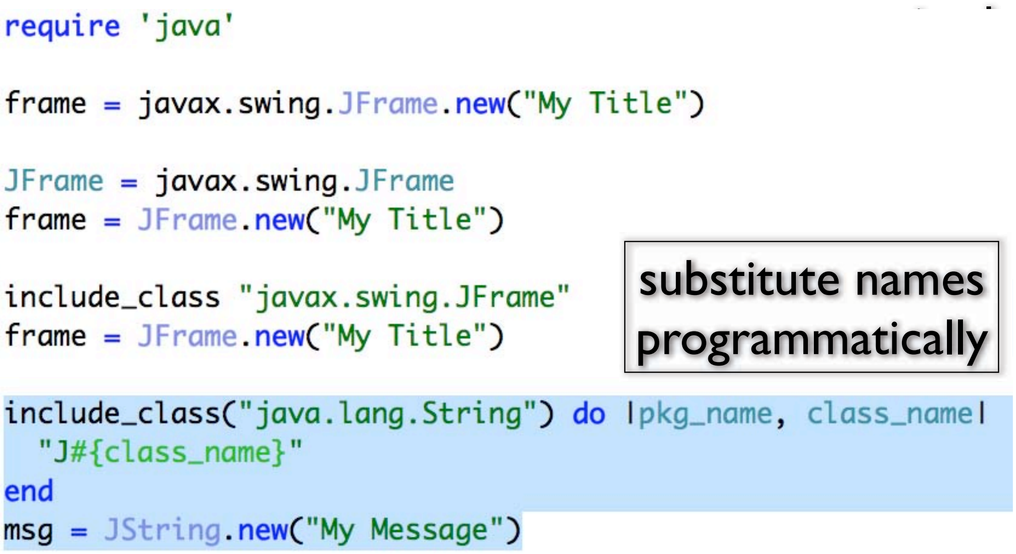


ruby tests

```
def test_int_braces
  for i in 0..@list.employees.size - 1 do
    assert @list[i] == @list.employees[i]
  end
end

def test_string_braces
  %w(First Second Third).each_with_index do |i, n|
    assert @list[n] == @list[i]
  end
  assert @list['foo'] == nil
  assert @list['foo'].nil?
end
```



java classes in jruby



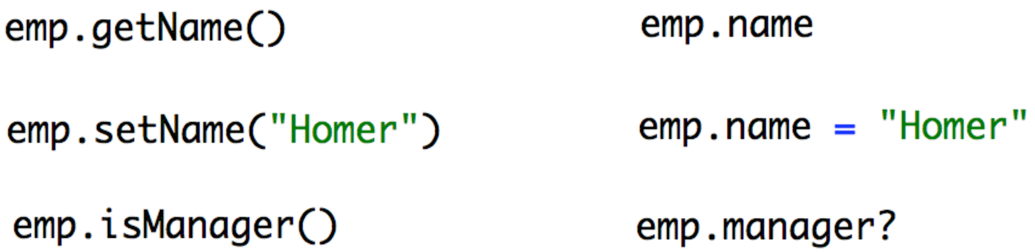
```
require 'java'

frame = javax.swing.JFrame.new("My Title")

JFrame = javax.swing.JFrame
frame = JFrame.new("My Title")

include_class "javax.swing.JFrame"
frame = JFrame.new("My Title")

include_class("java.lang.String") do |pkg_name, class_name|
  "J#{class_name}"
end
msg = JString.new("My Message")
```

substitute names programmatically

calling semantics

call methods like ruby methods

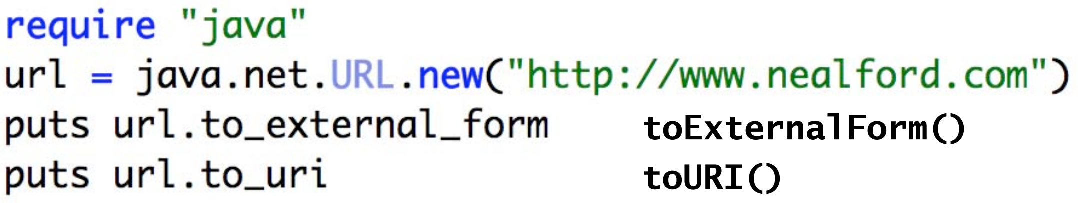
get/set/is methods invoked ala ruby



`emp.getName()`	`emp.name`
`emp.setName("Homer")`	`emp.name = "Homer"`
`emp.isManager()`	`emp.manager?`

calling semantics

camelcase java names may be called with underscores



```
require "java"
url = java.net.URL.new("http://www.nealford.com")
puts url.to_external_form      toExternalForm()
puts url.to_uri                toURI()
```

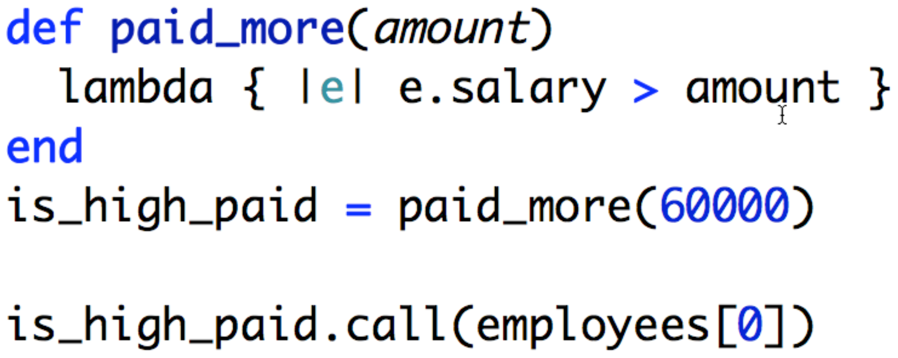
closures

a function evaluated in an environment containing one or more bound variables

can be passed as data

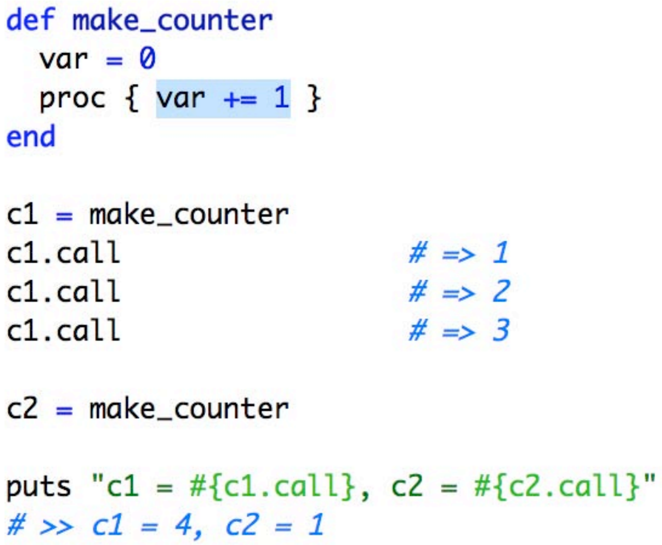
instances of Proc

closure



```
def paid_more(amount)
  lambda { |e| e.salary > amount }
end
is_high_paid = paid_more(60000)

is_high_paid.call(employees[0])
```

the big deal



```
def make_counter
  var = 0
  proc { var += 1 }
end

c1 = make_counter
c1.call                 # => 1
c1.call                 # => 2
c1.call                 # => 3

c2 = make_counter

puts "c1 = #{c1.call}, c2 = #{c2.call}"
# >> c1 = 4, c2 = 1
```

the big deal

executable data

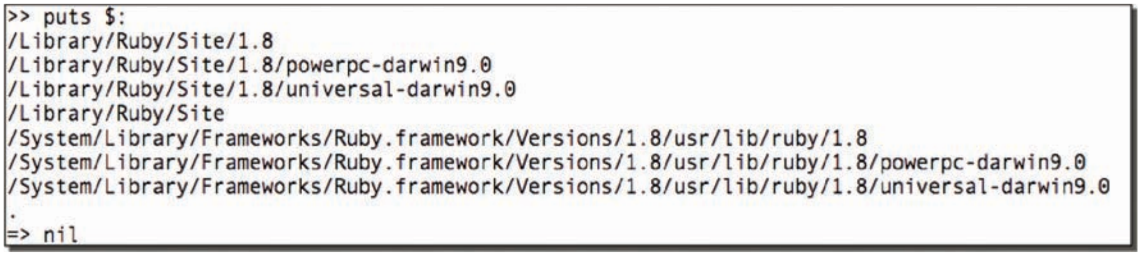
compact syntax

- crucial because of pervasiveness

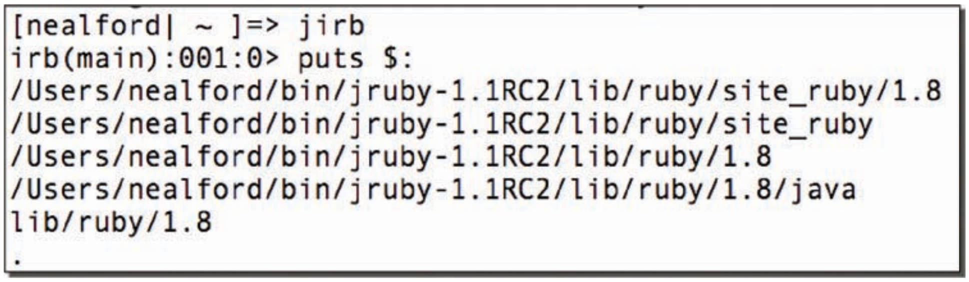
- heavily used in infrastructure

open classes

about classpaths



```
>> puts $:
/Library/Ruby/Site/1.8
/Library/Ruby/Site/1.8/powerpc-darwin9.0
/Library/Ruby/Site/1.8/universal-darwin9.0
/Library/Ruby/Site
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/1.8
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/1.8/powerpc-darwin9.0
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/1.8/universal-darwin9.0
.
=> nil
```



```
[nealford| ~ ]=> jirb
irb(main):001:0> puts $:
/Users/nealford/bin/jruby-1.1RC2/lib/ruby/site_ruby/1.8
/Users/nealford/bin/jruby-1.1RC2/lib/ruby/site_ruby
/Users/nealford/bin/jruby-1.1RC2/lib/ruby/1.8
/Users/nealford/bin/jruby-1.1RC2/lib/ruby/1.8/java
lib/ruby/1.8
.
```

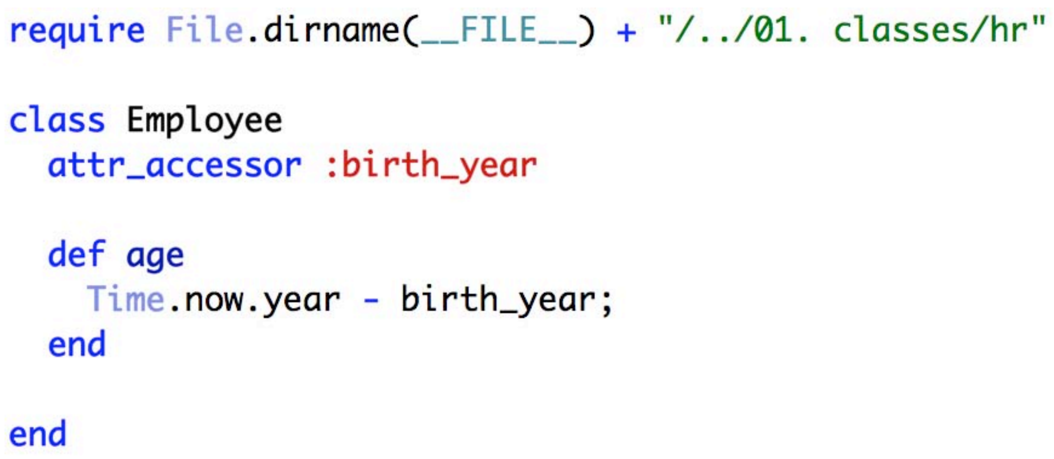
open classes

a class definition for a class that already appears on the classpath reopens the class

allows

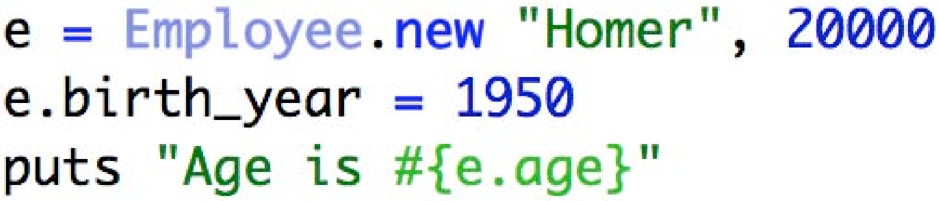
- adding new methods
- overriding existing methods
- removing methods

open employee



```
require File.dirname(__FILE__) + "/../01. classes/hr"

class Employee
  attr_accessor :birth_year

  def age
    Time.now.year - birth_year;
  end

end
```

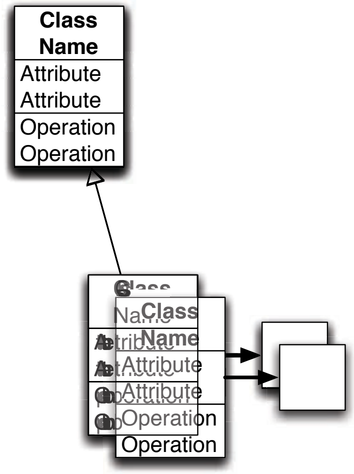
open employee



```
e = Employee.new "Homer", 20000
e.birth_year = 1950
puts "Age is #{e.age}"
```

open classes redux

shadow meta-class



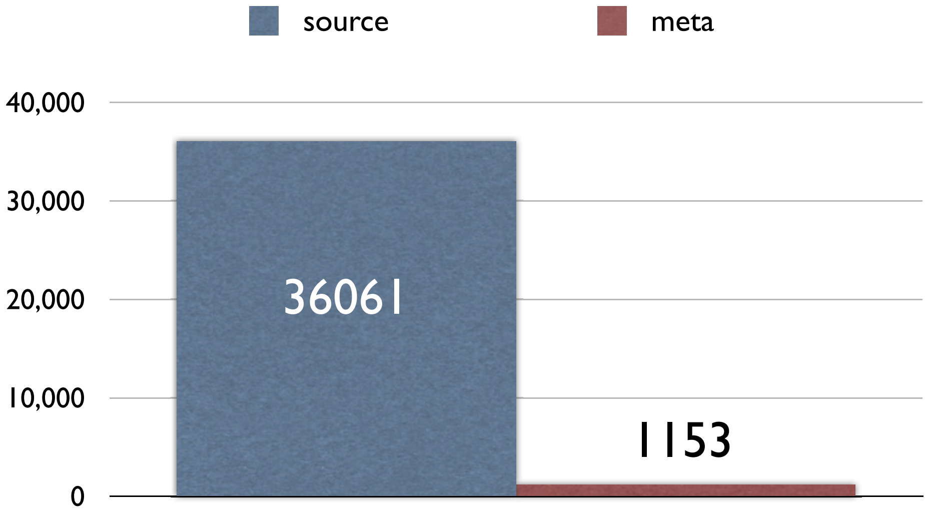
shadow meta-class

```ruby
e = Employee.new "Homer", 20000
e.birth_year = 1950
puts "Age is #{e.age}"

def e.big_raise(by_perc)
    @salary += (@salary * (by_perc * 0.1))
end

old_salary = e.salary
e.big_raise(5)
puts "Big money! From #{old_salary} to #{e.salary}"
```

shadow meta-class

```ruby
def e.raise_salary_by(perc)
  @salary -= (@salary * (perc * 0.01))
end

old_salary = e.salary
e.raise_salary_by(5)
puts "Small money! From #{old_salary} to #{e.salary}"
```

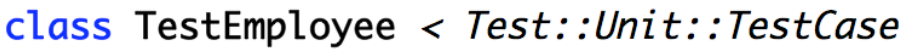
meta-programming



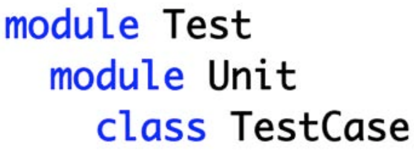
modules (aka mixins)

allows logical grouping of classes, methods, and constants

namespaces



```
class TestEmployee < Test::Unit::TestCase
```



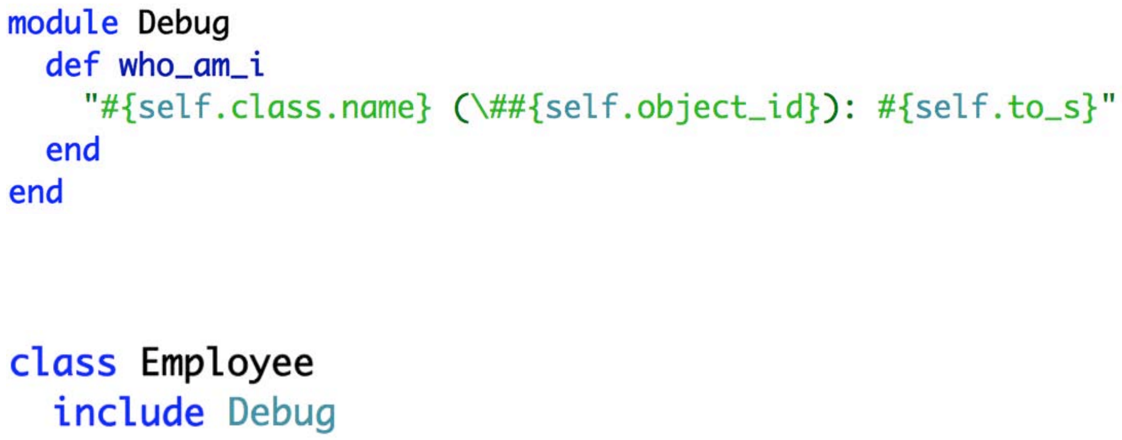
```
module Test
  module Unit
    class TestCase
```

mixins

when you **include** a module into a class, the module's methods are "mixed into" the class

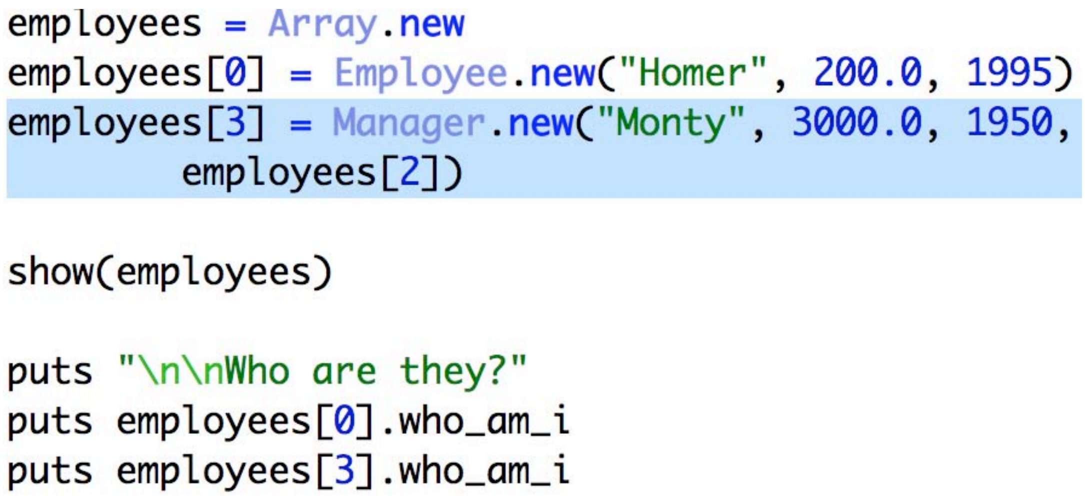
methods defined in the module may interact with the class's parts

mixin



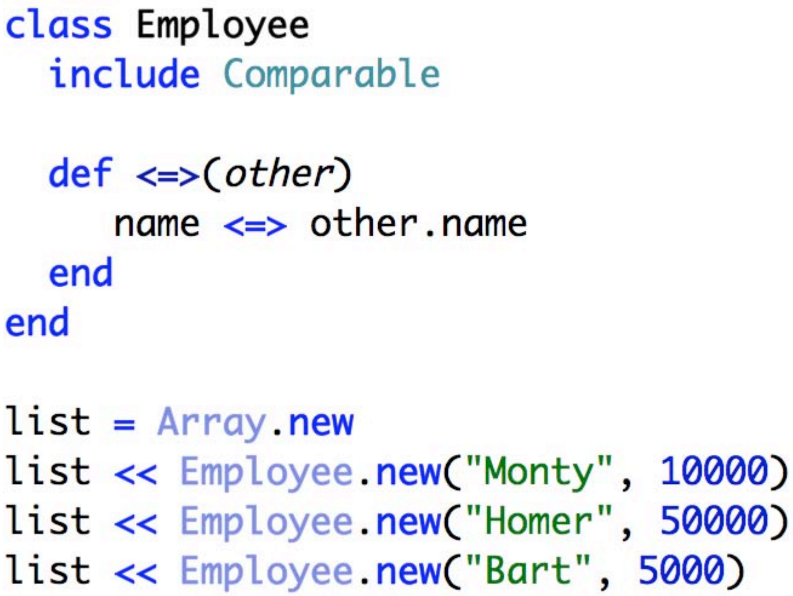
```
module Debug
  def who_am_i
    "#{self.class.name} (\##{self.object_id}): #{self.to_s}"
  end
end



class Employee
  include Debug
```

mixin



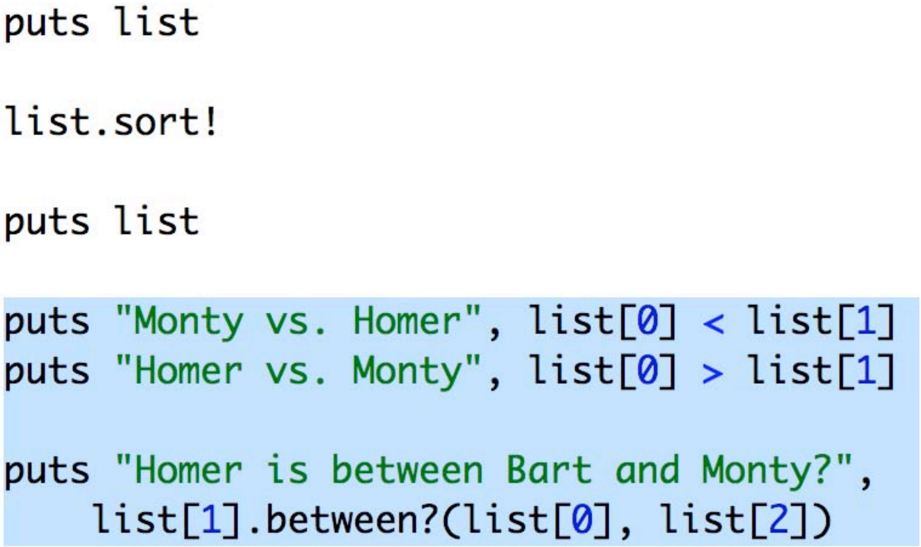
```
employees = Array.new
employees[0] = Employee.new("Homer", 200.0, 1995)
employees[3] = Manager.new("Monty", 3000.0, 1950,
        employees[2])

show(employees)

puts "\n\nWho are they?"
puts employees[0].who_am_i
puts employees[3].who_am_i
```

comparisons



```ruby
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)
```

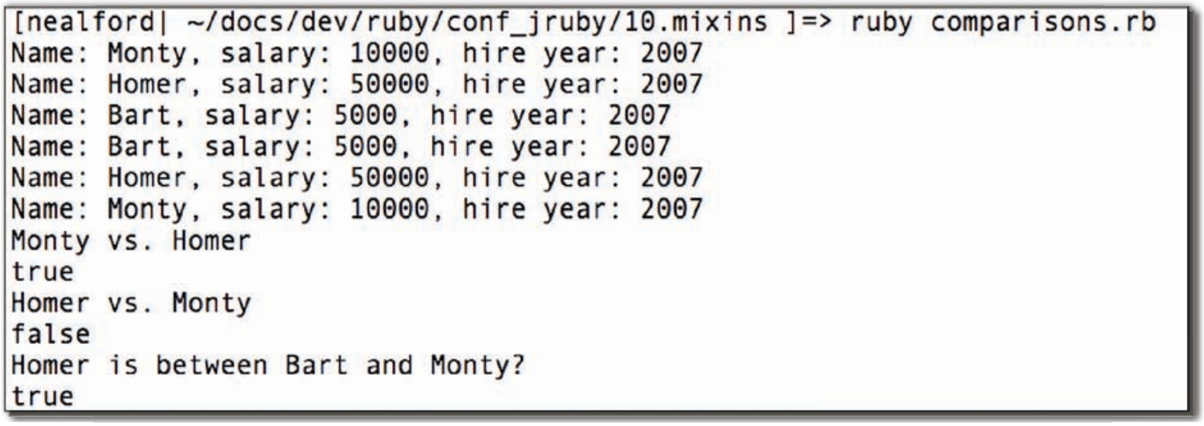
comparisons



```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])
```

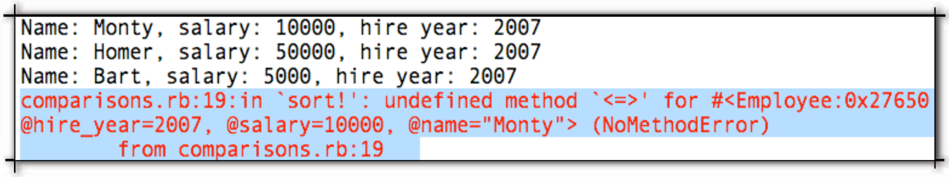
CONTRACT

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

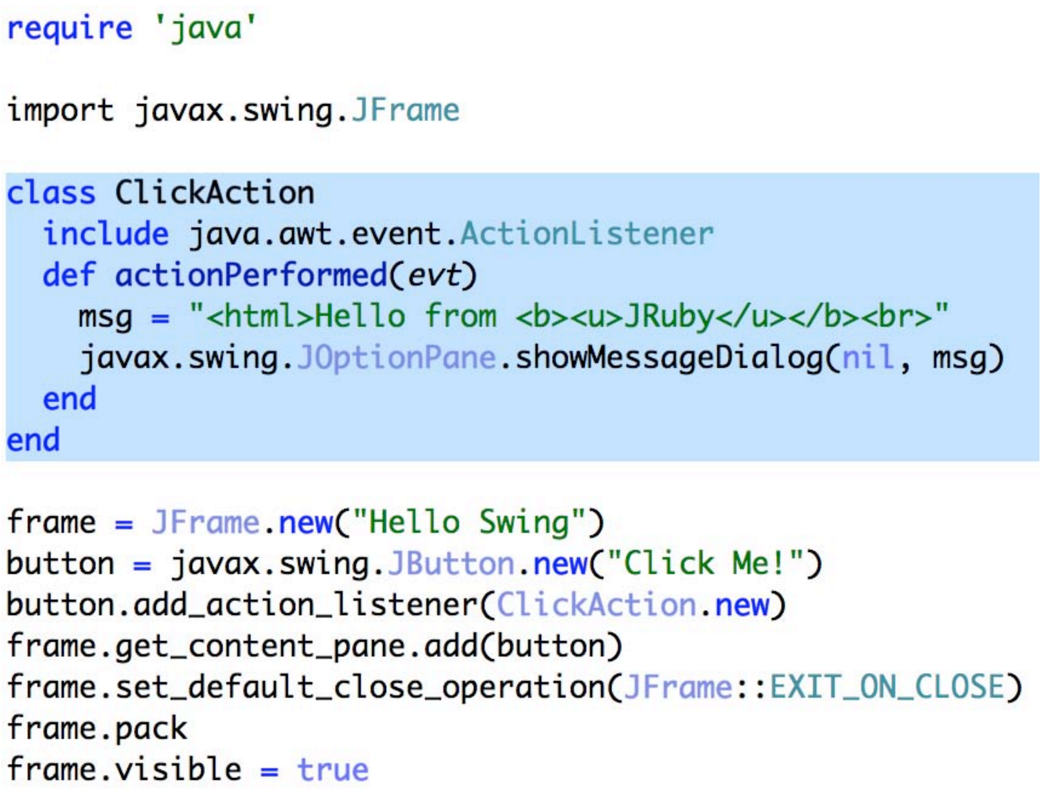


```
Name: Monty, salary: 10000, hire year: 2007
Name: Homer, salary: 50000, hire year: 2007
Name: Bart, salary: 5000, hire year: 2007
comparisons.rb:19:in `sort!': undefined method `<=>' for #<Employee:0x27650
@hire_year=2007, @salary=10000, @name="Monty"> (NoMethodError)
        from comparisons.rb:19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

```
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])
```

swing in jruby

just as ugly as in java!



```
require 'java'

import javax.swing.JFrame

class ClickAction
  include java.awt.event.ActionListener
  def actionPerformed(evt)
    msg = "<html>Hello from <b><u>JRuby</u></b><br>"
    javax.swing.JOptionPane.showMessageDialog(nil, msg)
  end
end

frame = JFrame.new("Hello Swing")
button = javax.swing.JButton.new("Click Me!")
button.add_action_listener(ClickAction.new)
frame.get_content_pane.add(button)
frame.set_default_close_operation(JFrame::EXIT_ON_CLOSE)
frame.pack
frame.visible = true
```

swing take 2

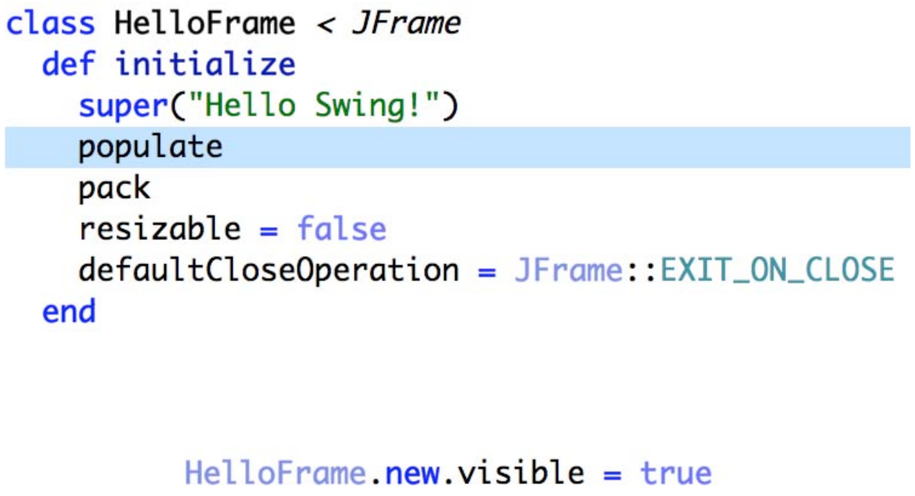
```
class BlockActionListener
  include java.awt.event.ActionListener

  def initialize(&block)
    super
    @block = block
  end

  def actionPerformed(e)
    @block.call(e)
  end
end
```

swing take 2

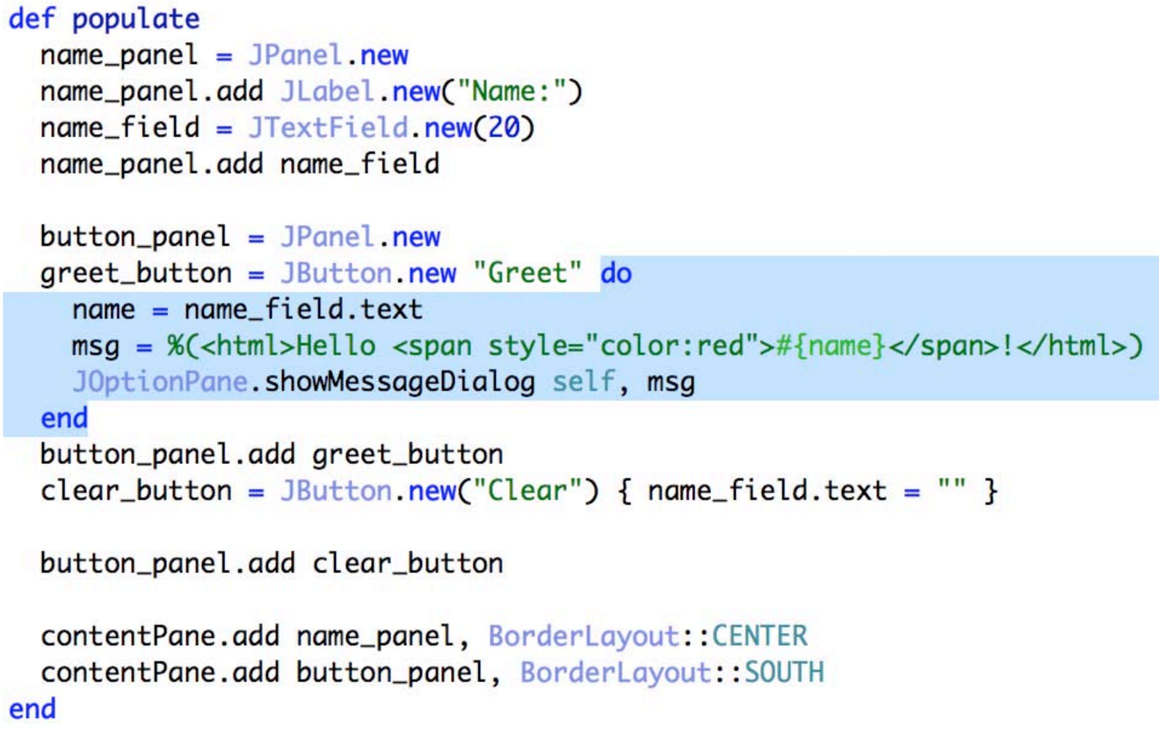
```
class JButton
  def initialize(name, &block)
    super(name)
    addActionListener(BlockActionListener.new(&block))
  end
end
```

```
clear_button = JButton.new("Clear") { name_field.text = "" }
```

swing, take 2

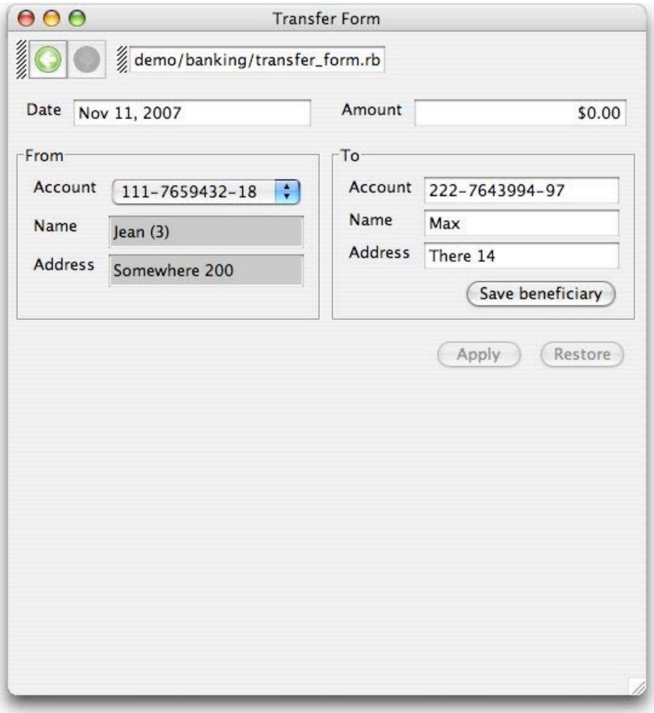


```
class HelloFrame < JFrame
  def initialize
    super("Hello Swing!")
    populate
    pack
    resizable = false
    defaultCloseOperation = JFrame::EXIT_ON_CLOSE
  end


          HelloFrame.new.visible = true
```



```
def populate
  name_panel = JPanel.new
  name_panel.add JLabel.new("Name:")
  name_field = JTextField.new(20)
  name_panel.add name_field

  button_panel = JPanel.new
  greet_button = JButton.new "Greet" do
    name = name_field.text
    msg = %(<html>Hello <span style="color:red">#{name}</span>!</html>)
    JOptionPane.showMessageDialog self, msg
  end
  button_panel.add greet_button
  clear_button = JButton.new("Clear") { name_field.text = "" }

  button_panel.add clear_button

  contentPane.add name_panel, BorderLayout::CENTER
  contentPane.add button_panel, BorderLayout::SOUTH
end
```

swiby: jruby + swing



Transfer Form

demo/banking/transfer_form.rb

Date	Nov 11, 2007	Amount	$0.00

From

Account	111-7659432-18
Name	Jean (3)
Address	Somewhere 200

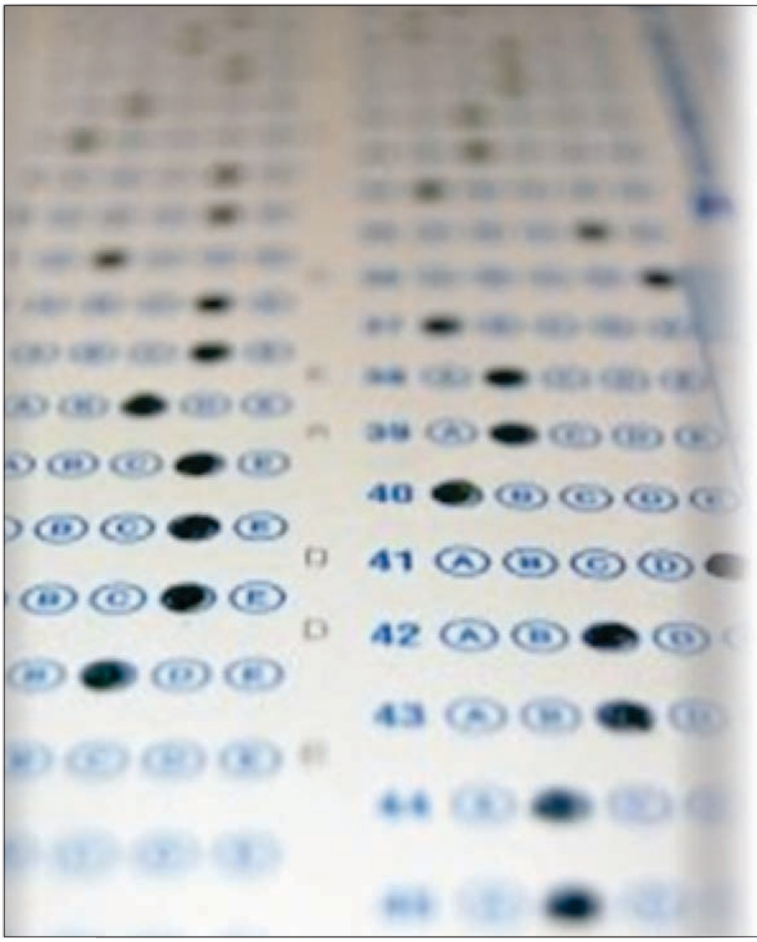
To

Account	222-7643994-97
Name	Max
Address	There 14

```
require 'transfer_ui'

from_accounts = Account.find_from_accounts
to_accounts = Account.find_to_accounts

current = Transfer.new 0.dollars, from_accounts[2], to_accounts[0]

title "Transfer Form"

content {
  data current
  input "Date", :value_date
  section
  input "Amount", :amount
  next_row
  section "From"
  combo "Account", from_accounts, :account_from do |selection|
      context['account_from.owner'].value = selection.owner
      context['account_from.address'].value = selection.address
  end
  input "Name", :account_from / :owner, :readonly => true
  input "Address", :account_from / :address, :readonly => true
  section "To"
  input "Account", :account_to / :number
  input "Name", :account_to / :owner
  input "Address", :account_to / :address
  button "Save beneficiary"
  next_row
  command :apply, :restore
}

$context.apply_styles $context.session.styles if $context.session.styles
$context.start
```

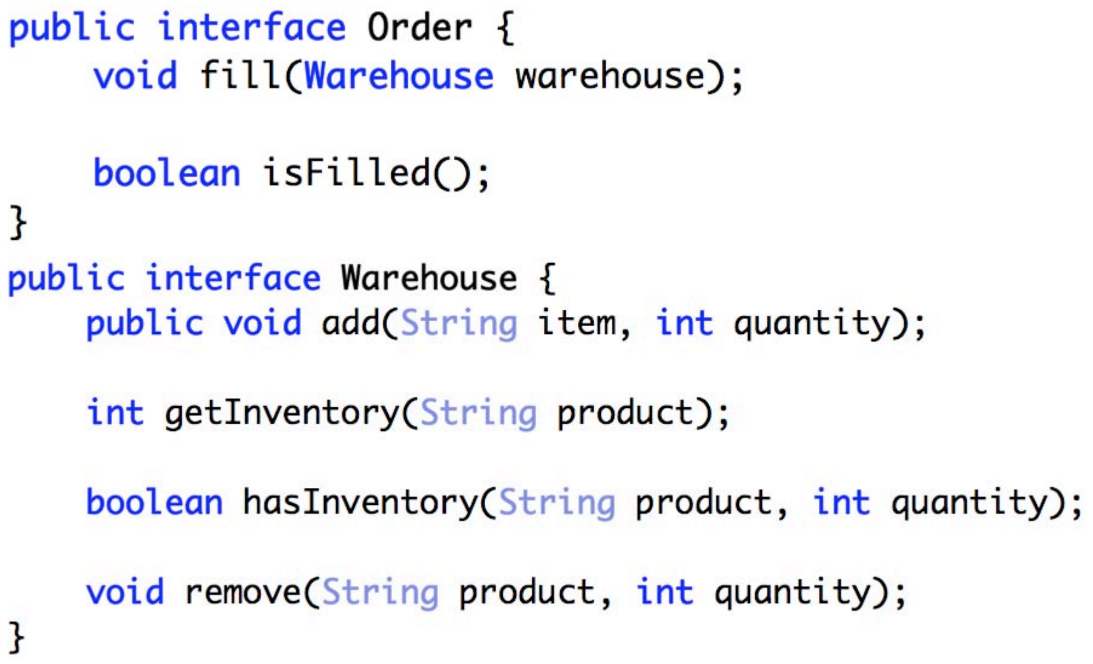
jruby adds "humane interface" methods to standard java classes

`<=>,   <<,   between?`



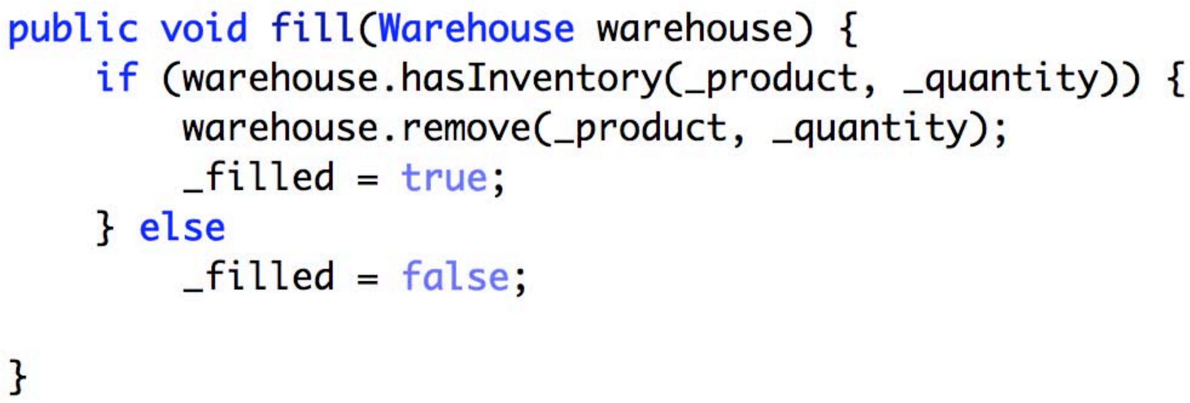
testing
java
with
jruby

the java part



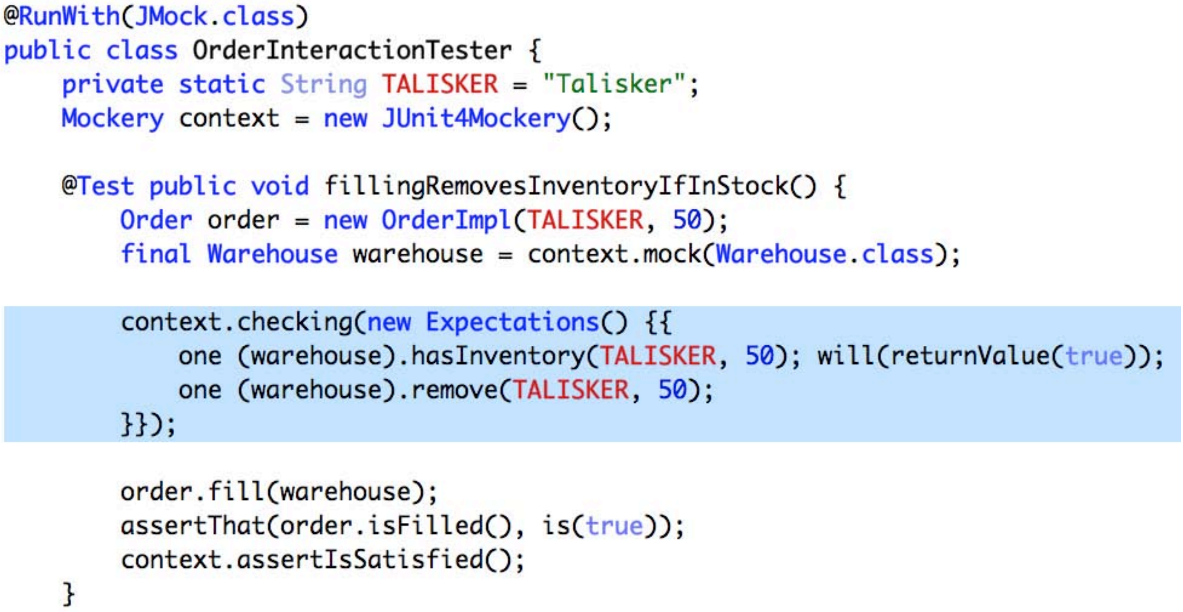
```
public interface Order {
    void fill(Warehouse warehouse);

    boolean isFilled();
}
public interface Warehouse {
    public void add(String item, int quantity);

    int getInventory(String product);

    boolean hasInventory(String product, int quantity);

    void remove(String product, int quantity);
}
```

testing fill()



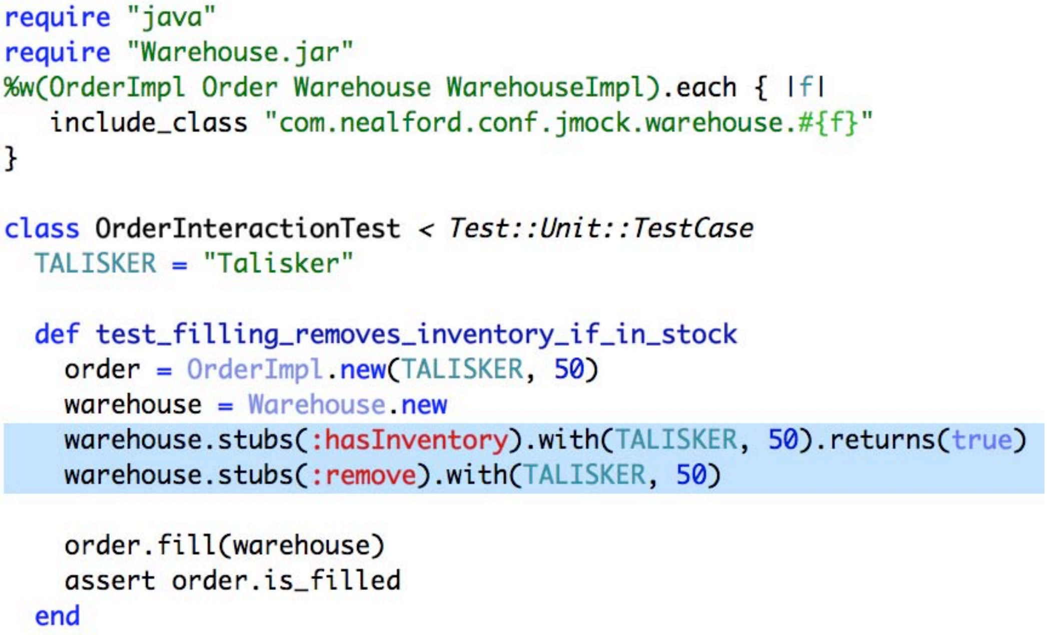
```
public void fill(Warehouse warehouse) {
    if (warehouse.hasInventory(_product, _quantity)) {
        warehouse.remove(_product, _quantity);
        _filled = true;
    } else
        _filled = false;

}
```

jmock



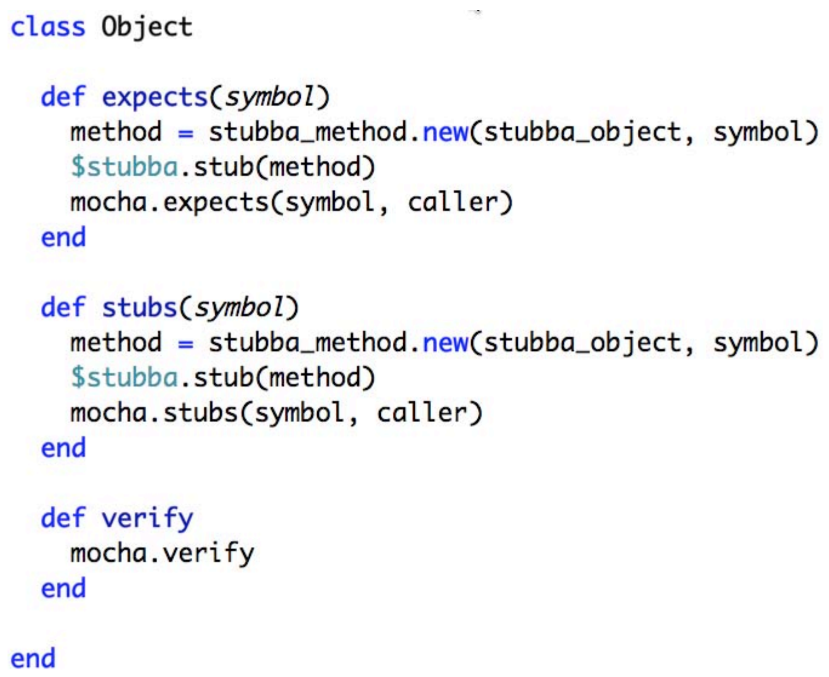
```
@RunWith(JMock.class)
public class OrderInteractionTester {
    private static String TALISKER = "Talisker";
    Mockery context = new JUnit4Mockery();

    @Test public void fillingRemovesInventoryIfInStock() {
        Order order = new OrderImpl(TALISKER, 50);
        final Warehouse warehouse = context.mock(Warehouse.class);

        context.checking(new Expectations() {{
            one (warehouse).hasInventory(TALISKER, 50); will(returnValue(true));
            one (warehouse).remove(TALISKER, 50);
        }});

        order.fill(warehouse);
        assertThat(order.isFilled(), is(true));
        context.assertIsSatisfied();
    }
```

mocha



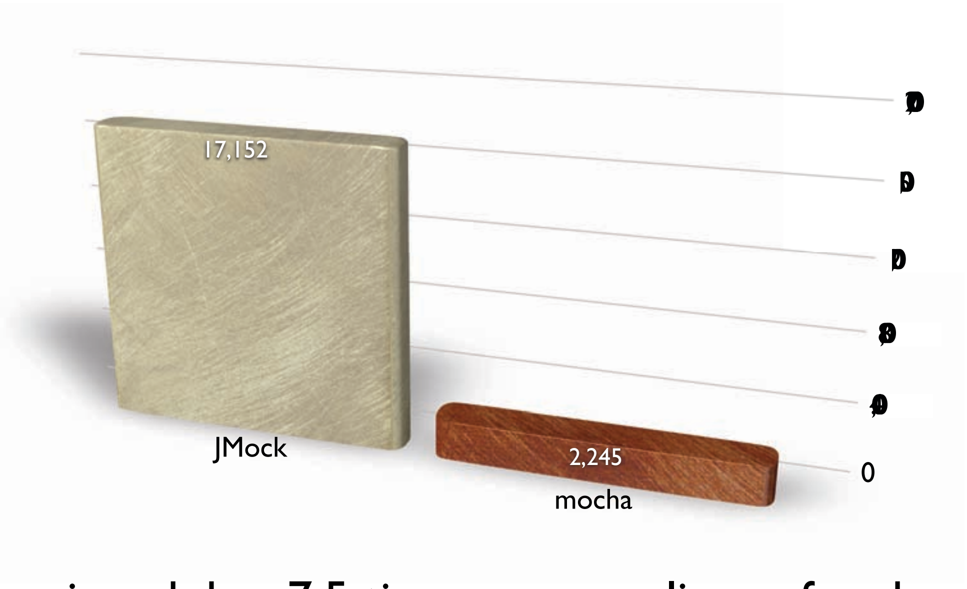
```
require "java"
require "Warehouse.jar"
%w(OrderImpl Order Warehouse WarehouseImpl).each { |f|
  include_class "com.nealford.conf.jmock.warehouse.#{f}"
}

class OrderInteractionTest < Test::Unit::TestCase
  TALISKER = "Talisker"

  def test_filling_removes_inventory_if_in_stock
    order = OrderImpl.new(TALISKER, 50)
    warehouse = Warehouse.new
    warehouse.stubs(:hasInventory).with(TALISKER, 50).returns(true)
    warehouse.stubs(:remove).with(TALISKER, 50)

    order.fill(warehouse)
    assert order.is_filled
  end
```

what does it take???



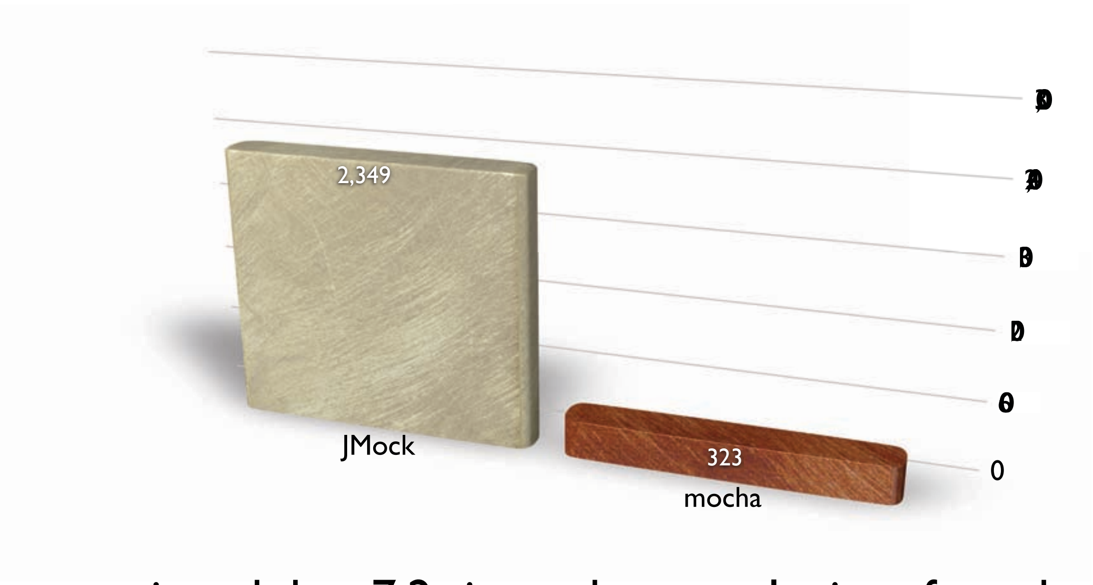
```
class Object

  def expects(symbol)
    method = stubba_method.new(stubba_object, symbol)
    $stubba.stub(method)
    mocha.expects(symbol, caller)
  end

  def stubs(symbol)
    method = stubba_method.new(stubba_object, symbol)
    $stubba.stub(method)
    mocha.stubs(symbol, caller)
  end

  def verify
    mocha.verify
  end

end
```

jmock vs mocha loc



jmock has 7.5 times as many lines of code

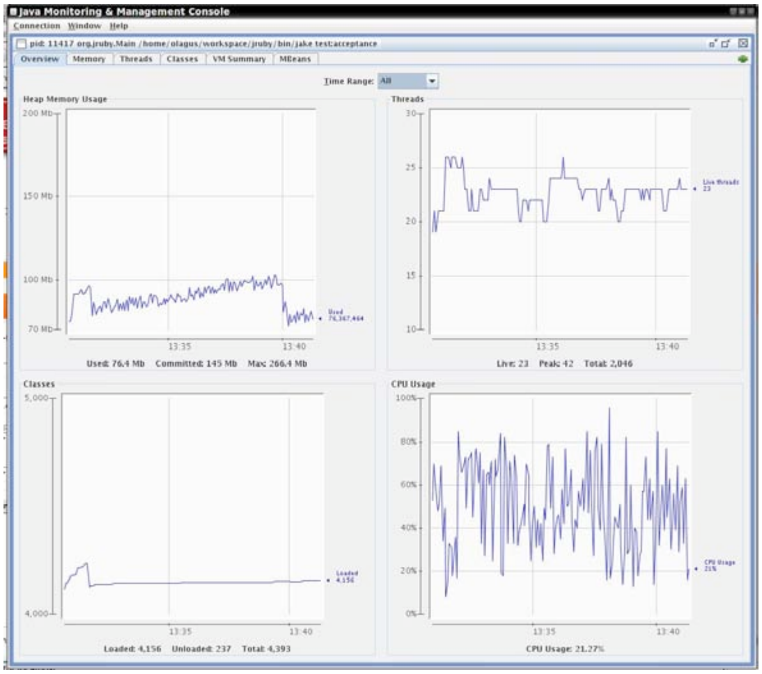
jmock vs mocha cc



jmock has 7.2 times the complexity of mocha

jruby runs ruby on rails

fun with jruby



pid: 11417 org.jruby.Main /home/olagus/workspace/jruby/bin/jake test:acceptance

Time Range: All

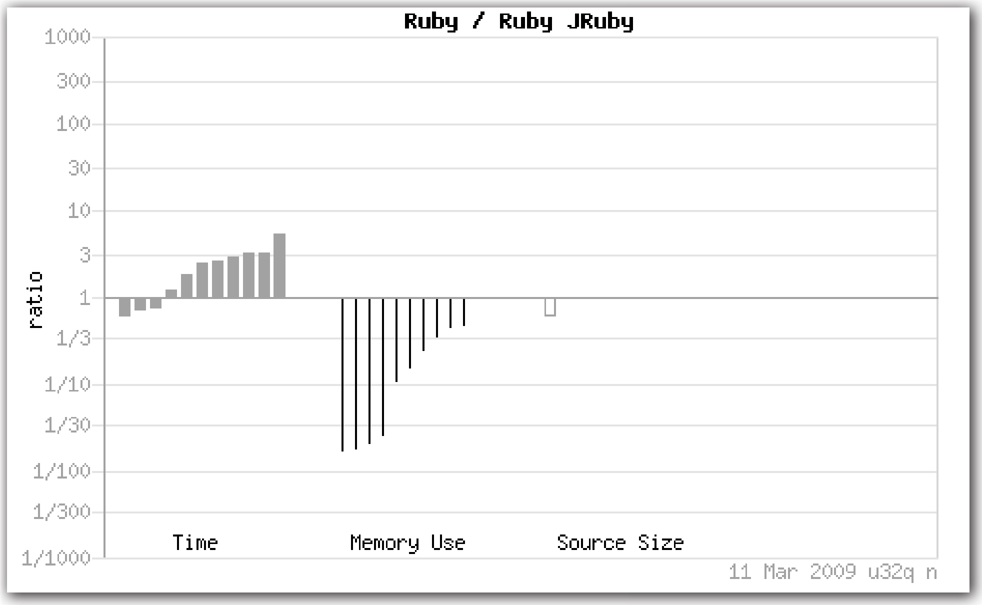
Used: 76.4 Mb Committed: 145 Mb Max: 266.4 Mb

Live: 23 Peak: 42 Total: 2,046

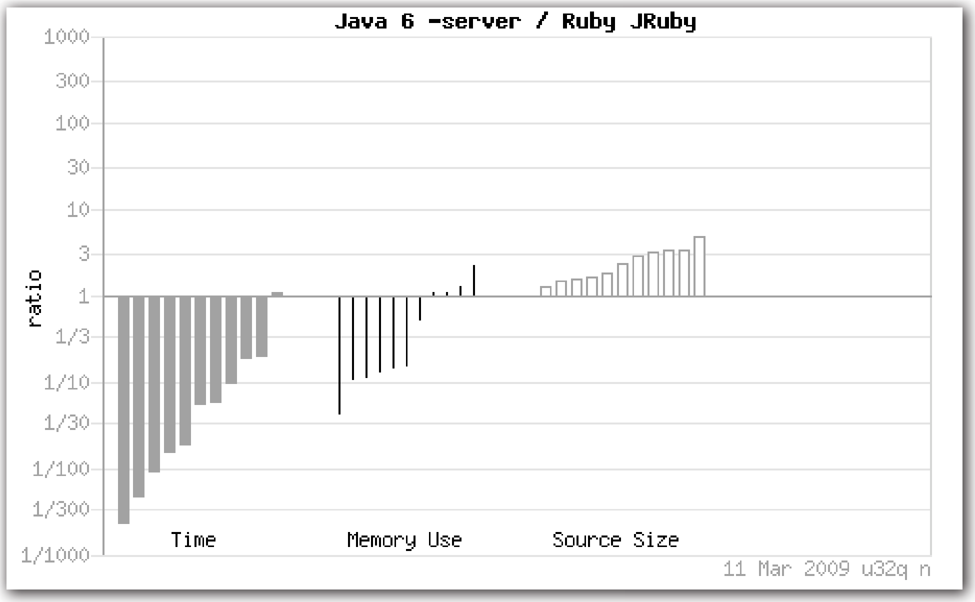
Loaded: 4,156 Unloaded: 237 Total: 4,393

CPU Usage: 21.27%

benchmarks: ruby vs jruby



benchmark: java vs jruby

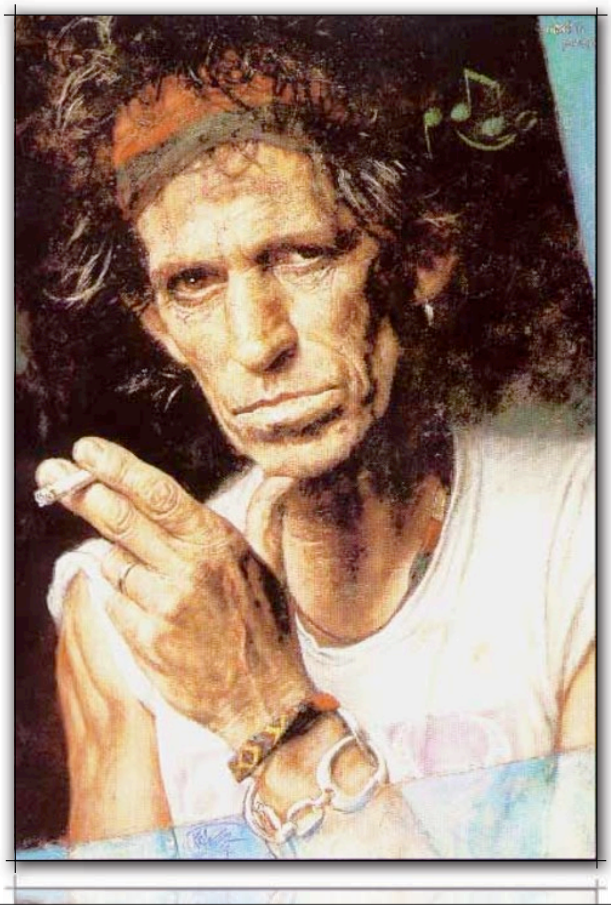


remember, back in 1997...

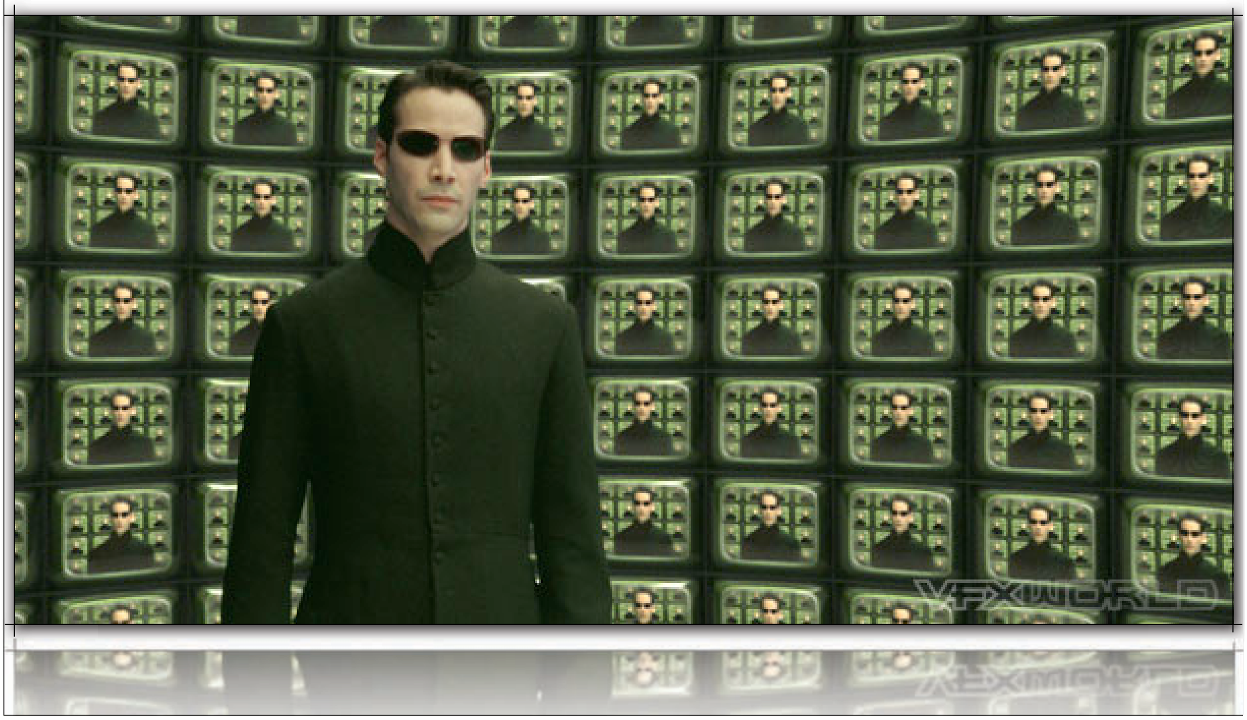
java was considered too slow for "serious development" make it right...

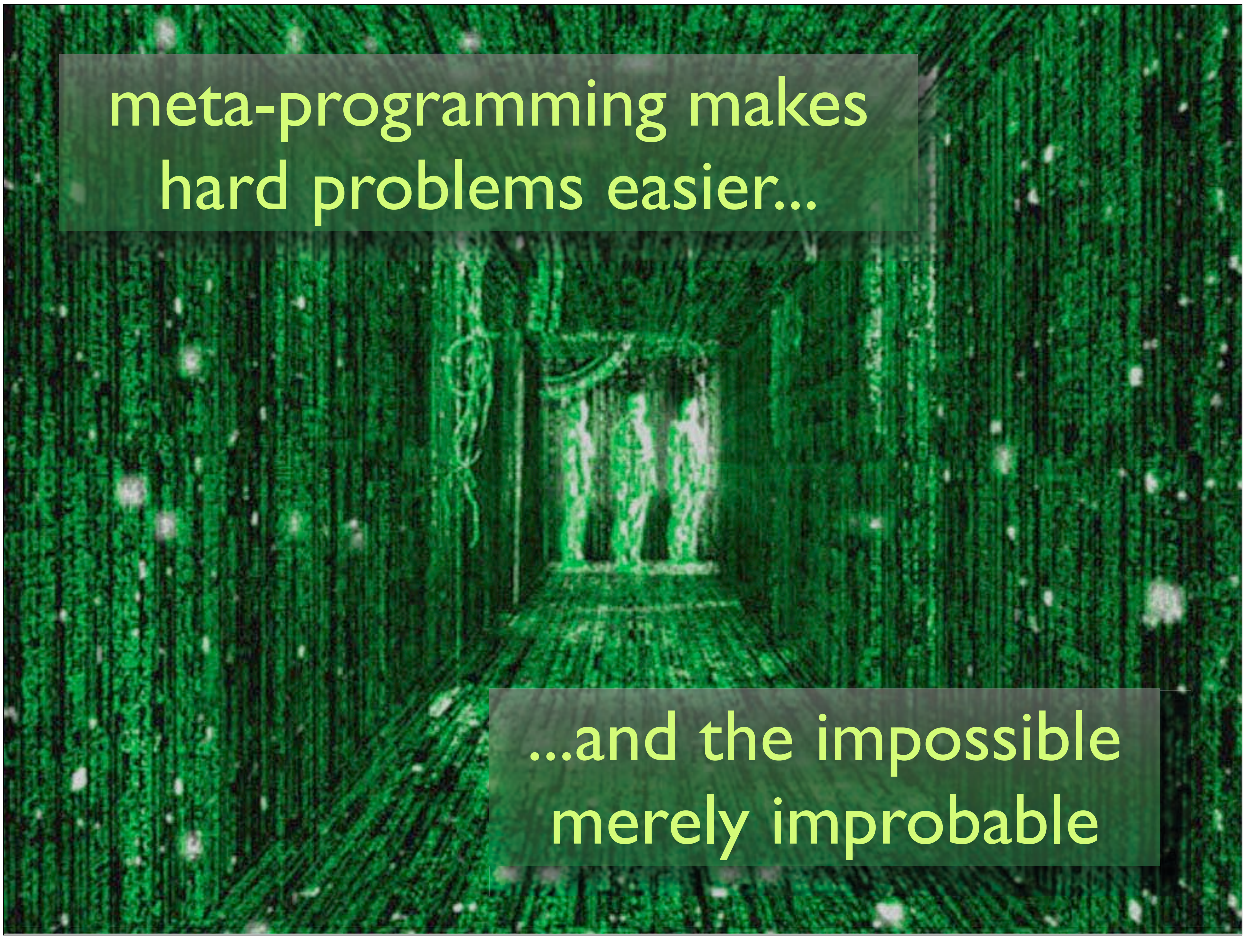
...then make it fast

java in 2008 ==



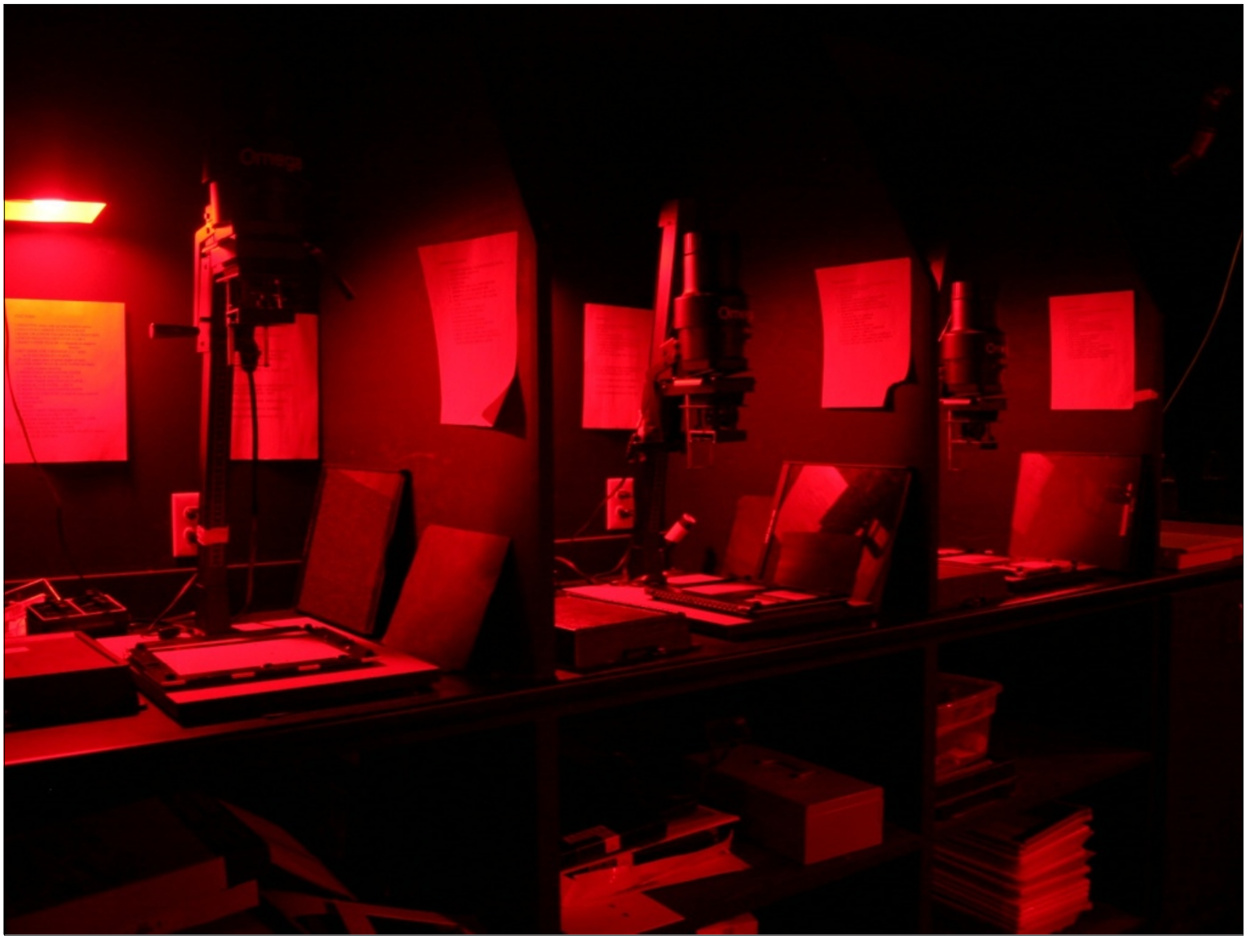
/j?ruby/

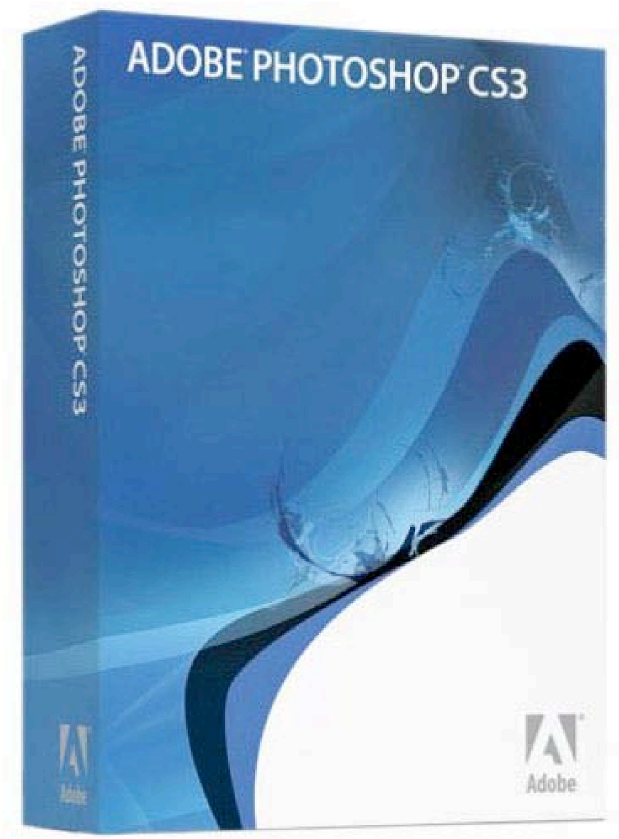




meta-programming makes hard problems easier...

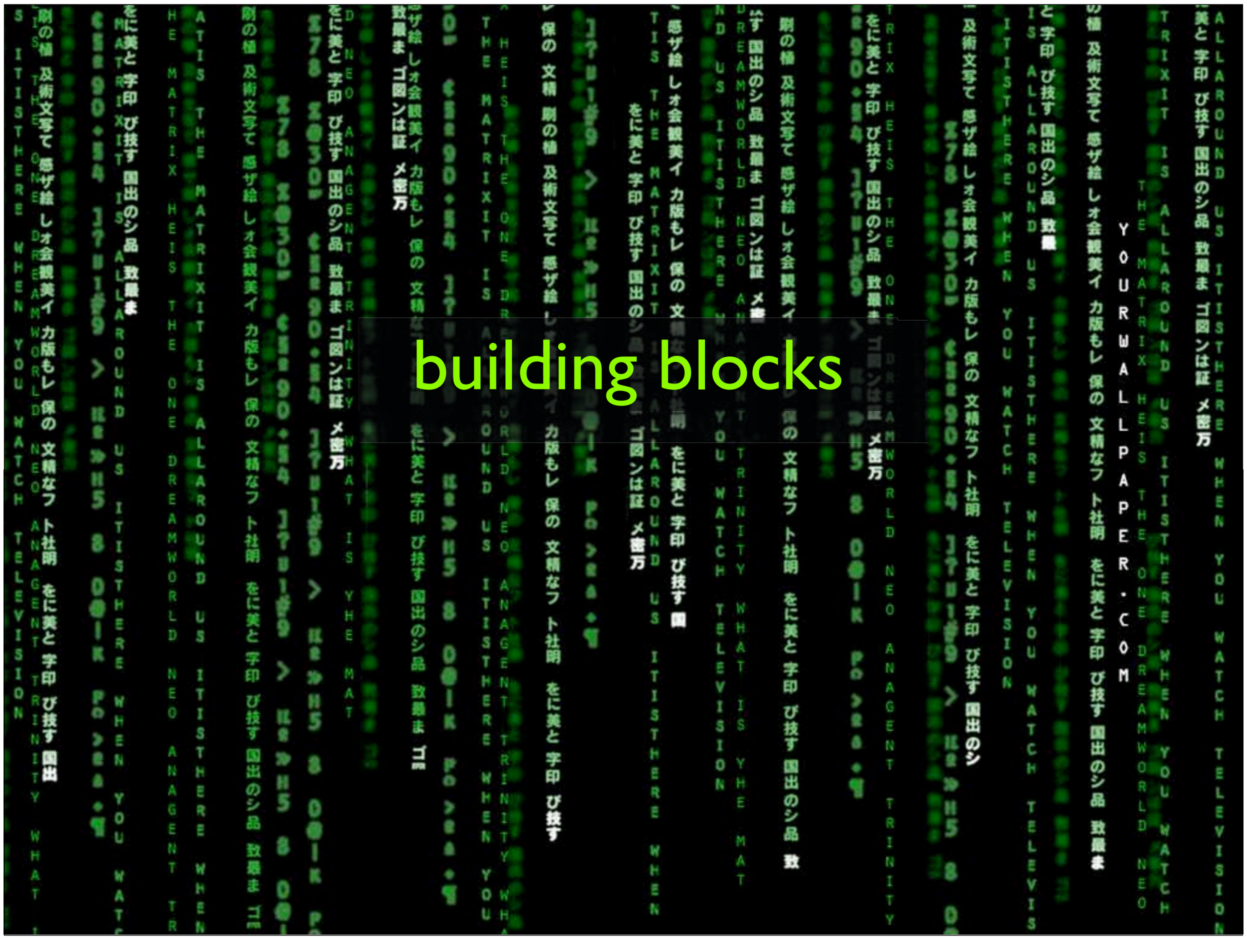
...and the impossible merely improbable


Omega

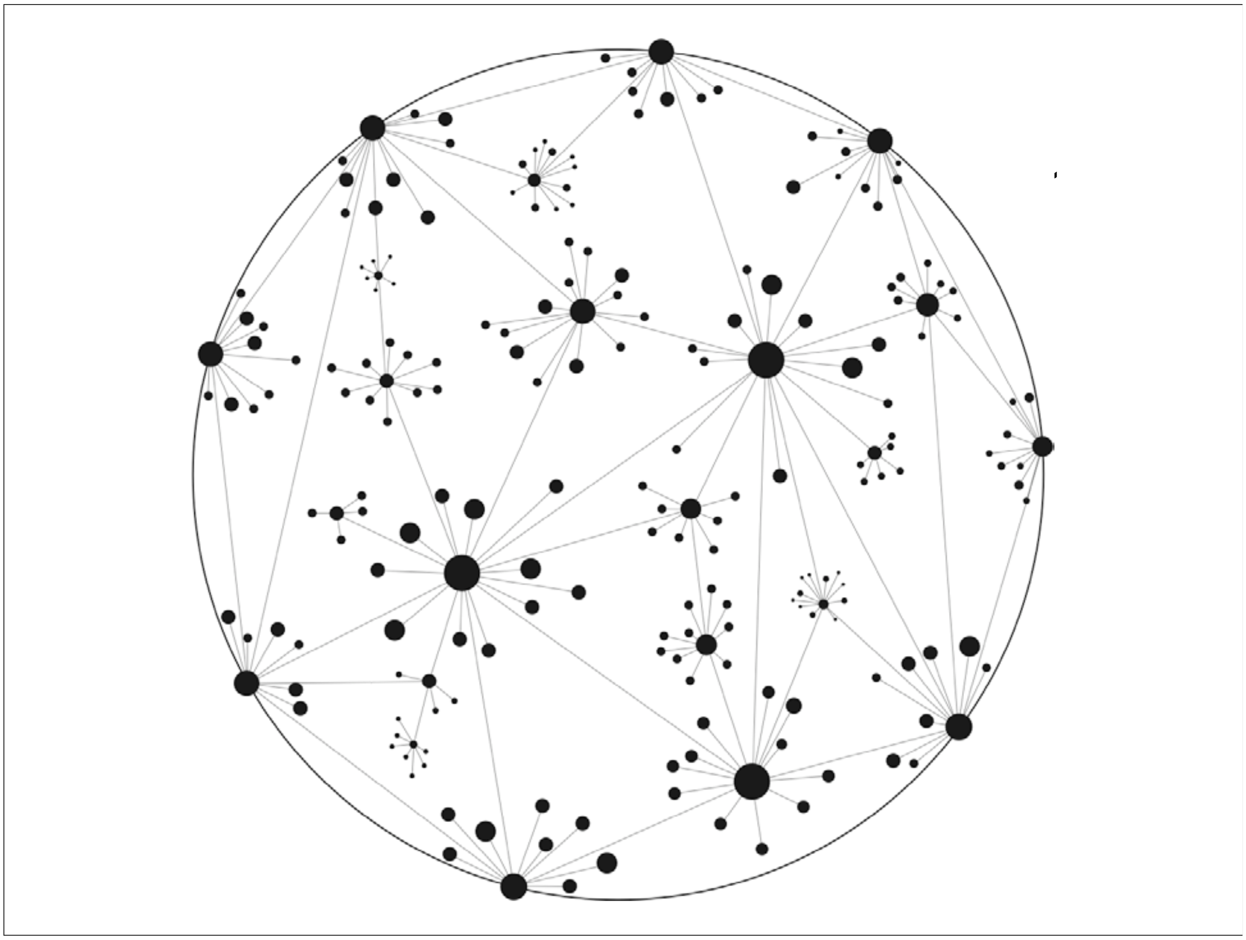


sapir-whorf hypothesis

language affects the capabilities of thoughts



building blocks



features from weaker languages can be synthesized in more powerful languages

all computation in ruby

binding names to objects (assignment)

primitive control structures (if/else, while)

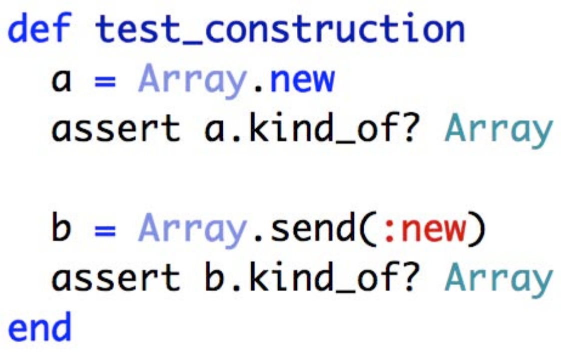
sending messages to objects

messages

```
def test_messages_equal_method_calls
  tagline = "Unfortunately, no one can be told what the Matrix is."
  assert tagline[0..12].downcase == "unfortunately"
  assert tagline[0..12].send(:downcase) == "unfortunately"
  assert tagline[0..12].__send__(:downcase) == 'unfortunately'
  assert tagline[0..12].send("downcase".to_sym) == 'unfortunately'
end
```

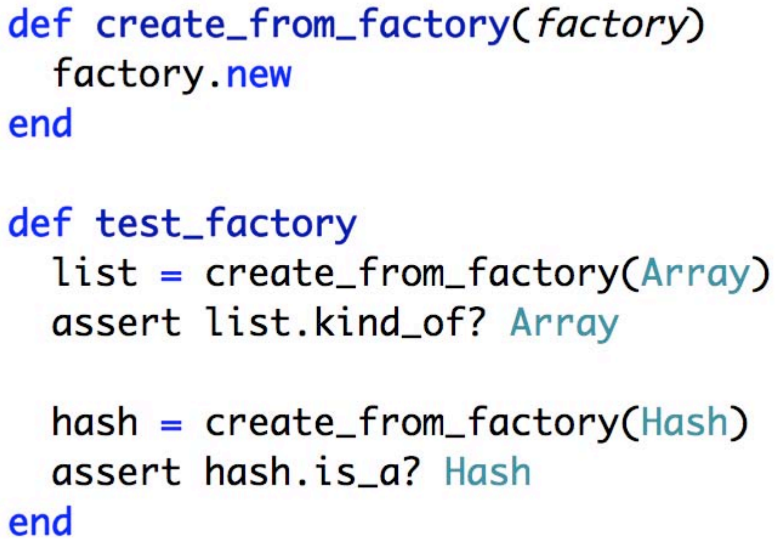
reflection

construction isn’t special

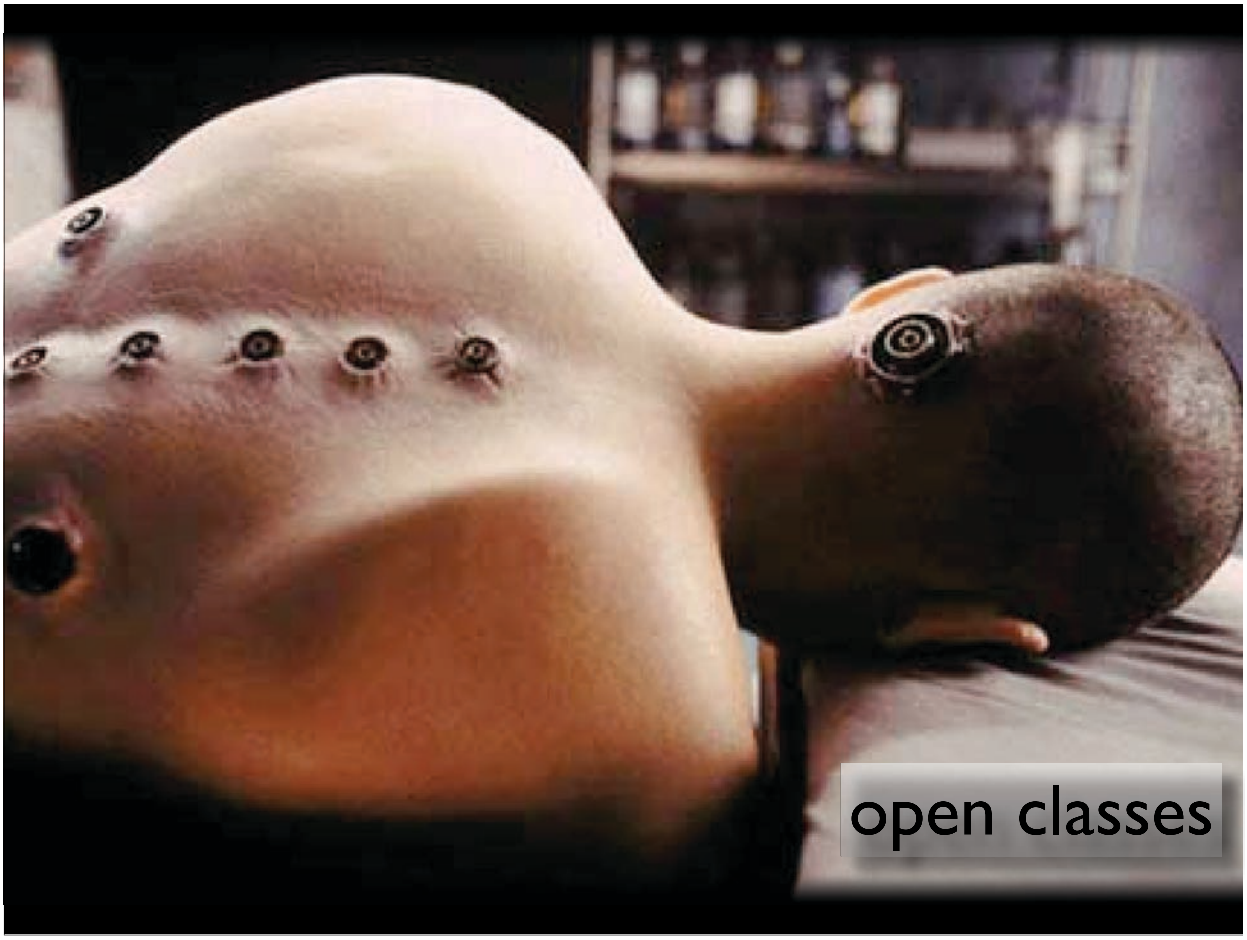


```
def test_construction
  a = Array.new
  assert a.kind_of? Array

  b = Array.send(:new)
  assert b.kind_of? Array
end
```

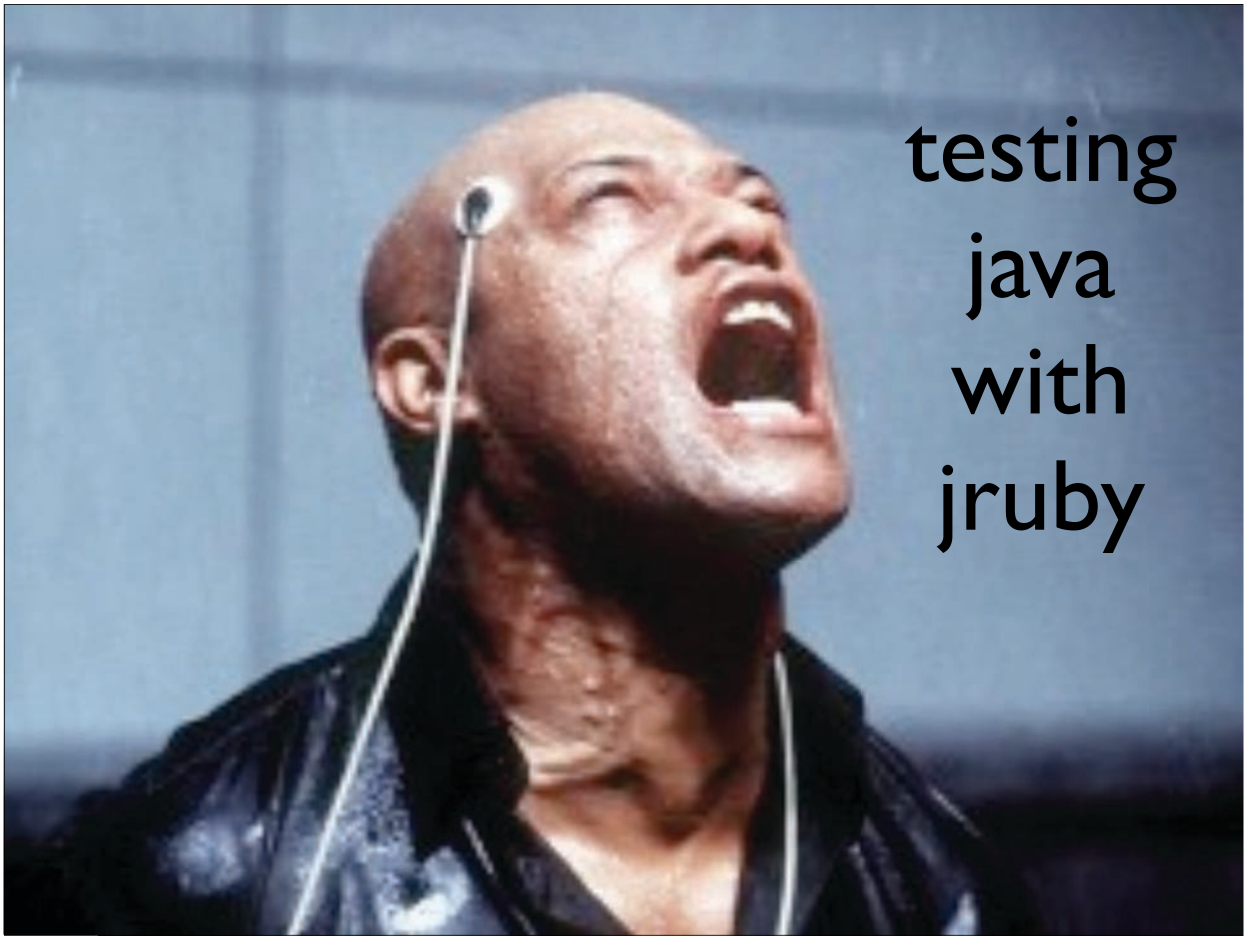
factory “design pattern”



```
def create_from_factory(factory)
  factory.new
end

def test_factory
  list = create_from_factory(Array)
  assert list.kind_of? Array

  hash = create_from_factory(Hash)
  assert hash.is_a? Hash
end
```



open classes



testing java with jruby

}

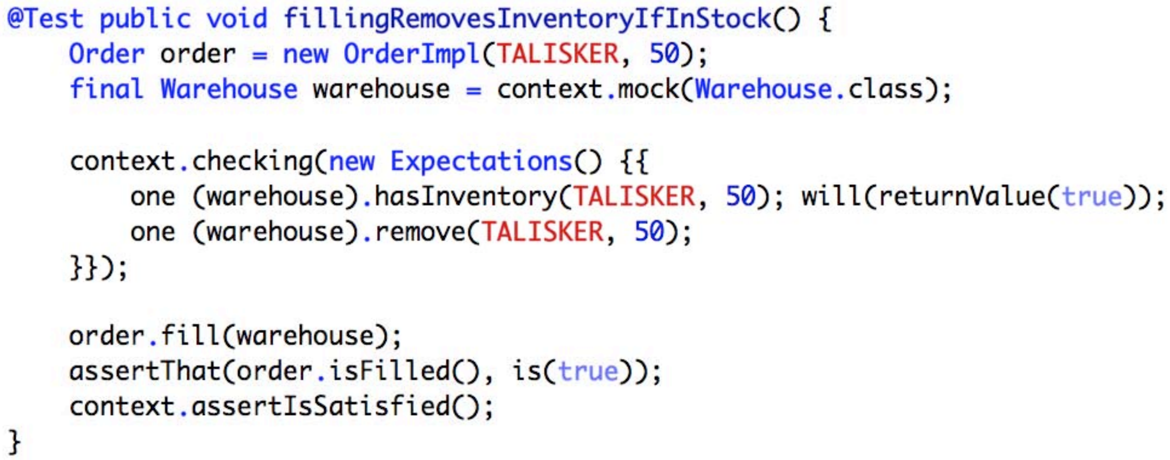
the java part

```
public interface Order {
    void fill(Warehouse warehouse);

    boolean isFilled();
}
public interface Warehouse {
    public void add(String item, int quantity);

    int getInventory(String product);

    boolean hasInventory(String product, int quantity);

    void remove(String product, int quantity);
}
```

testing fill()

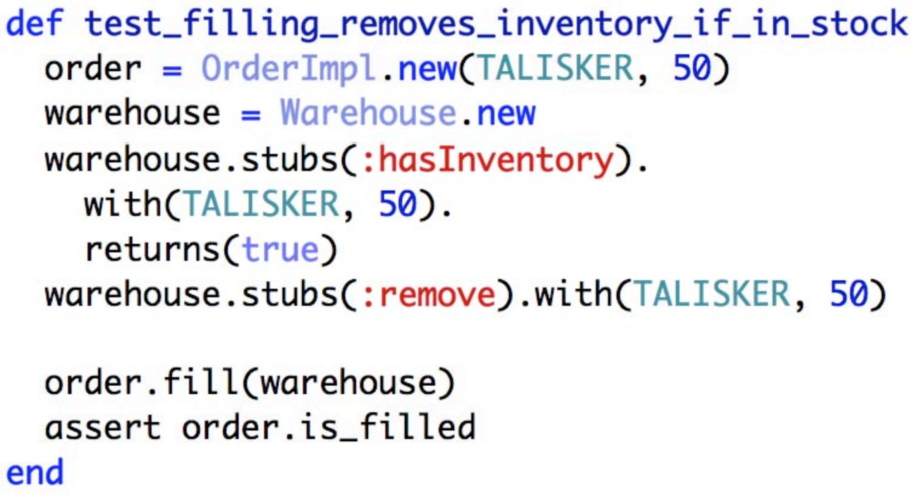
```
public void fill(Warehouse warehouse) {
    if (warehouse.hasInventory(_product, _quantity)) {
        warehouse.remove(_product, _quantity);
        _filled = true;
    } else
        _filled = false;

}
```

jmock



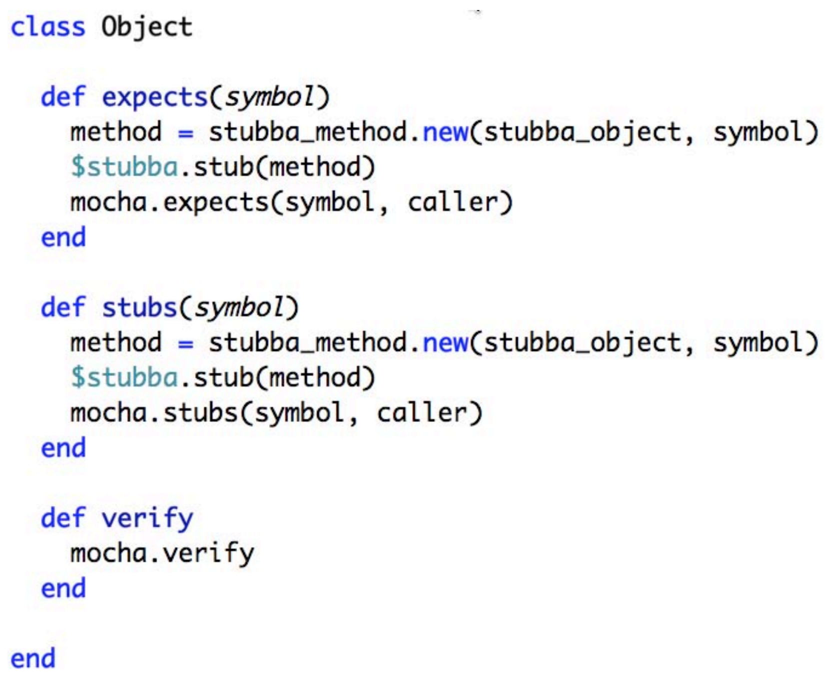
```
@Test public void fillingRemovesInventoryIfInStock() {
    Order order = new OrderImpl(TALISKER, 50);
    final Warehouse warehouse = context.mock(Warehouse.class);

    context.checking(new Expectations() {{
        one (warehouse).hasInventory(TALISKER, 50); will(returnValue(true));
        one (warehouse).remove(TALISKER, 50);
    }});

    order.fill(warehouse);
    assertThat(order.isFilled(), is(true));
    context.assertIsSatisfied();
}
```

mocha



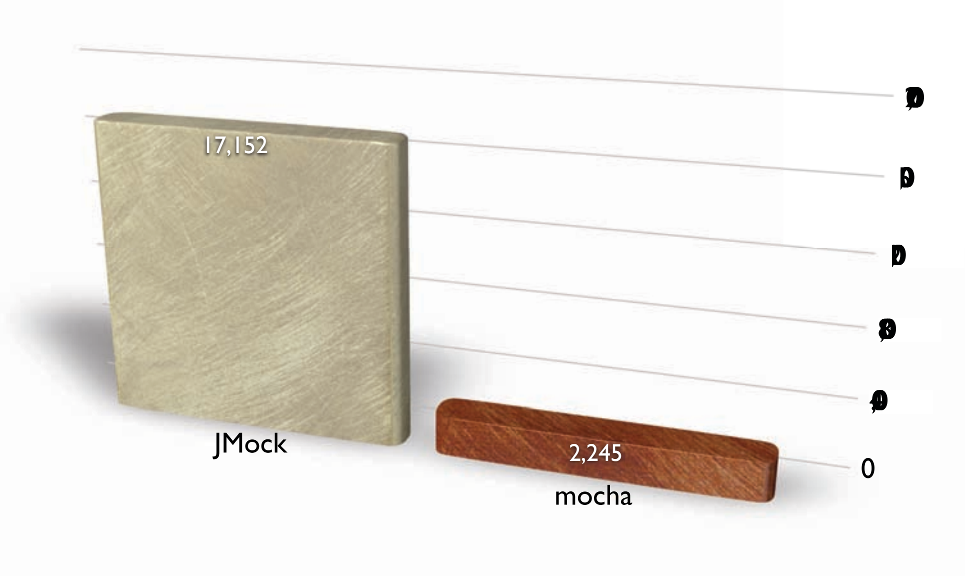
```
def test_filling_removes_inventory_if_in_stock
  order = OrderImpl.new(TALISKER, 50)
  warehouse = Warehouse.new
  warehouse.stubs(:hasInventory).
    with(TALISKER, 50).
    returns(true)
  warehouse.stubs(:remove).with(TALISKER, 50)

  order.fill(warehouse)
  assert order.is_filled
end
```

what does it take???



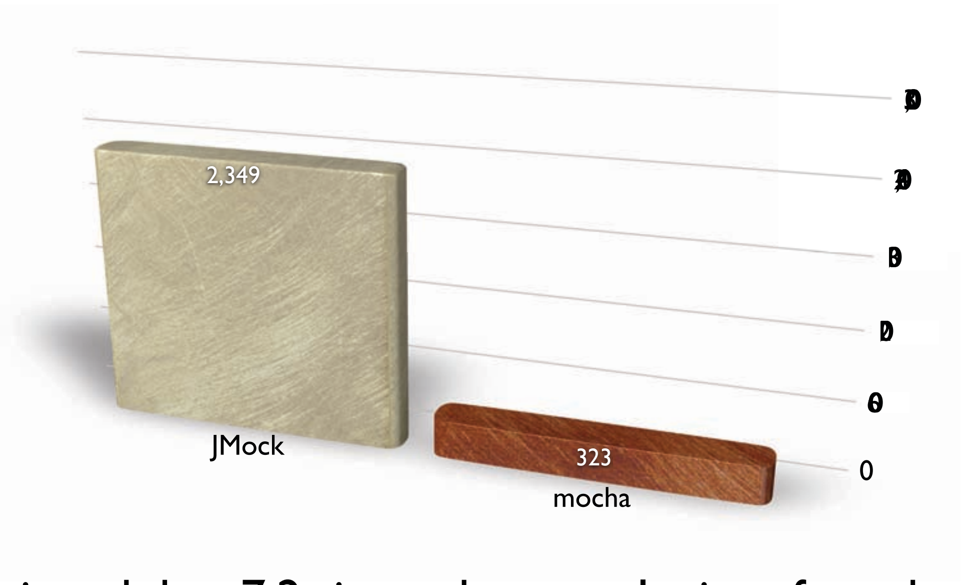
```
class Object

  def expects(symbol)
    method = stubba_method.new(stubba_object, symbol)
    $stubba.stub(method)
    mocha.expects(symbol, caller)
  end

  def stubs(symbol)
    method = stubba_method.new(stubba_object, symbol)
    $stubba.stub(method)
    mocha.stubs(symbol, caller)
  end

  def verify
    mocha.verify
  end

end
```

jmock vs mocha loc



jmock has 7.5 times as many lines of code

jmock vs mocha cc



jmock has 7.2 times the complexity of mocha

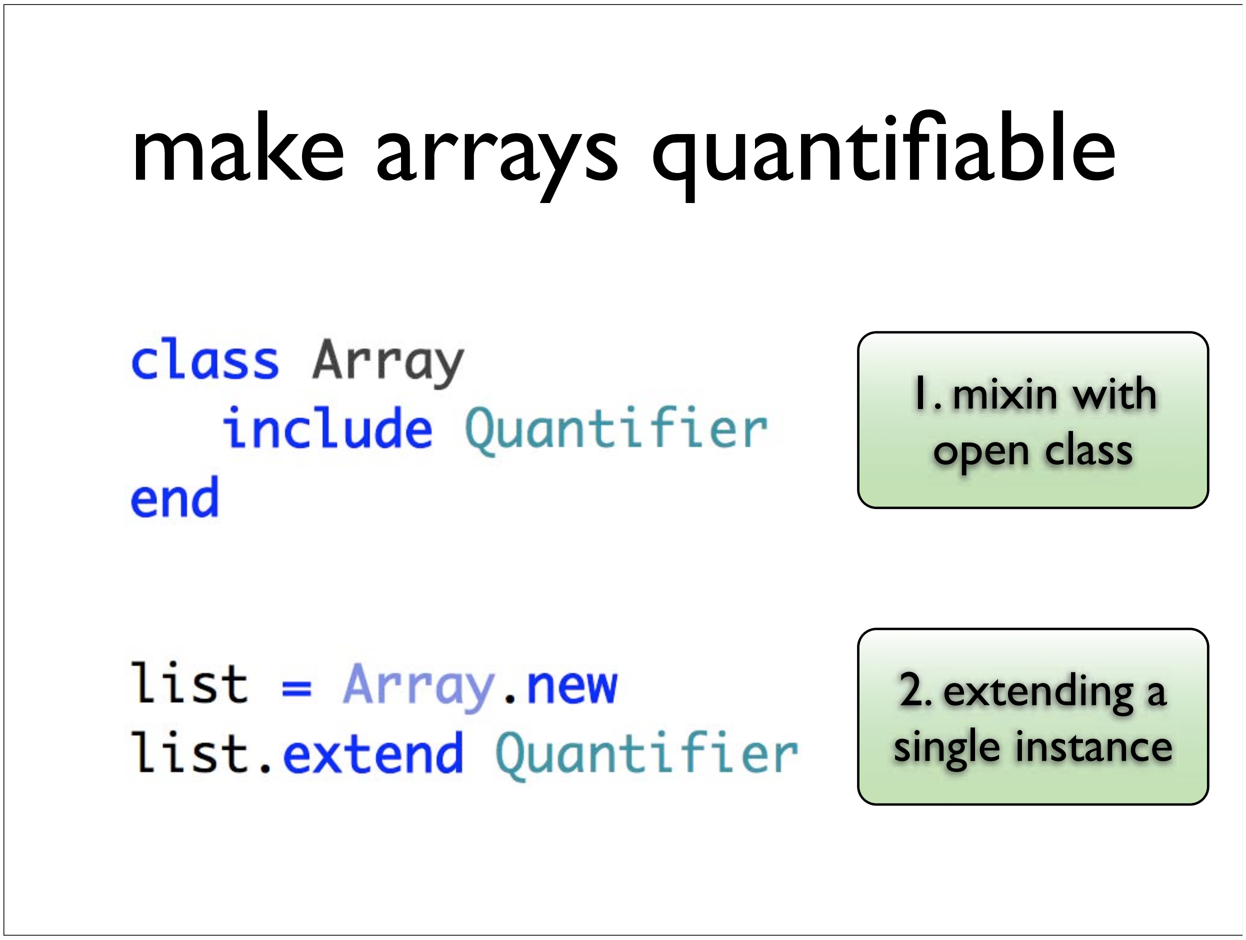
modules

block syntax

```
def use_block flag
  yield if flag
end

use_block(1 == 1) { puts "What is the Matrix?"}

use_block(1 == 2) do
  puts "The answer is out there, Neo"
end
```

`quantifier` module

```
module Quantifier
  def any?
    each { |x| return true if yield x }
    false
  end

  def all?
    each { |x| return false if not yield x }
    true
  end
end
```



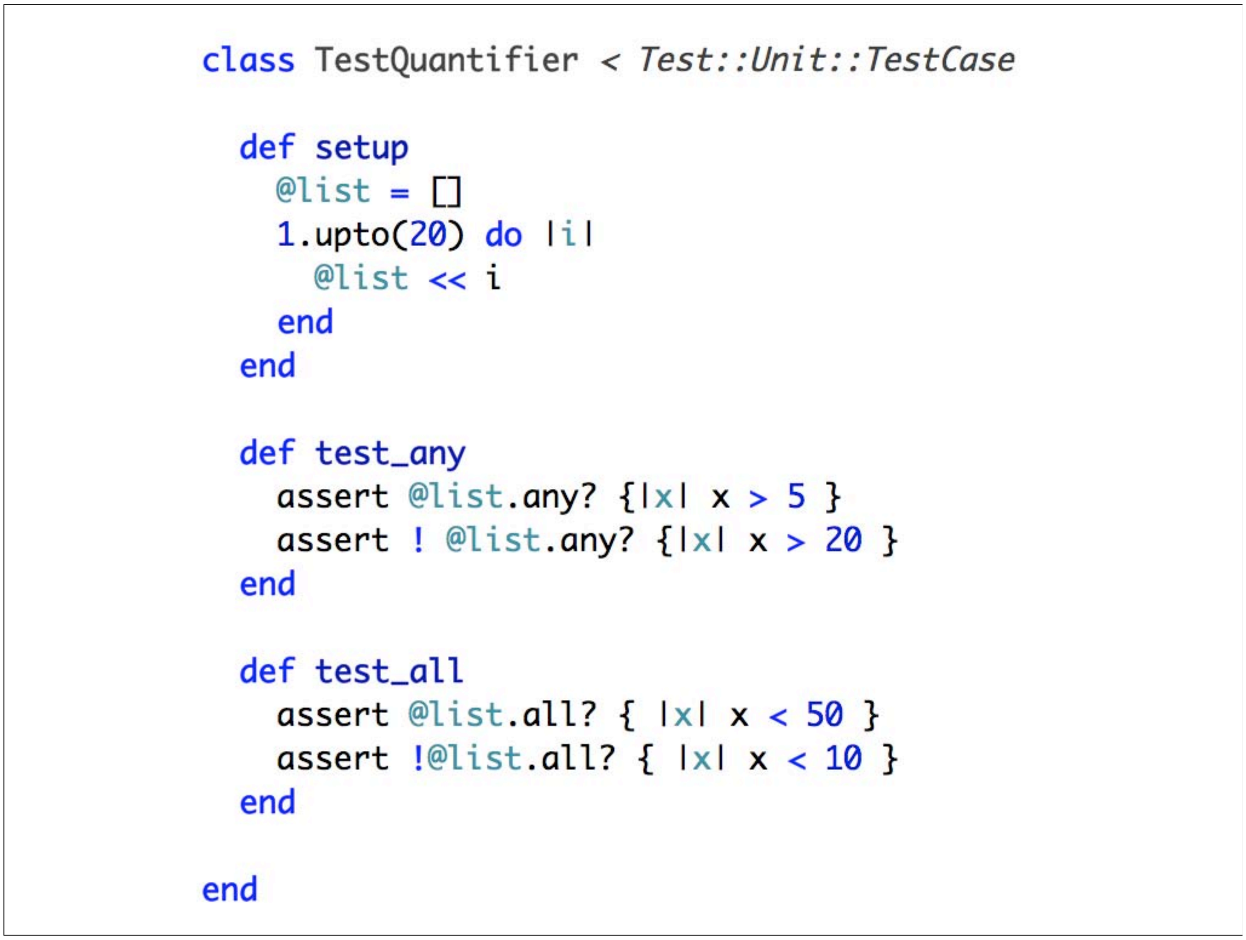
make arrays quantifiable

```
class Array
  include Quantifier
end
```

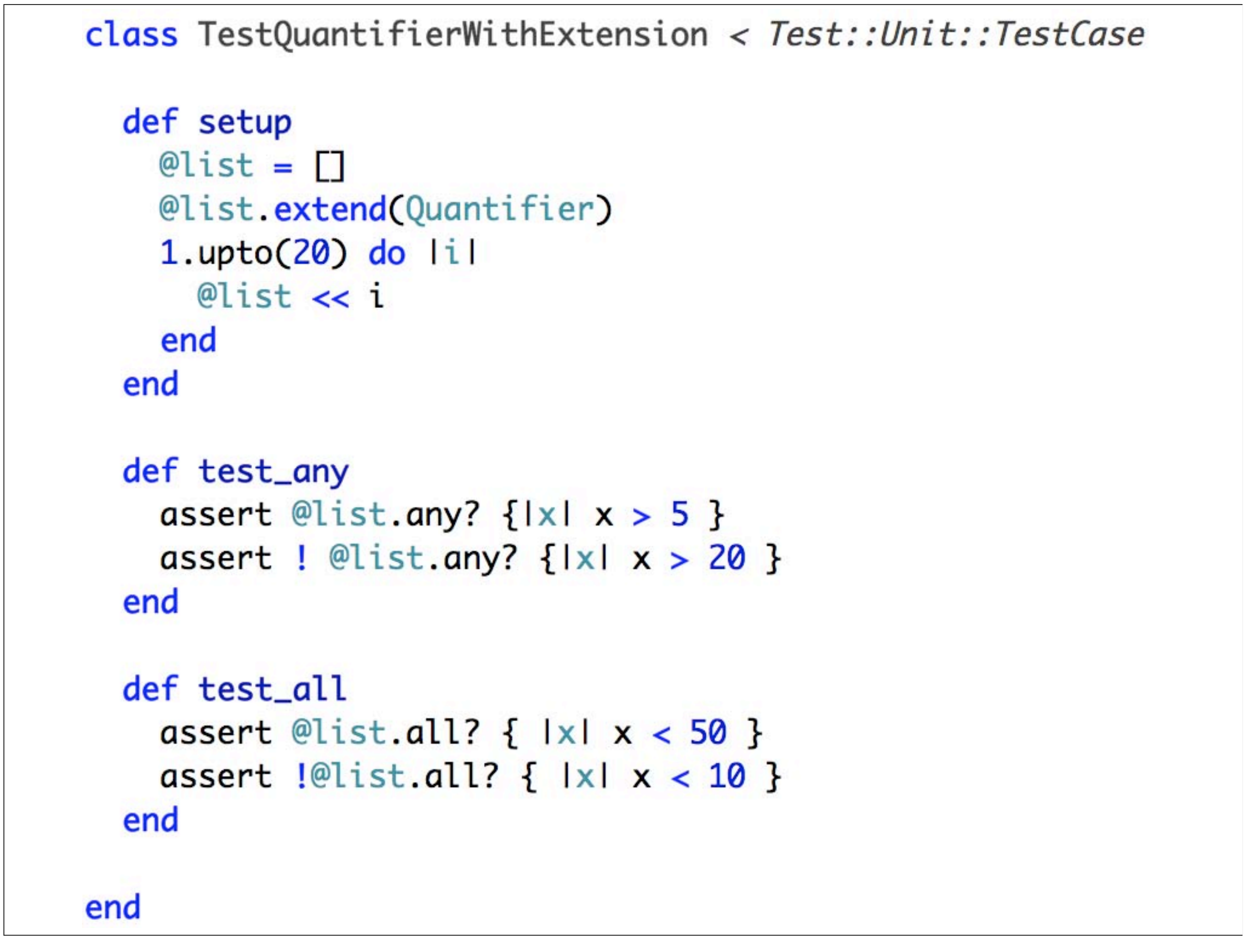
1. mixin with open class

```
list = Array.new
list.extend Quantifier
```

2. extending a single instance

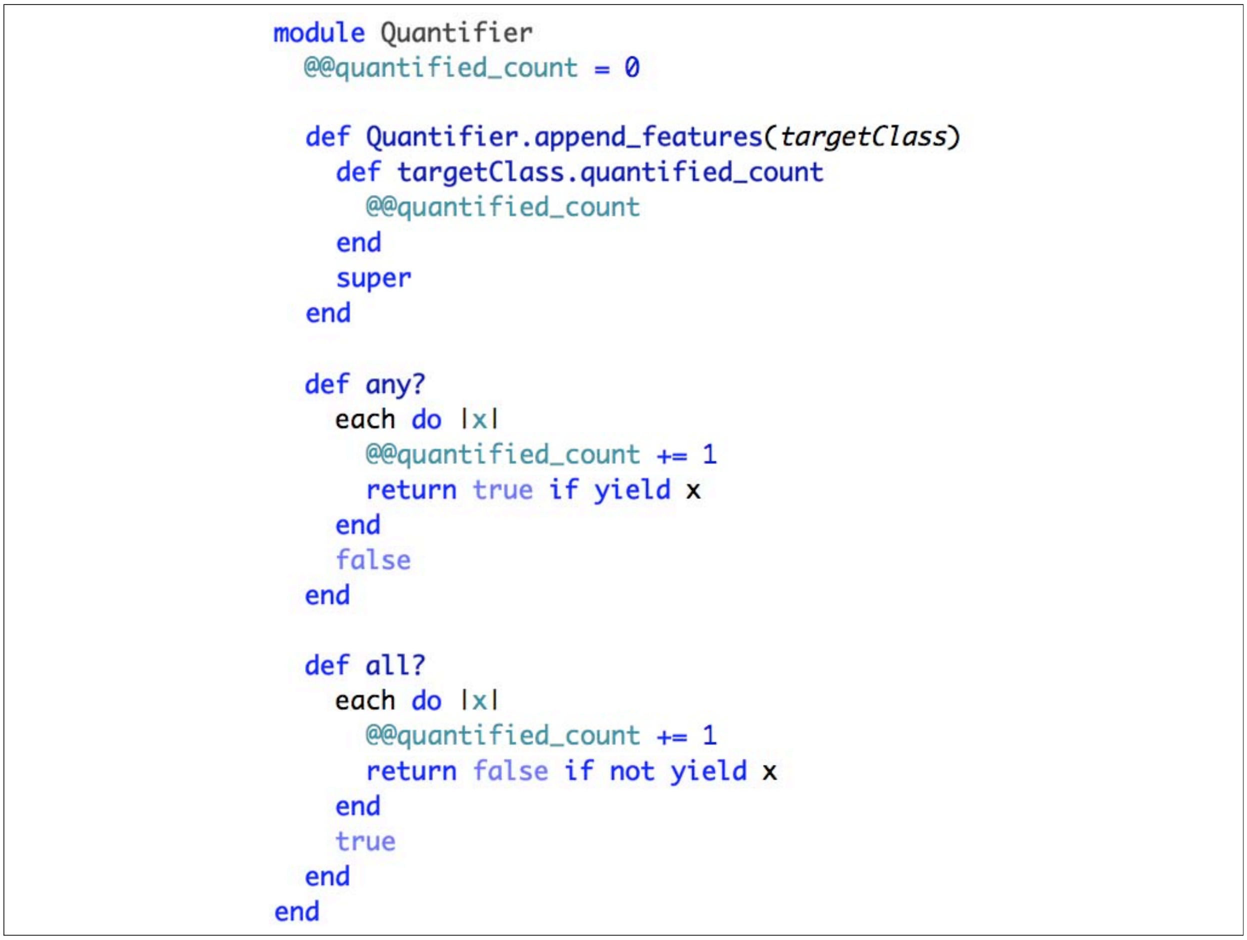


```
class TestQuantifier < Test::Unit::TestCase

  def setup
    @list = []
    1.upto(20) do |i|
      @list << i
    end
  end

  def test_any
    assert @list.any? {|x| x > 5 }
    assert ! @list.any? {|x| x > 20 }
  end

  def test_all
    assert @list.all? { |x| x < 50 }
    assert !@list.all? { |x| x < 10 }
  end

end
```

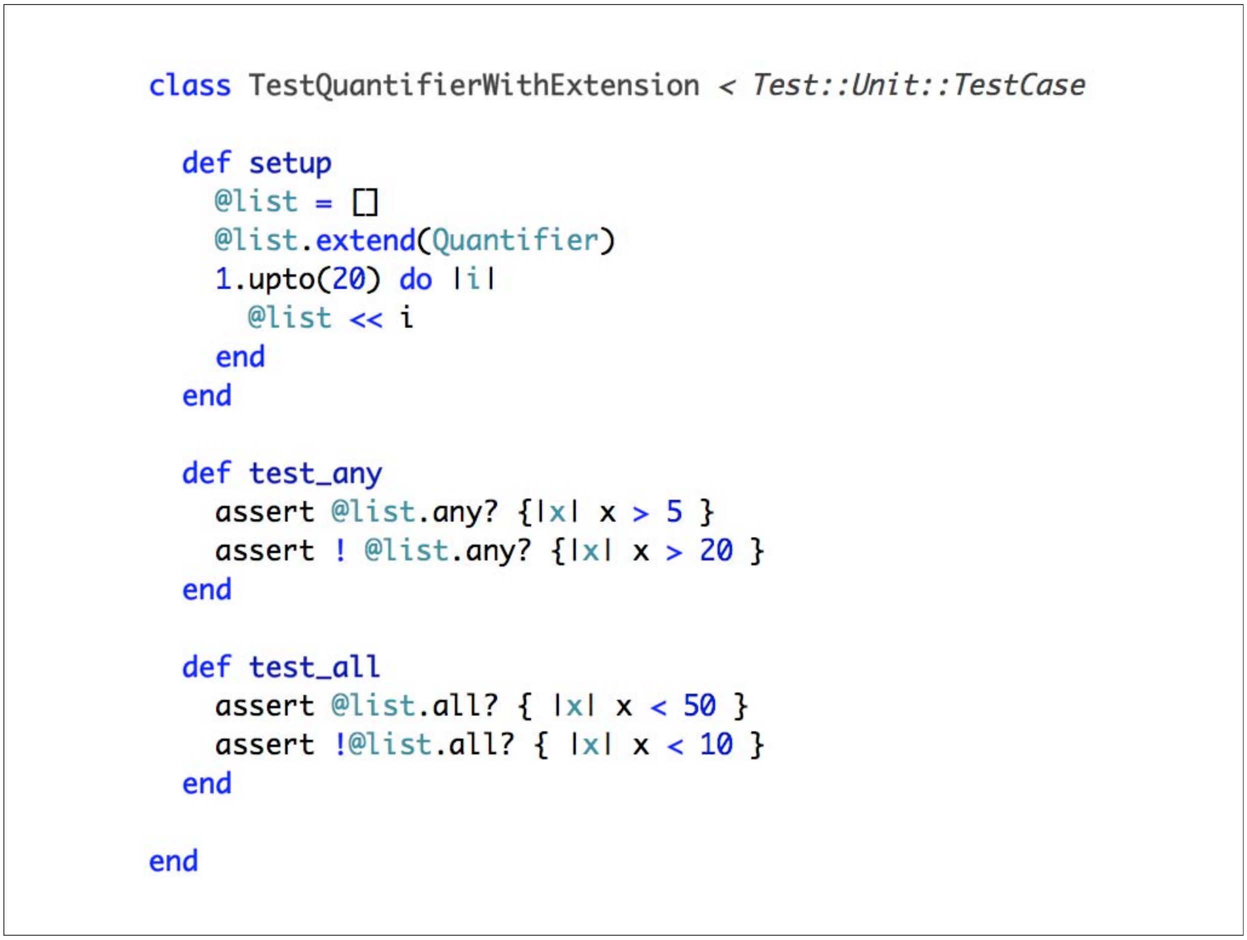


```
class TestQuantifierWithExtension < Test::Unit::TestCase

  def setup
    @list = []
    @list.extend(Quantifier)
    1.upto(20) do |i|
      @list << i
    end
  end

  def test_any
    assert @list.any? {|x| x > 5 }
    assert ! @list.any? {|x| x > 20 }
  end

  def test_all
    assert @list.all? { |x| x < 50 }
    assert !@list.all? { |x| x < 10 }
  end

end
```

what if we wanted to count everything we quantified?



```
module Quantifier
  @@quantified_count = 0

  def Quantifier.append_features(targetClass)
    def targetClass.quantified_count
      @@quantified_count
    end
    super
  end

  def any?
    each do |x|
      @@quantified_count += 1
      return true if yield x
    end
    false
  end

  def all?
    each do |x|
      @@quantified_count += 1
      return false if not yield x
    end
    true
  end
end
```

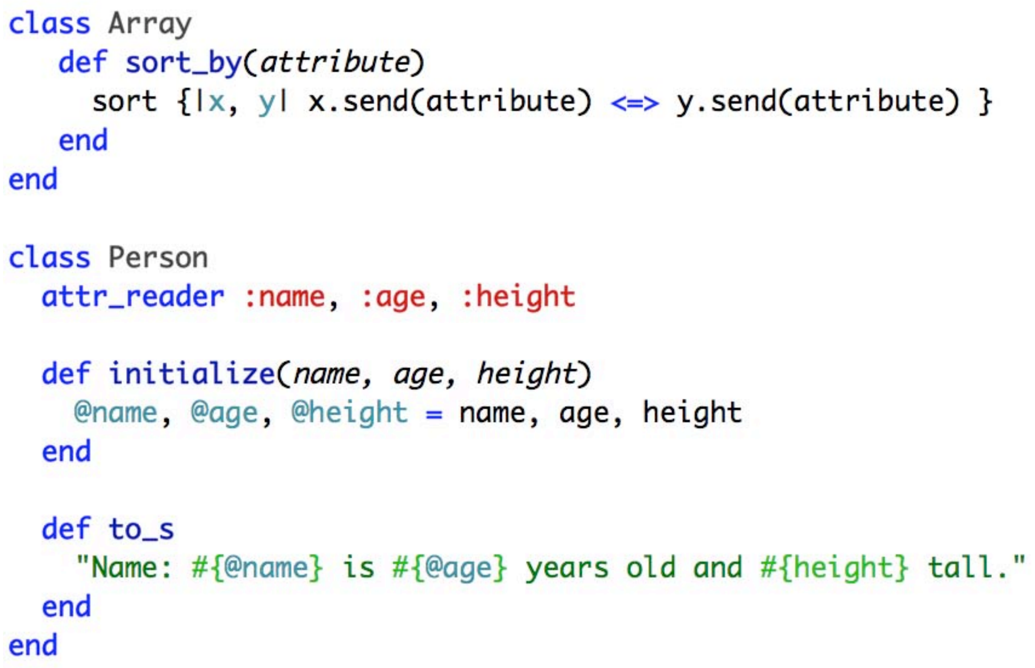


```
class TestQuantifierWithExtension < Test::Unit::TestCase

  def setup
    @list = []
    @list.extend(Quantifier)
    1.upto(20) do |i|
      @list << i
    end
  end

  def test_any
    assert @list.any? {|x| x > 5 }
    assert ! @list.any? {|x| x > 20 }
  end

  def test_all
    assert @list.all? { |x| x < 50 }
    assert !@list.all? { |x| x < 10 }
  end

end
```

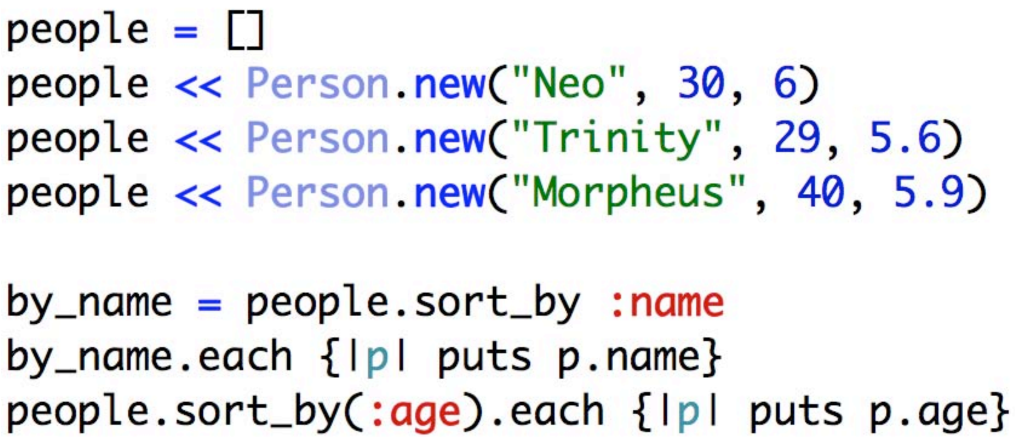
everything is a message

sort on any field of a class

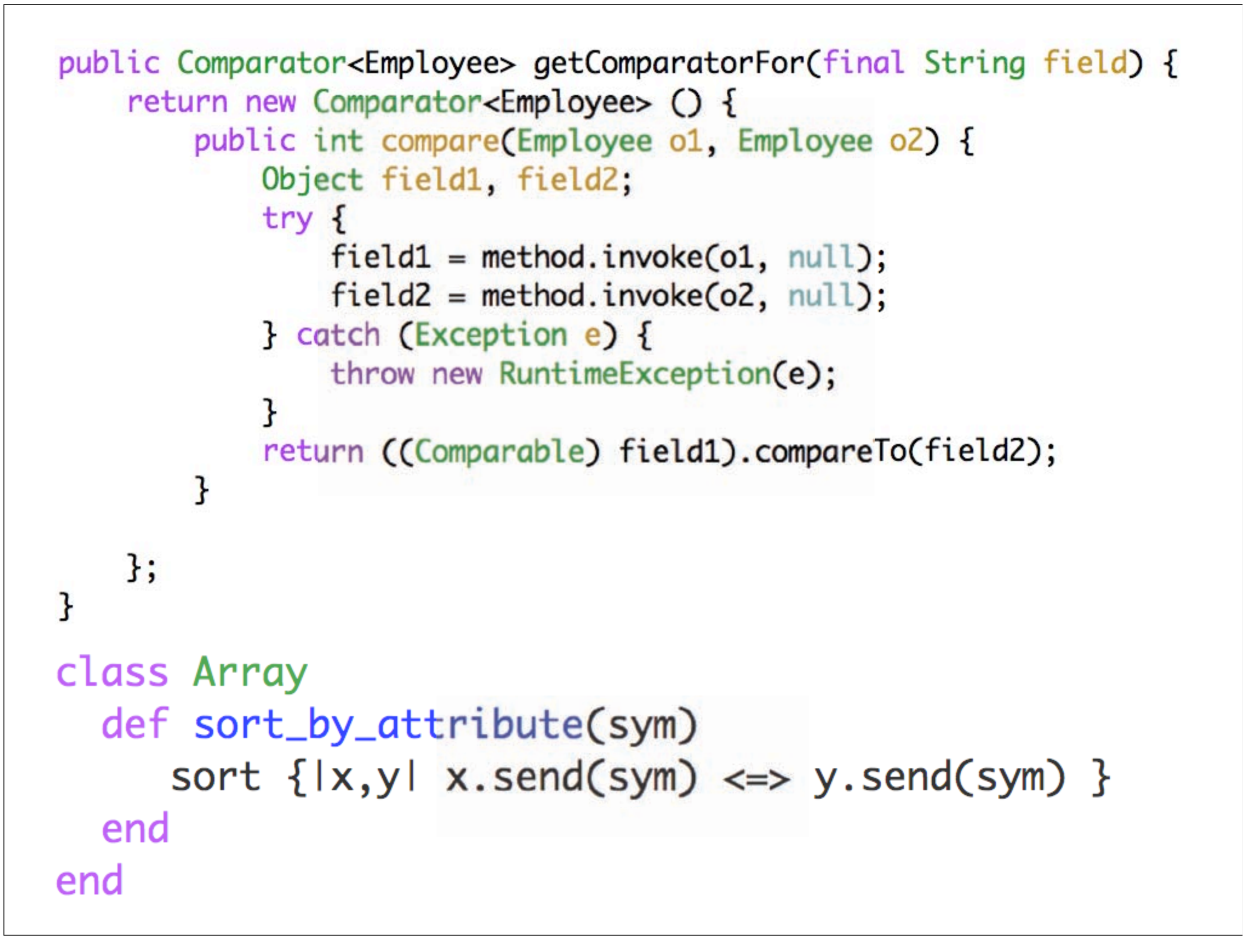
the ruby way



```ruby
class Array
   def sort_by(attribute)
     sort {|x, y| x.send(attribute) <=> y.send(attribute) }
   end
end

class Person
  attr_reader :name, :age, :height

  def initialize(name, age, height)
    @name, @age, @height = name, age, height
  end

  def to_s
    "Name: #{@name} is #{@age} years old and #{height} tall."
  end
end
```



```ruby
people = []
people << Person.new("Neo", 30, 6)
people << Person.new("Trinity", 29, 5.6)
people << Person.new("Morpheus", 40, 5.9)

by_name = people.sort_by :name
by_name.each {|p| puts p.name}
people.sort_by(:age).each {|p| puts p.age}
```

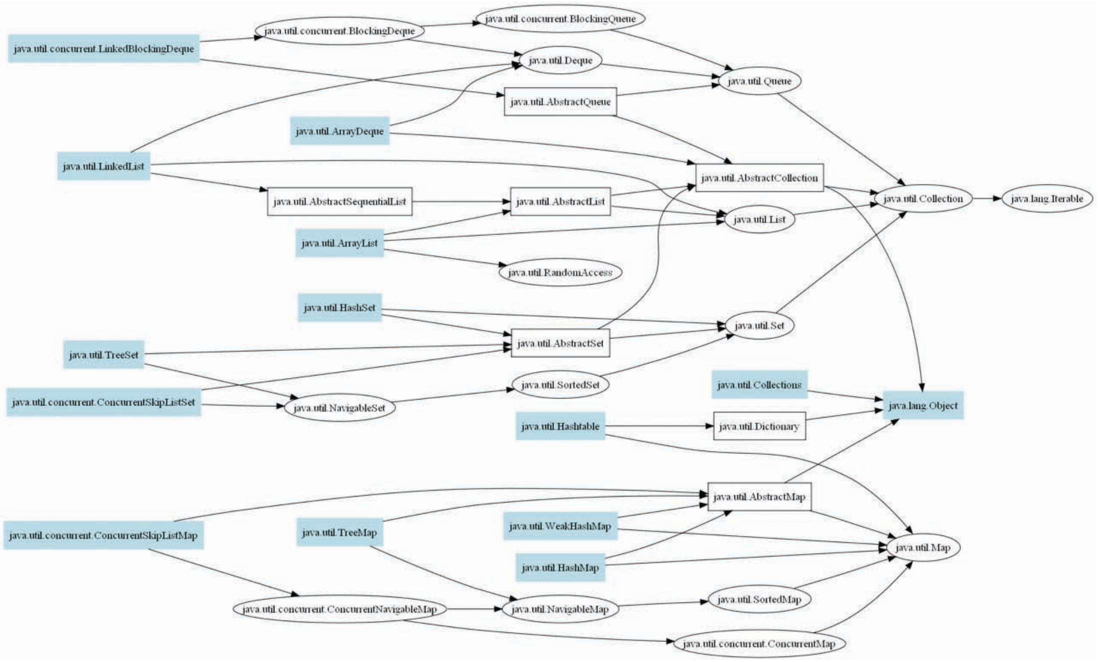


```
public Comparator<Employee> getComparatorFor(final String field) {
    return new Comparator<Employee> () {
        public int compare(Employee o1, Employee o2) {
            Object field1, field2;
            try {
                field1 = method.invoke(o1, null);
                field2 = method.invoke(o2, null);
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
            return ((Comparable) field1).compareTo(field2);
        }

    };
}
```

```
class Array
  def sort_by_attribute(sym)
     sort {|x,y| x.send(sym) <=> y.send(sym) }
  end
end
```

delegation

java’s collection package



ruby’s collections

Array

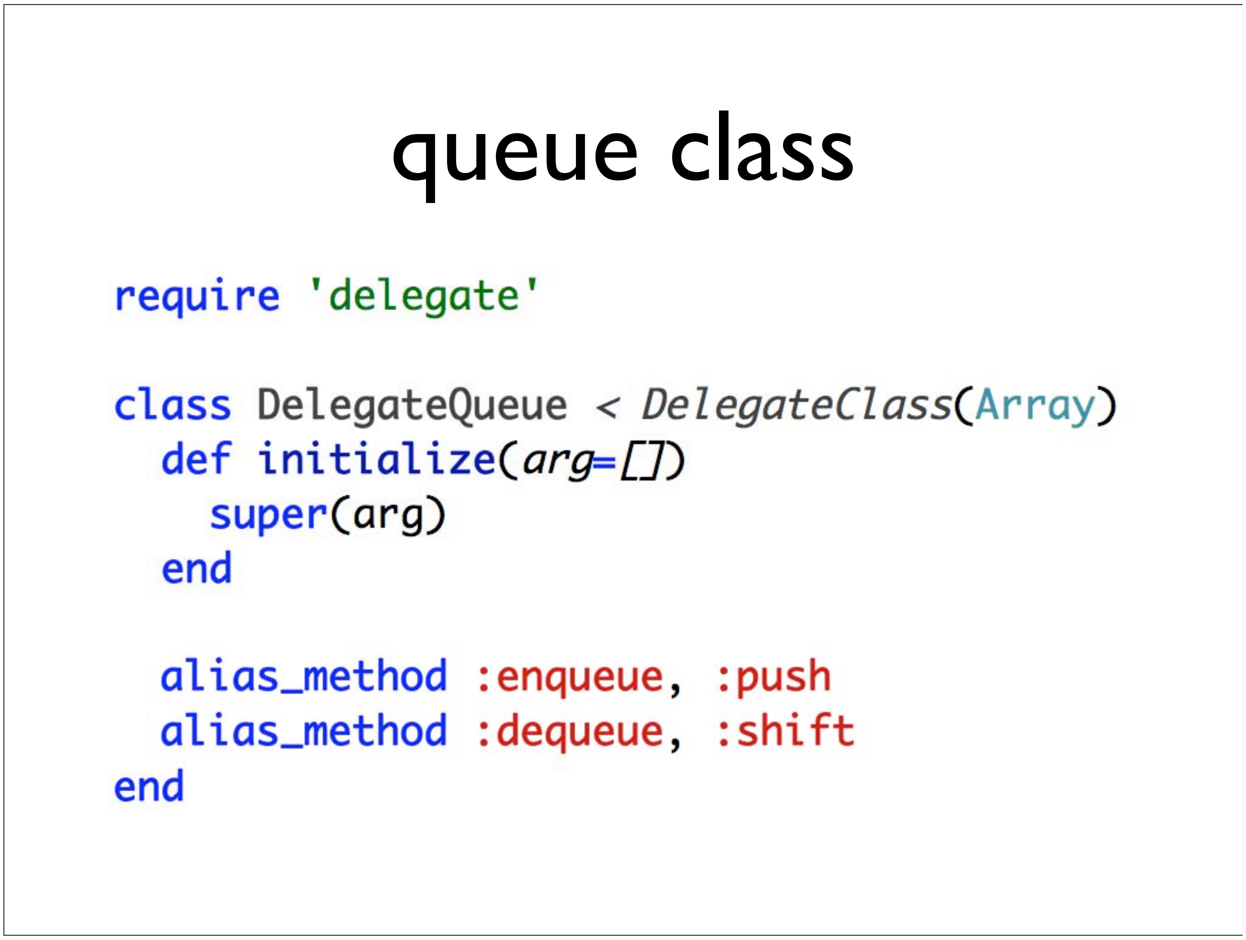
Set

“humane interface”

Hash

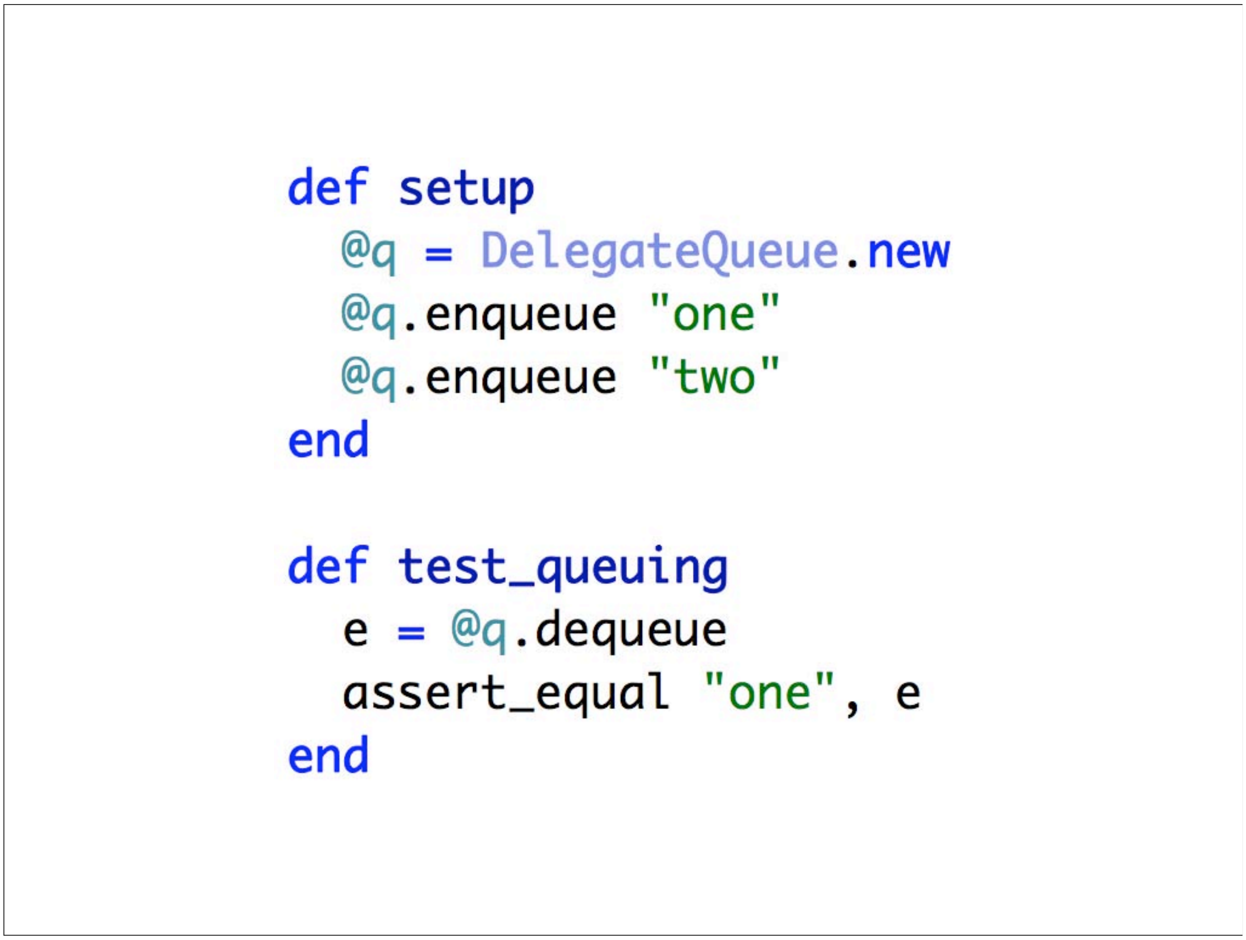
sometimes you want a class to act like another class

but with limitations

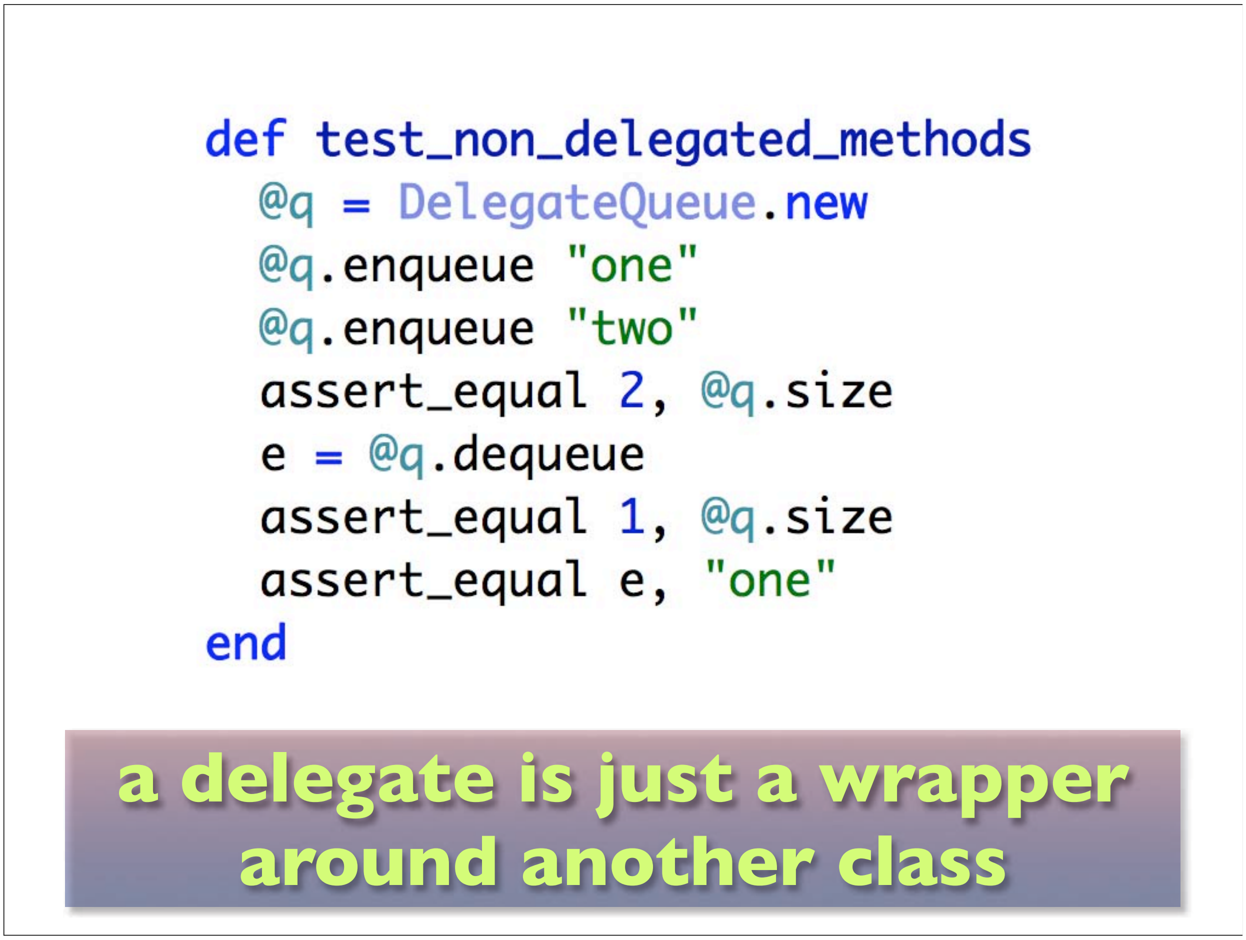


queue class

```ruby
require 'delegate'

class DelegateQueue < DelegateClass(Array)
  def initialize(arg=[])
    super(arg)
  end

  alias_method :enqueue, :push
  alias_method :dequeue, :shift
end
```

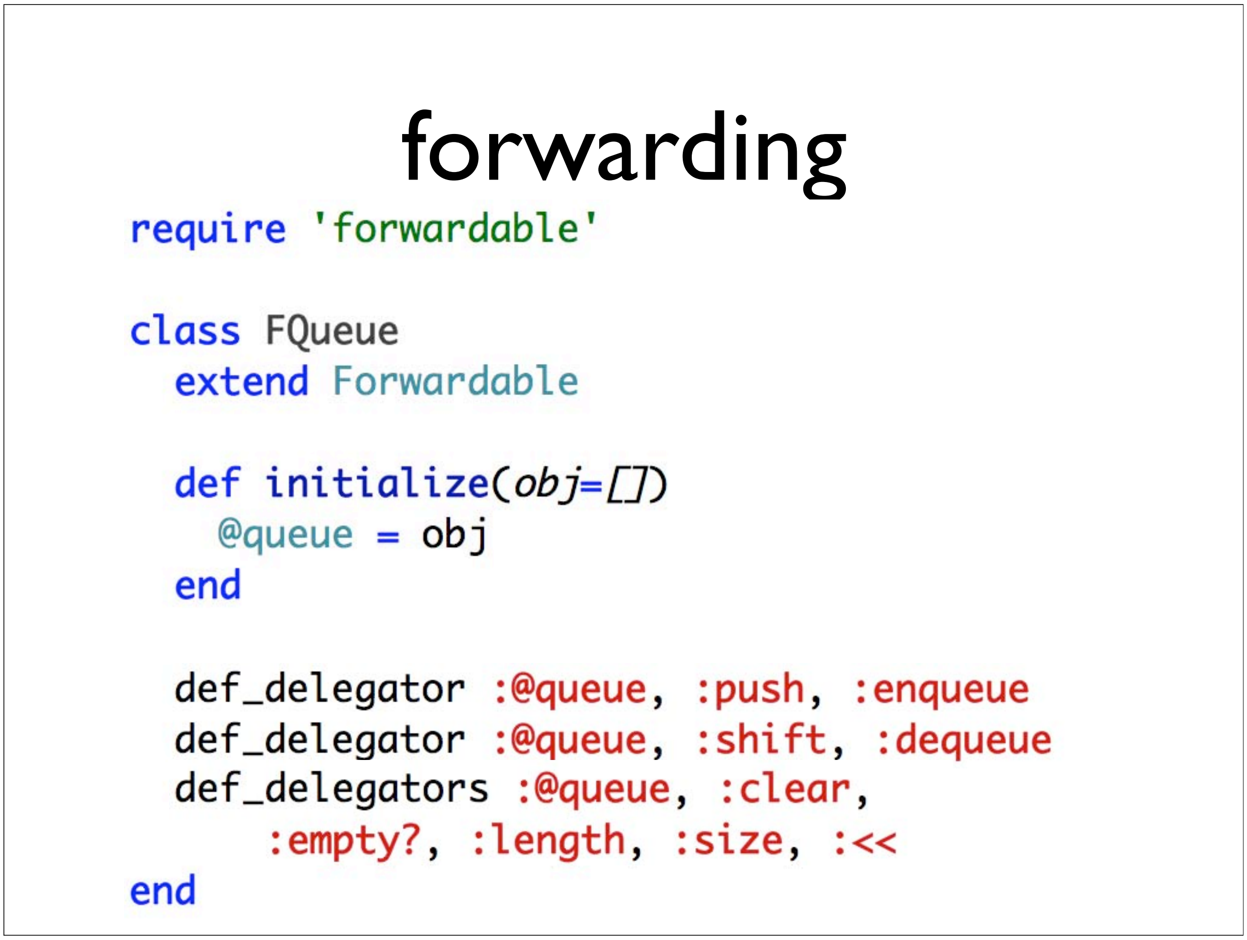


```
def setup
  @q = DelegateQueue.new
  @q.enqueue "one"
  @q.enqueue "two"
end

def test_queuing
  e = @q.dequeue
  assert_equal "one", e
end
```



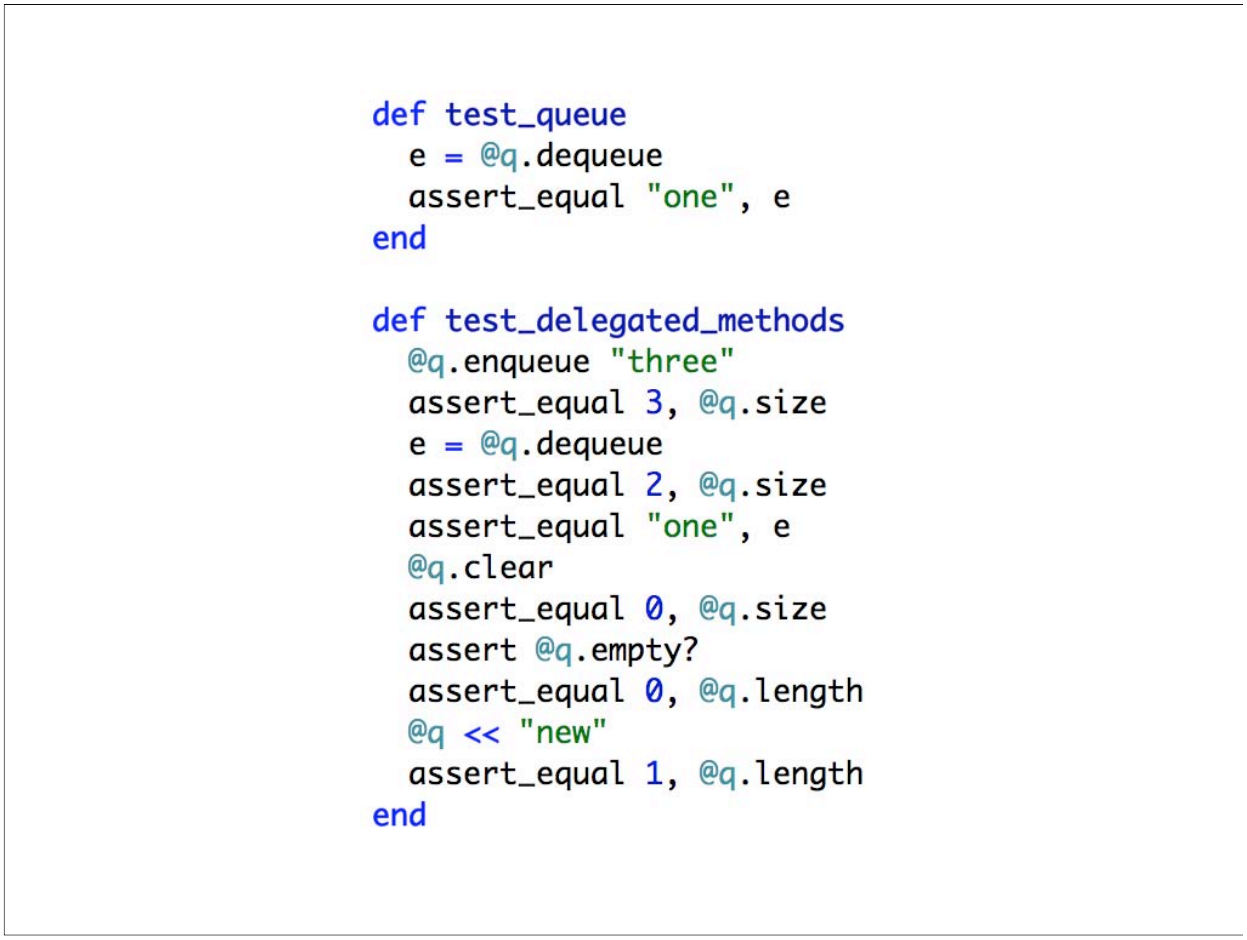
```
def test_non_delegated_methods
  @q = DelegateQueue.new
  @q.enqueue "one"
  @q.enqueue "two"
  assert_equal 2, @q.size
  e = @q.dequeue
  assert_equal 1, @q.size
  assert_equal e, "one"
end
```

a delegate is just a wrapper around another class



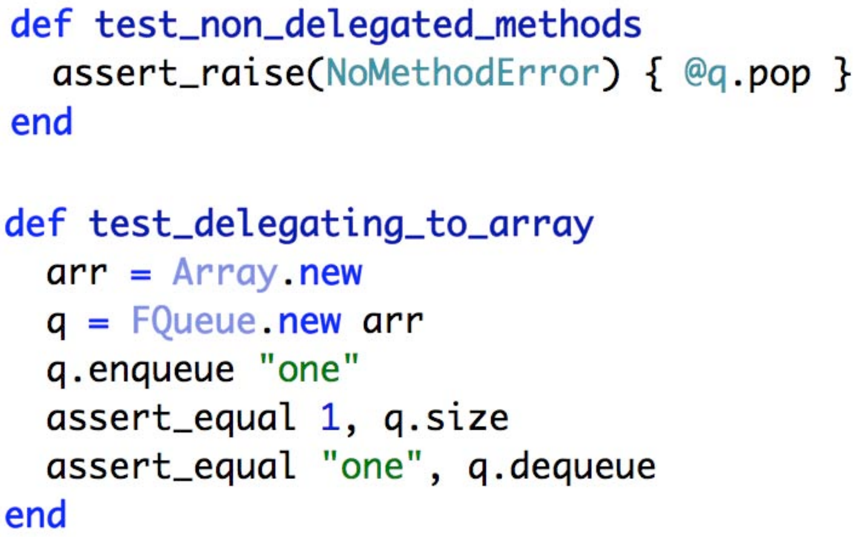
forwarding

```ruby
require 'forwardable'

class FQueue
  extend Forwardable

  def initialize(obj=[])
    @queue = obj
  end

  def_delegator :@queue, :push, :enqueue
  def_delegator :@queue, :shift, :dequeue
  def_delegators :@queue, :clear,
      :empty?, :length, :size, :<<
end
```

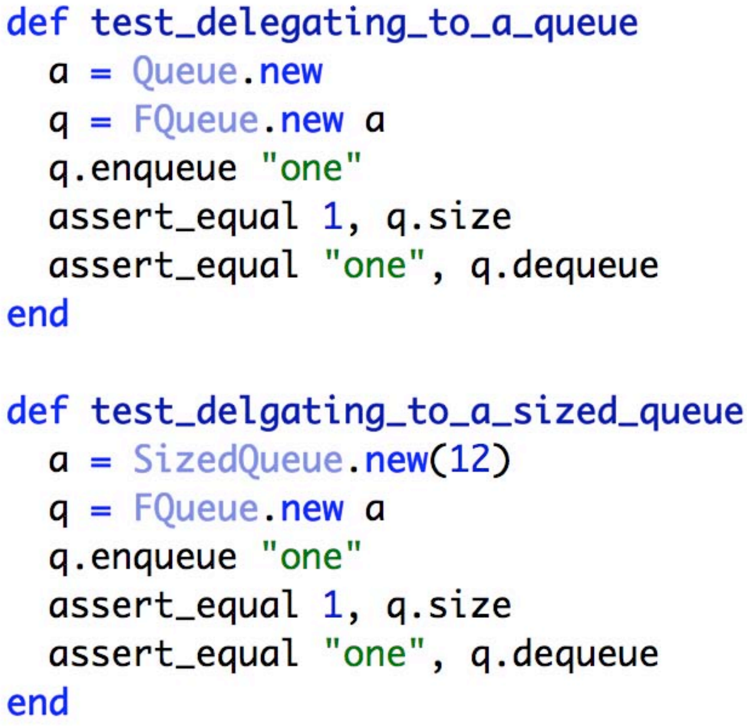


```ruby
def test_queue
  e = @q.dequeue
  assert_equal "one", e
end

def test_delegated_methods
  @q.enqueue "three"
  assert_equal 3, @q.size
  e = @q.dequeue
  assert_equal 2, @q.size
  assert_equal "one", e
  @q.clear
  assert_equal 0, @q.size
  assert @q.empty?
  assert_equal 0, @q.length
  @q << "new"
  assert_equal 1, @q.length
end
```

non-delegating methods



```
def test_non_delegated_methods
  assert_raise(NoMethodError) { @q.pop }
end

def test_delegating_to_array
  arr = Array.new
  q = FQueue.new arr
  q.enqueue "one"
  assert_equal 1, q.size
  assert_equal "one", q.dequeue
end
```



```
def test_delegating_to_a_queue
  a = Queue.new
  q = FQueue.new a
  q.enqueue "one"
  assert_equal 1, q.size
  assert_equal "one", q.dequeue
end

def test_delgating_to_a_sized_queue
  a = SizedQueue.new(12)
  q = FQueue.new a
  q.enqueue "one"
  assert_equal 1, q.size
  assert_equal "one", q.dequeue
end
```

any duck

```
class FQueue
  extend Forwardable

  def initialize(obj=[])
    @queue = obj
  end

  def_delegator :@queue, :push, :enqueue
  def_delegator :@queue, :shift, :dequeue
  def_delegators :@queue, :clear,
      :empty?, :length, :size, :<<
end
```

missings

things gone missing

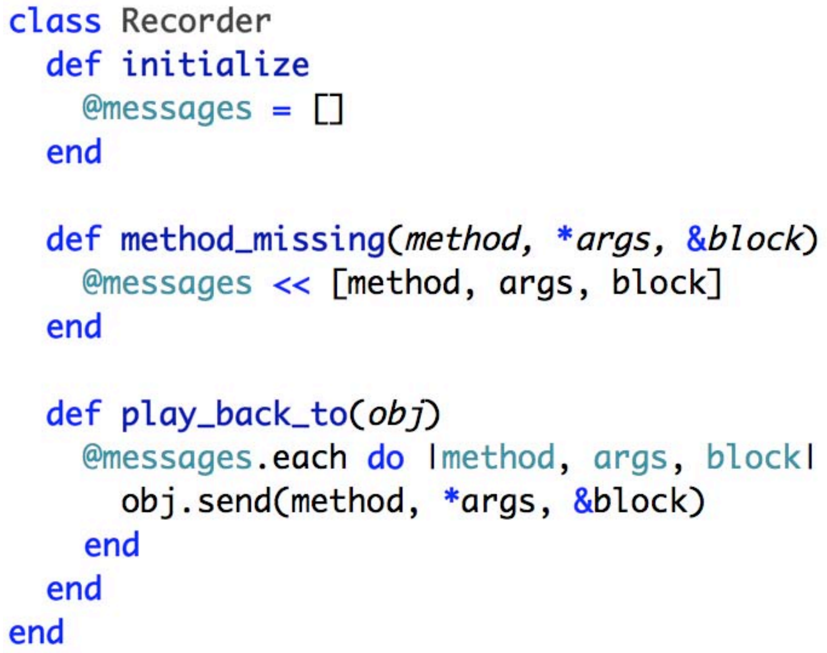
when you call a method or reference a constant that isn't around

ruby handles it with a **`missing`** method

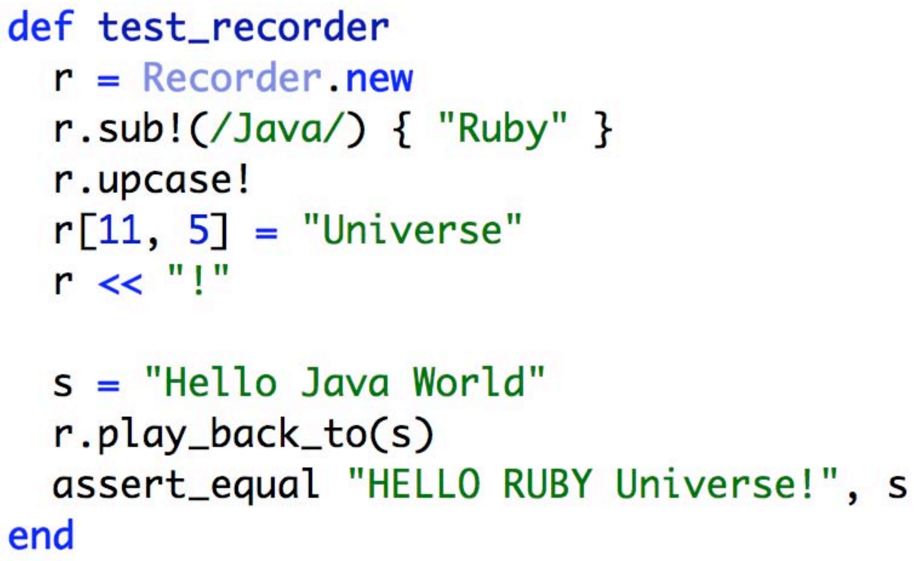
```
const_missing

method_missing
```

decorator design pattern

recorder

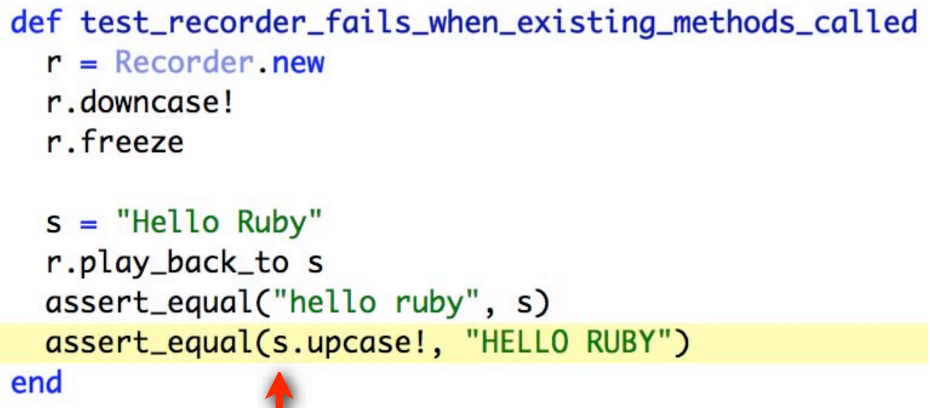


```
class Recorder
  def initialize
    @messages = []
  end

  def method_missing(method, *args, &block)
    @messages << [method, args, block]
  end

  def play_back_to(obj)
    @messages.each do |method, args, block|
      obj.send(method, *args, &block)
    end
  end
end
```



```
def test_recorder
  r = Recorder.new
  r.sub!(/Java/) { "Ruby" }
  r.upcase!
  r[11, 5] = "Universe"
  r << "!"

  s = "Hello Java World"
  r.play_back_to(s)
  assert_equal "HELLO RUBY Universe!", s
end
```

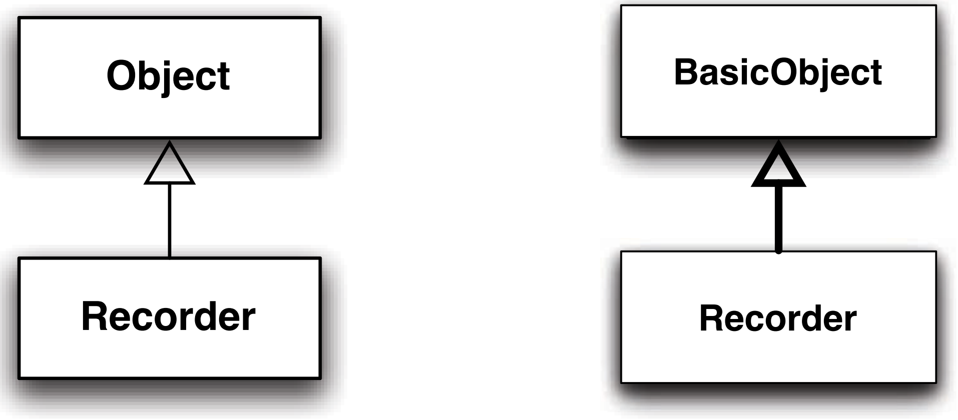
but what about this?



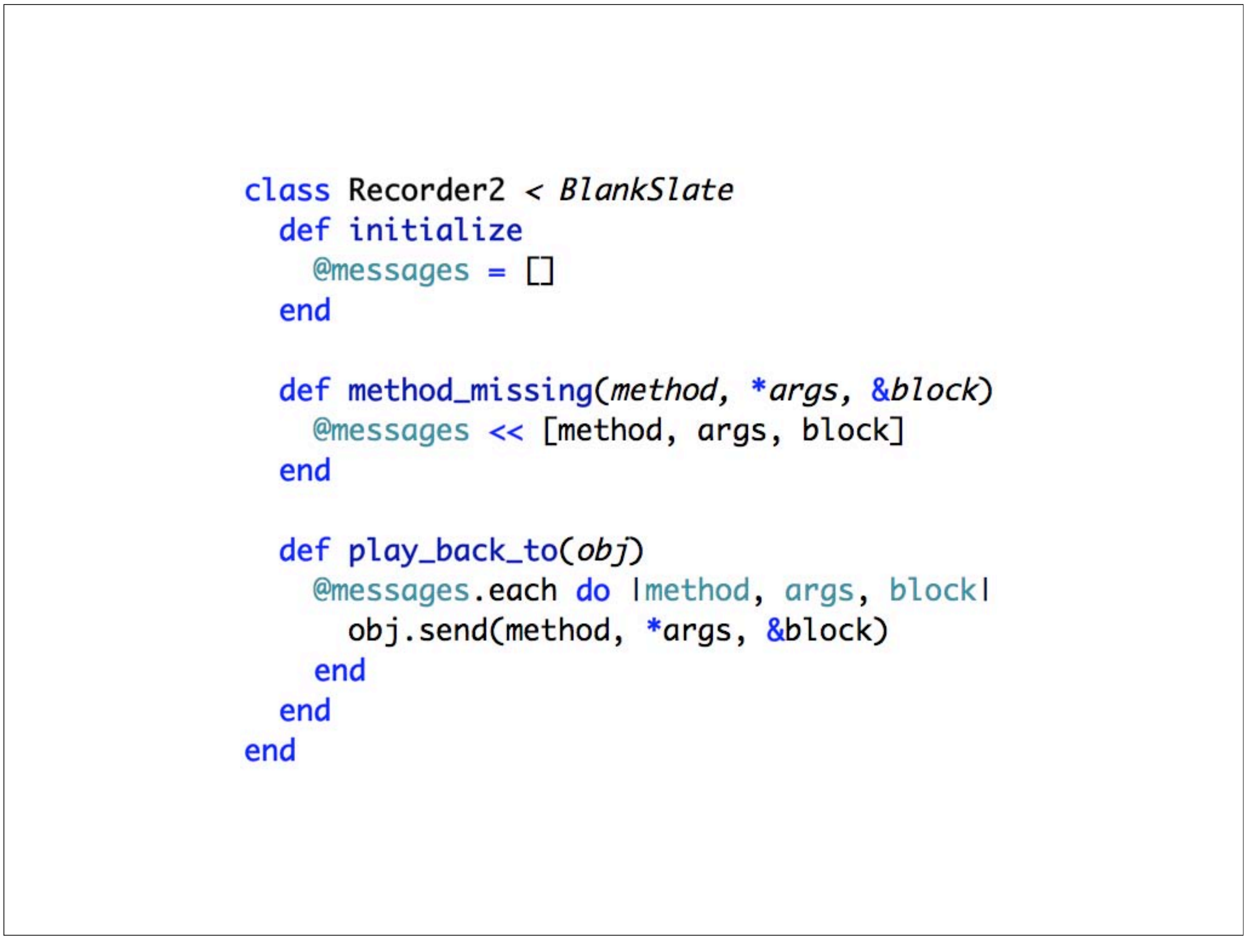
```
def test_recorder_fails_when_existing_methods_called
  r = Recorder.new
  r.downcase!
  r.freeze

  s = "Hello Ruby"
  r.play_back_to s
  assert_equal("hello ruby", s)
  assert_equal(s.upcase!, "HELLO RUBY")
end
```

should fail because *s* should be frozen

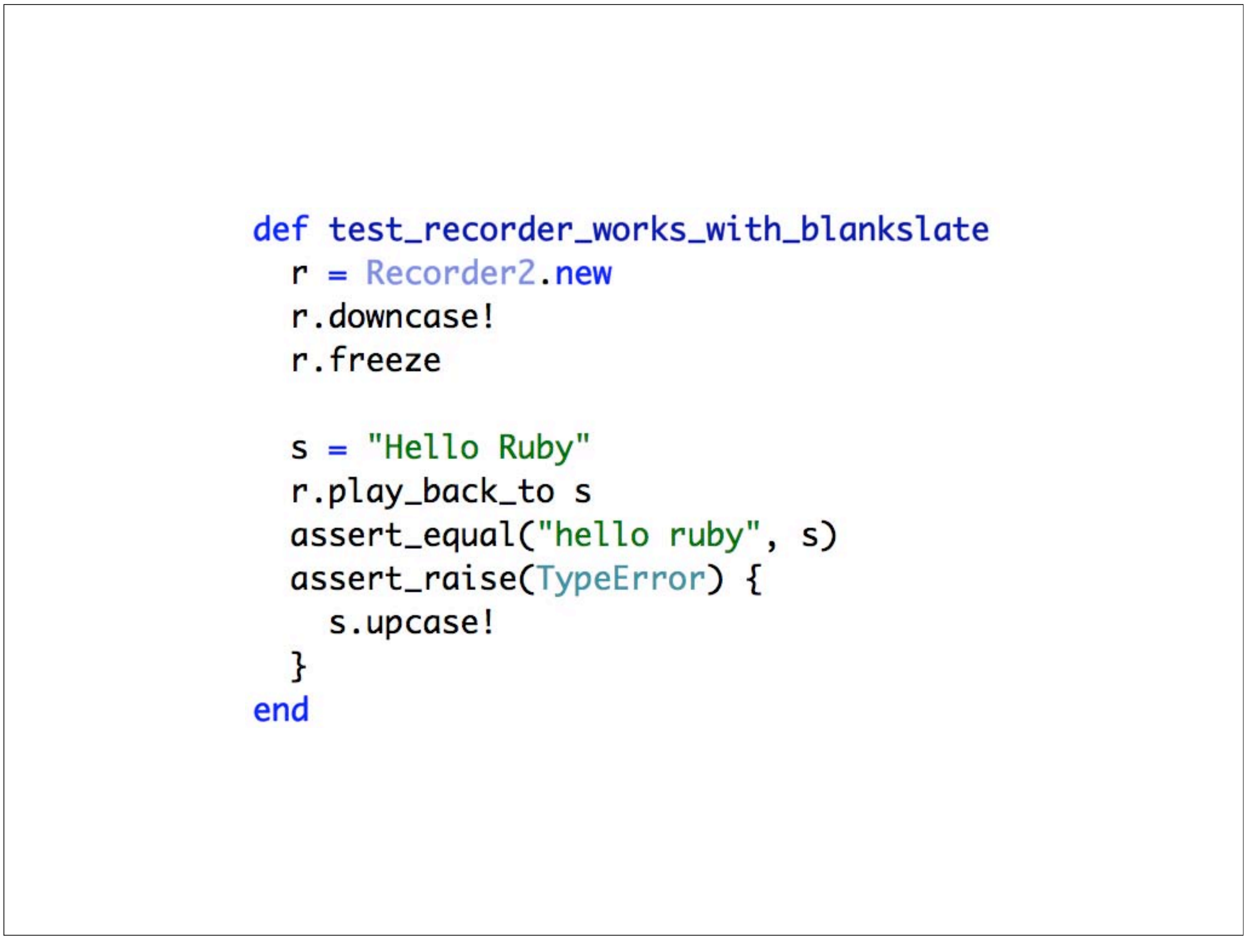
1.8 Jim Weirich's BlankSlate



1.9 BasicObject



```ruby
class Recorder2 < BlankSlate
  def initialize
    @messages = []
  end

  def method_missing(method, *args, &block)
    @messages << [method, args, block]
  end

  def play_back_to(obj)
    @messages.each do |method, args, block|
      obj.send(method, *args, &block)
    end
  end
end
```



```ruby
def test_recorder_works_with_blankslate
  r = Recorder2.new
  r.downcase!
  r.freeze

  s = "Hello Ruby"
  r.play_back_to s
  assert_equal("hello ruby", s)
  assert_raise(TypeError) {
    s.upcase!
  }
end
```

method magic

runtime access to methods

create methods with **define_method**

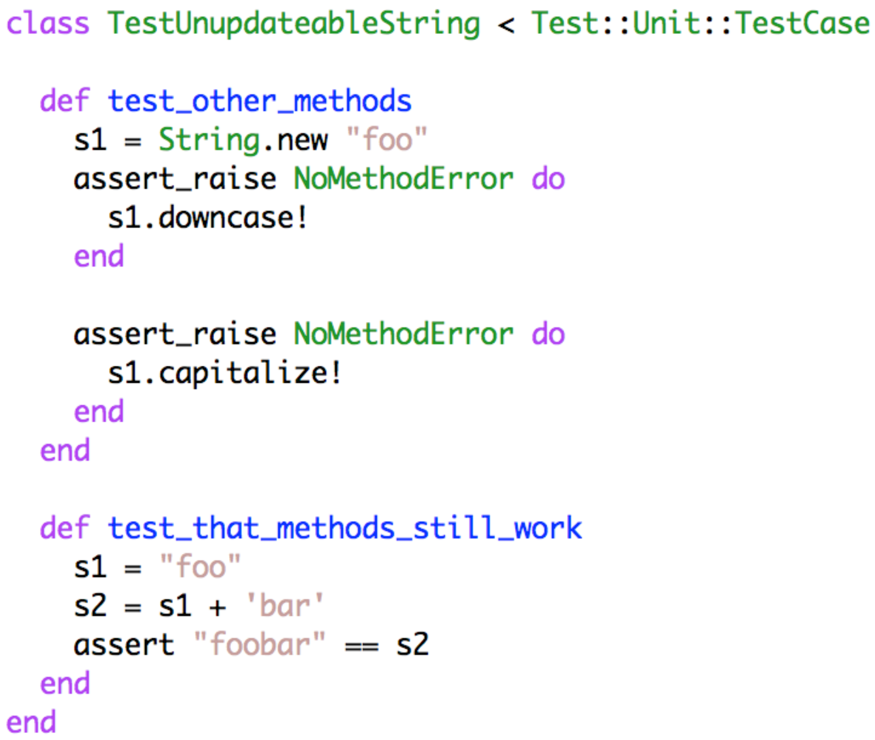
get rid of methods

remove_method - from the current class

undef_method - from the entire hierarchy!

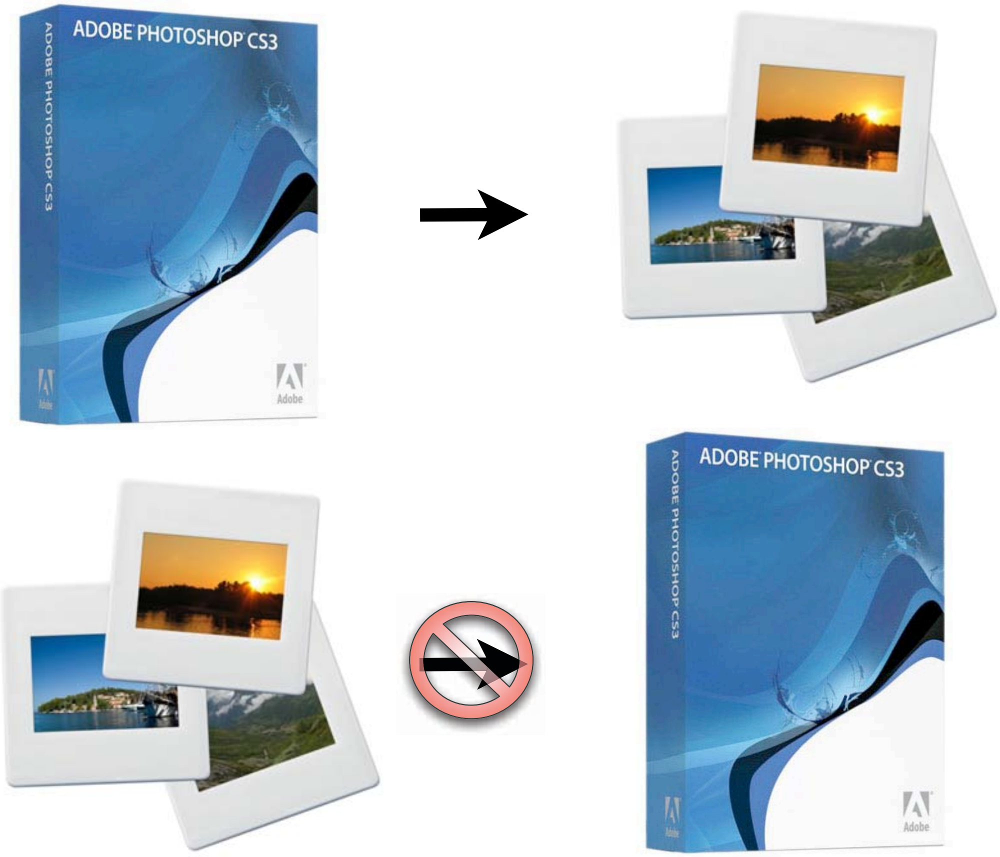
immutable string?

```ruby
class String
  instance_methods.each do |m|
    undef_method m.to_sym if m =~ /.*!$/
  end
end
```



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

  def test_other_methods
    s1 = String.new "foo"
    assert_raise NoMethodError do
      s1.downcase!
    end

    assert_raise NoMethodError do
      s1.capitalize!
    end
  end

  def test_that_methods_still_work
    s1 = "foo"
    s2 = s1 + 'bar'
    assert "foobar" == s2
  end
end
```

hooks part 1


ADOBE PHOTOSHOP CS3
ADOBE PHOTOSHOP CS3
Adobe
ADOBE PHOTOSHOP CS3
ADOBE PHOTOSHOP CS3
Adobe

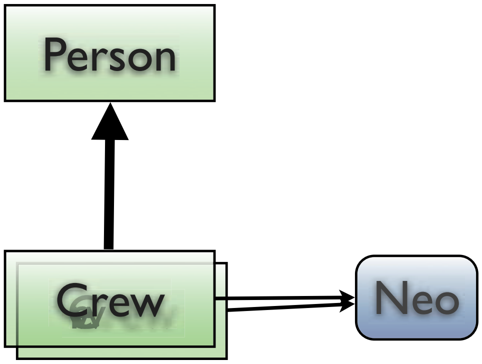
adding `final`

```
module Final
  def self.included(c)
    c.instance_eval do
      def inherited(sub)
        raise Exception,
            "Attempt to create subclass #{sub} "
            "of Final class #{self}"
      end
    end
  end
end
```

```
class P; include Final; end

class C < P; end
```

eigenclass

eigenclass



the ability to add methods
to object instances

adding methods via proxies

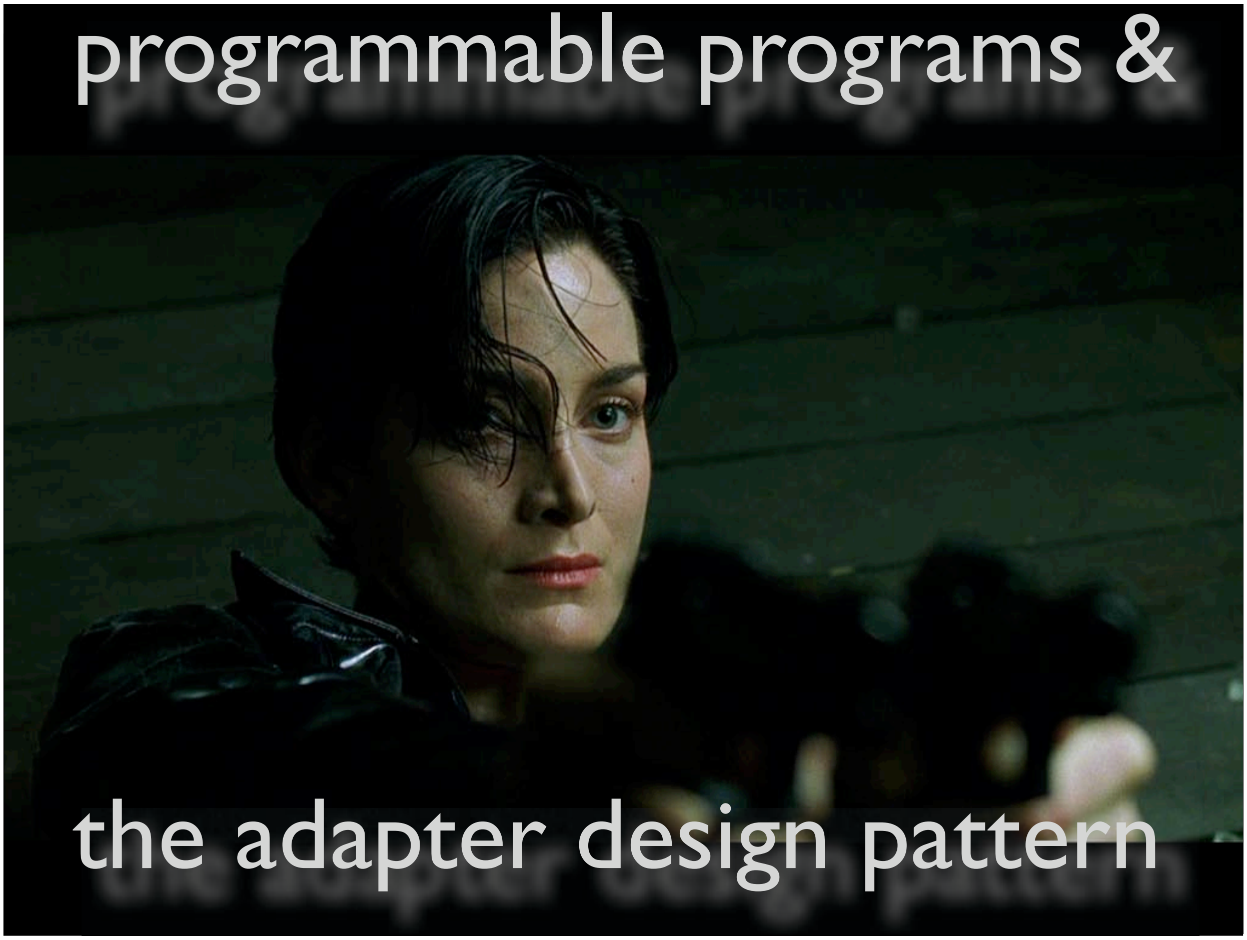
```
require "java"

include_class "java.util.ArrayList"

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

```
class TestArrayListProxy < Test::Unit::TestCase
  def setup
    @list = ArrayList.new
    @list << 'Red' << 'Green' << 'Blue'
    def @list.last
      size == 0 ? nil : get(size - 1)
    end
  end

  def test_first
    assert_equal "Red", @list.first
  end

  def test_last
    assert_equal "Blue", @list.last
  end
```

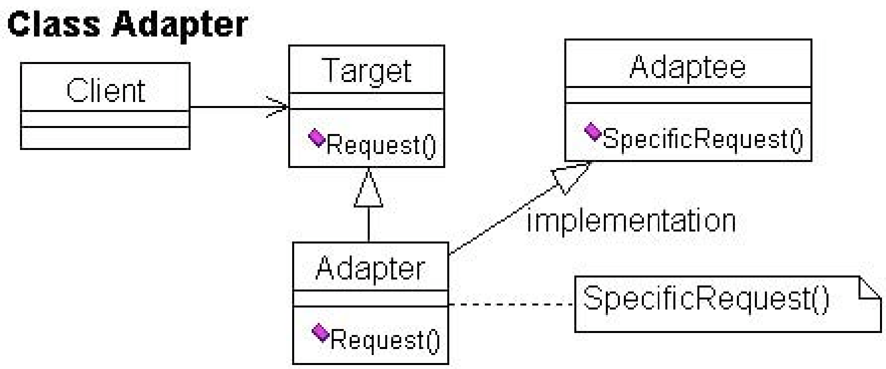
metaclass/ eigenclass

```
class Object
   def eigenclass
     class << self
       self
     end
   end
end
```

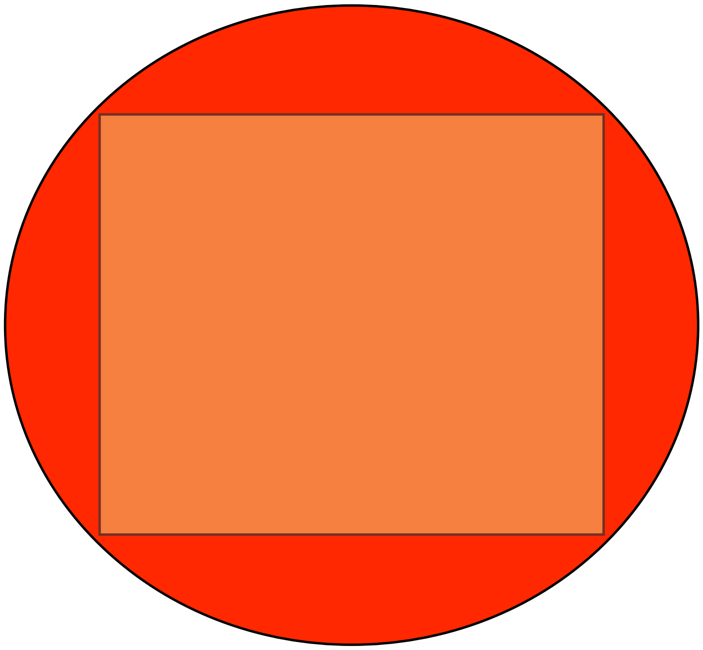
programmable programs &



the adapter design pattern



Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.



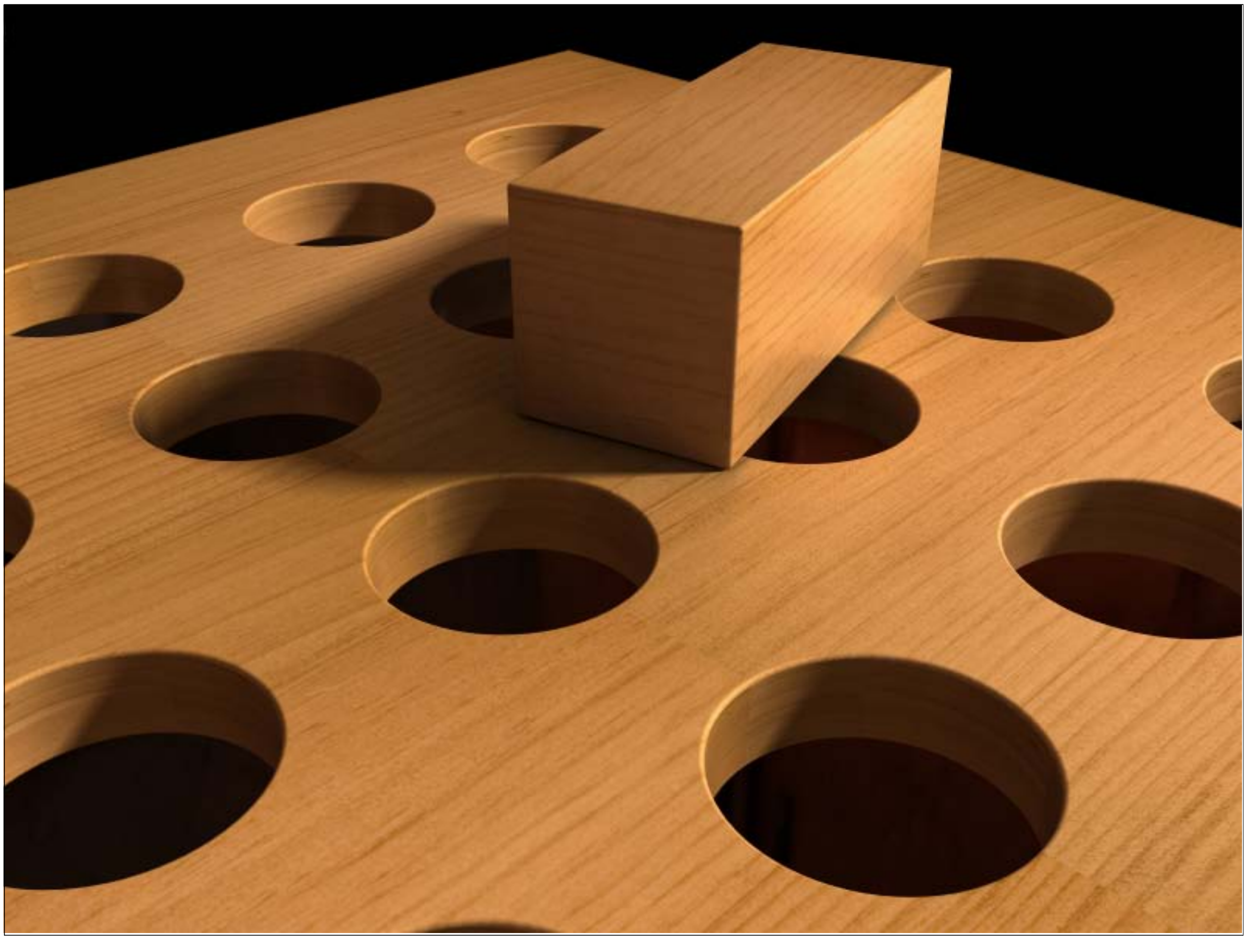
step 1:“normal” adaptor

```
class SquarePeg
    attr_reader :width

    def initialize(width)
        @width = width
    end
end

class RoundPeg
    attr_reader :radius

    def initialize(radius)
        @radius = radius
    end
end
```

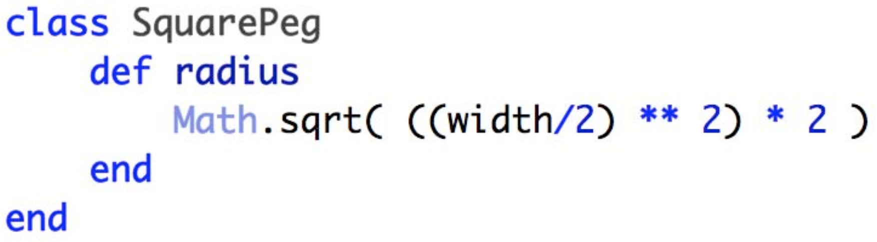
```
class RoundHole
    attr_reader :radius

    def initialize(r)
        @radius = r
    end

    def peg_fits?( peg )
        peg.radius <= radius
    end
end
```

```
class SquarePegAdaptor
    def initialize(square_peg)
        @peg = square_peg
    end

    def radius
        Math.sqrt(((@peg.width/2) ** 2)*2)
    end
end
```

```
def test_pegs
  hole = RoundHole.new(4.0)
  4.upto(7) do |i|
      peg = SquarePegAdaptor.new(SquarePeg.new(i))
      if (i < 6)
        assert hole.peg_fits?(peg)
      else
        assert ! hole.peg_fits?(peg)
      end
  end
end
```



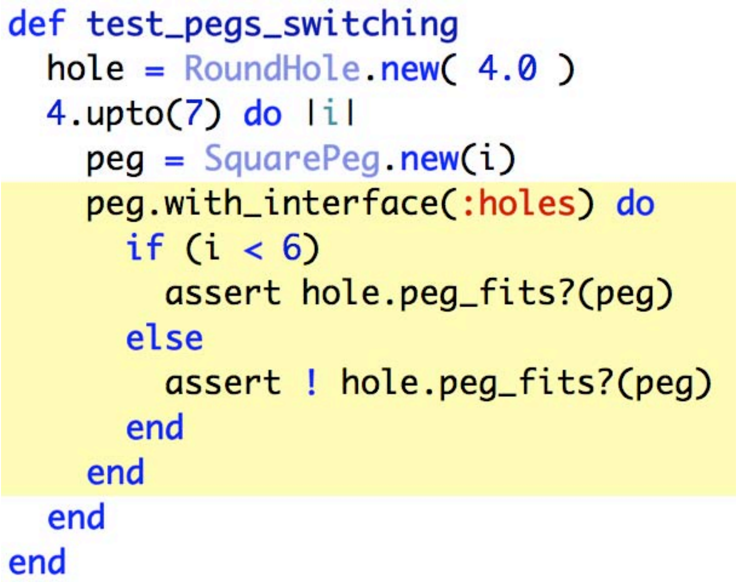
why bother with extra adaptor class?



```
class SquarePeg
    def radius
        Math.sqrt( ((width/2) ** 2) * 2 )
    end
end
```

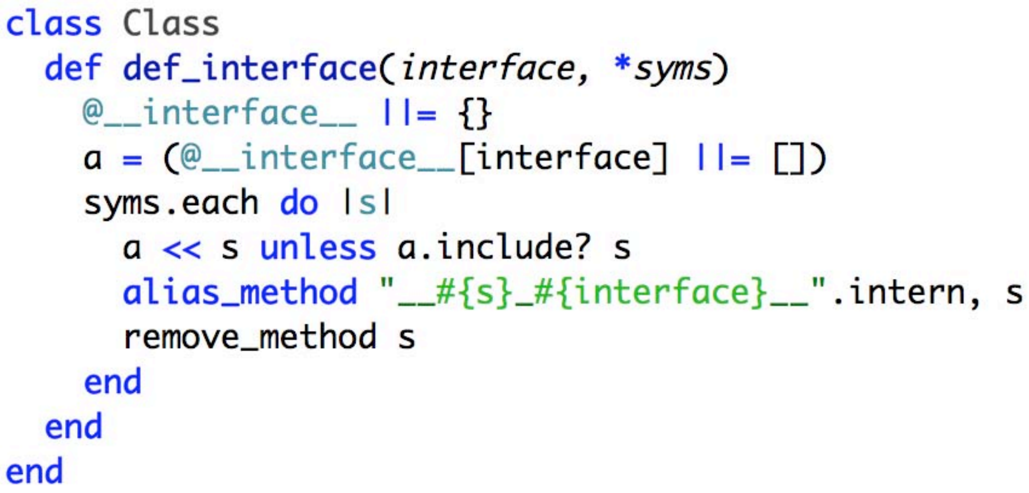
what if open class added adaptor methods clash with existing methods?

```
class SquarePeg
  include InterfaceSwitching

  def radius
    @width
  end

  def_interface :square, :radius

  def radius
    Math.sqrt(((@width/2) ** 2) * 2)
  end

  def_interface :holes, :radius

  def initialize(width)
    set_interface :square
    @width = width
  end
end
```

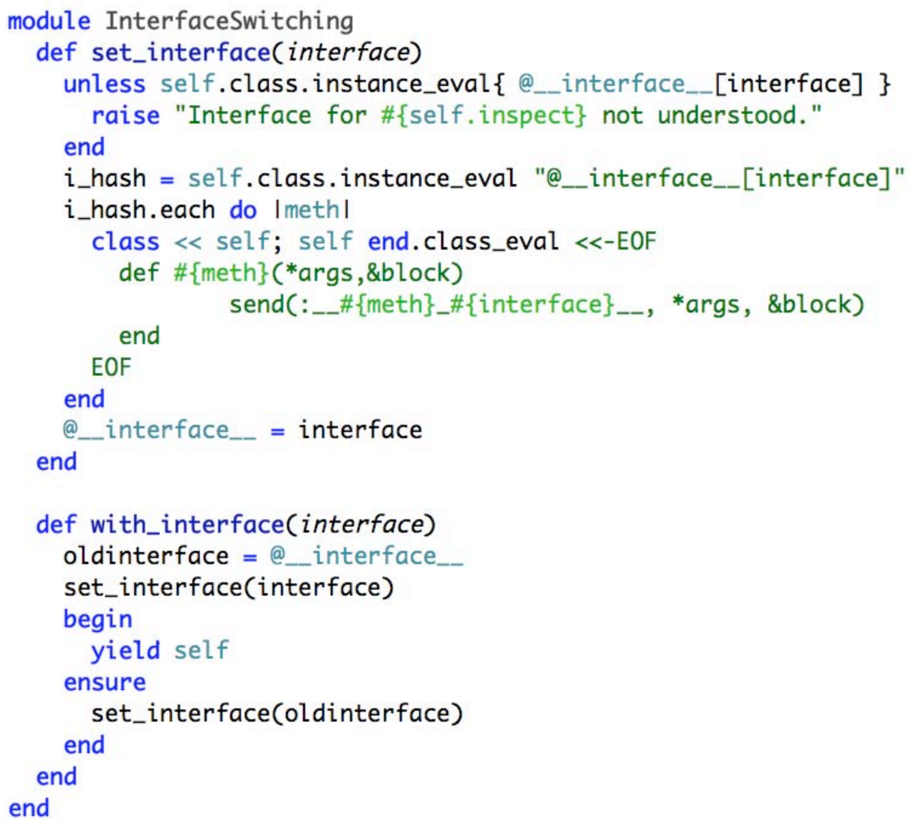


```
def test_pegs_switching
  hole = RoundHole.new( 4.0 )
  4.upto(7) do |i|
    peg = SquarePeg.new(i)
    peg.with_interface(:holes) do
      if (i < 6)
        assert hole.peg_fits?(peg)
      else
        assert ! hole.peg_fits?(peg)
      end
    end
  end
end
```

interface helper

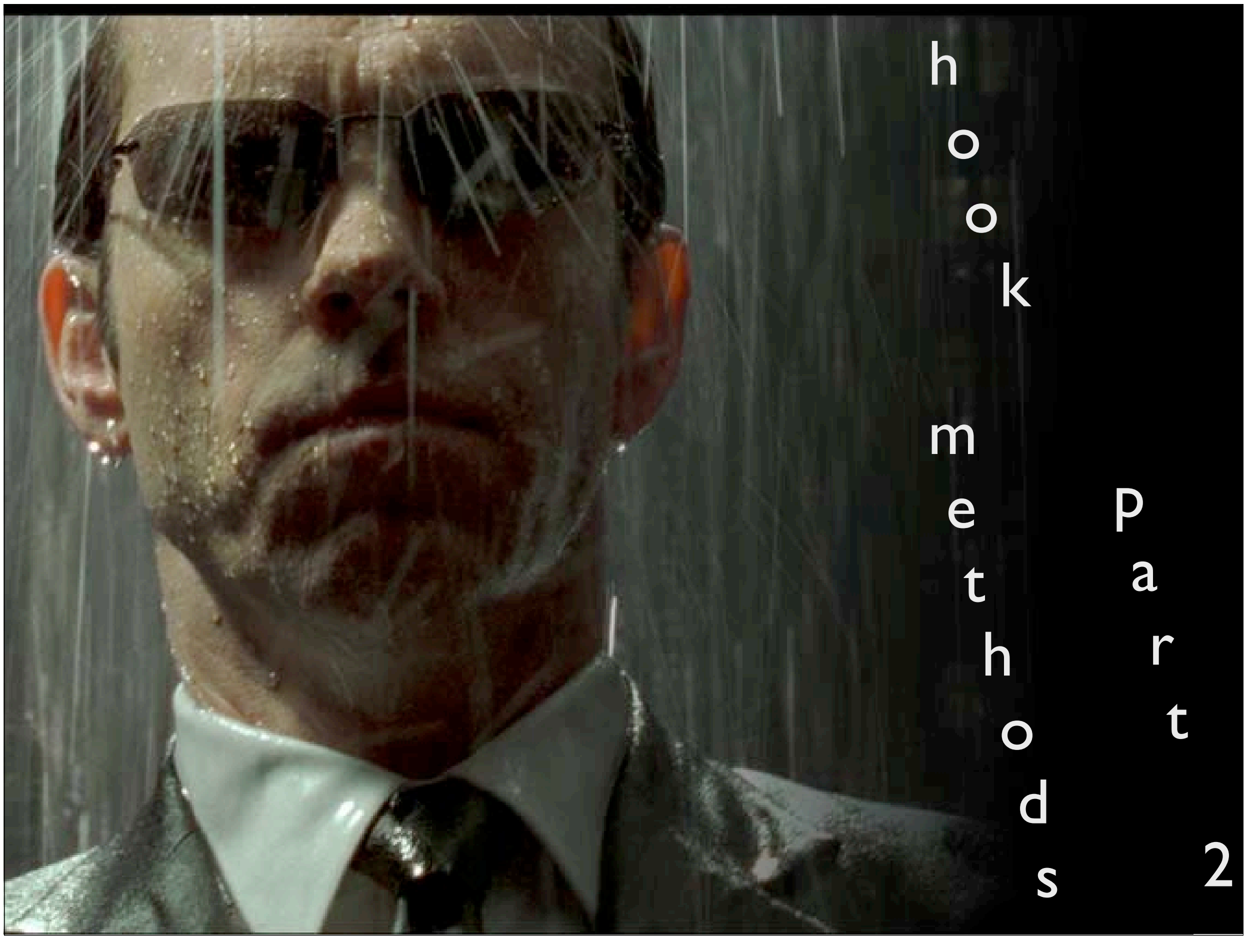


```
class Class
  def def_interface(interface, *syms)
    @__interface__ ||= {}
    a = (@__interface__[interface] ||= [])
    syms.each do |s|
      a << s unless a.include? s
      alias_method "__#{s}_#{interface}__".intern, s
      remove_method s
    end
  end
end
```



```
module InterfaceSwitching
  def set_interface(interface)
    unless self.class.instance_eval{ @__interface__[interface] }
      raise "Interface for #{self.inspect} not understood."
    end
    i_hash = self.class.instance_eval "@__interface__[interface]"
    i_hash.each do |meth|
      class << self; self end.class_eval <<-EOF
        def #{meth}(*args,&block)
                send(:__#{meth}_#{interface}__, *args, &block)
        end
      EOF
    end
    @__interface__ = interface
  end

  def with_interface(interface)
    oldinterface = @__interface__
    set_interface(interface)
    begin
      yield self
    ensure
      set_interface(oldinterface)
    end
  end
end
```

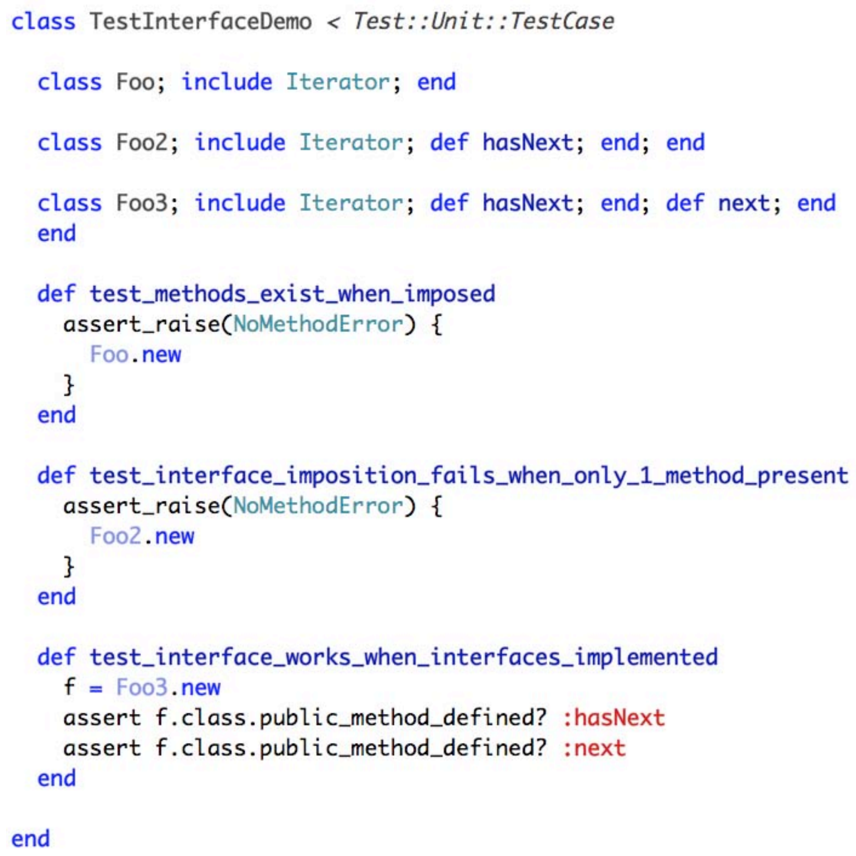
compilation is premature optimization



hook methods part 2

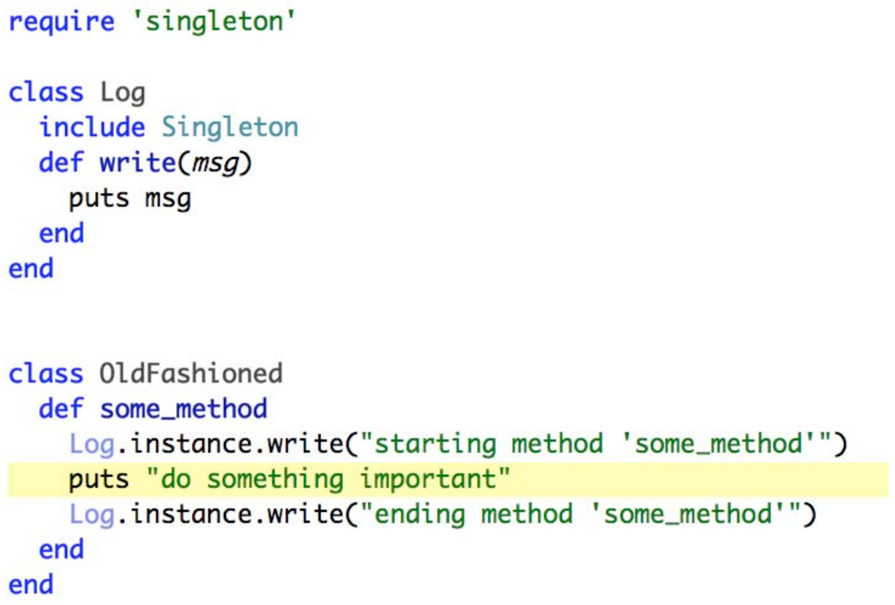
interfaces in ruby?

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



```
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
```

logging



```
require 'singleton'

class Log
  include Singleton
  def write(msg)
    puts msg
  end
end


class OldFashioned
  def some_method
    Log.instance.write("starting method 'some_method'")
    puts "do something important"
    Log.instance.write("ending method 'some_method'")
  end
end
```

```ruby
module Aop
  def Aop.included(into)
    into.instance_methods(false).each { |m| Aop.hook_method(into, m) }

    def into.method_added(meth)
      unless @adding
        @adding = true
        Aop.hook_method(self, meth)
        @adding = false
      end
    end
  end

  def Aop.hook_method(klass, meth)
    klass.class_eval do
      if meth.to_s =~ /^persist_.*/
        alias_method "old_#{meth}", "#{meth}"
        define_method(meth) do |*args|
          Log.instance.write("calling method #{meth}")
          self.send("old_#{meth}",*args)
          Log.instance.write("call finished for #{meth}")
        end
      end
    end
  end
end
```

is monkey patching evil?

aspect nomenclature

join point

points of program execution where new behavior might be inserted.

pointcut

sets of *join points* with a similar “theme”

advice

code invoked before, after, or around a *join point*

aspect oriented ruby

interception

interjection of advice, at least around methods

introduction

enhancing with new (orthogonal!) state & behavior

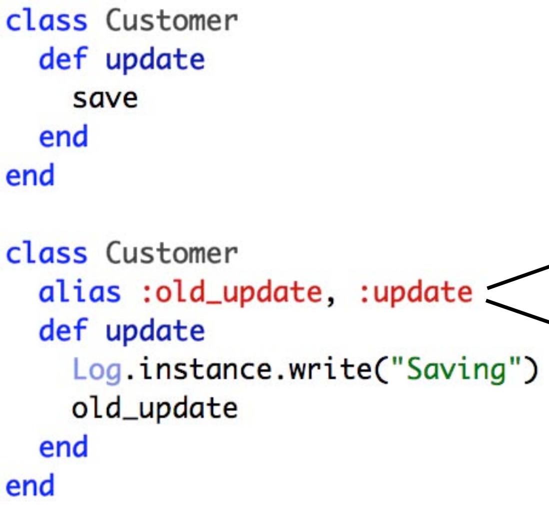
inspection

access to meta-information that may be exploited by pointcuts or advice

modularization

encapsulate as aspects

aop: interception



```
class Customer
  def update
    save
  end
end

class Customer
  alias :old_update, :update
  def update
    Log.instance.write("Saving")
    old_update
  end
end
```

alias name clashes

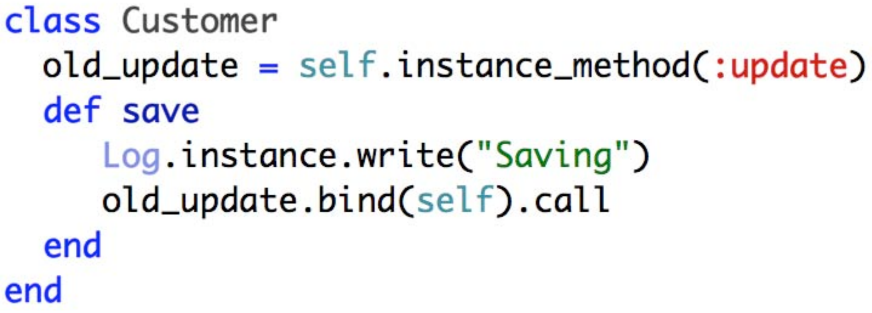
new method available

better interception

capture the target method as an unbound method

bind it to the current object

call it explicitly



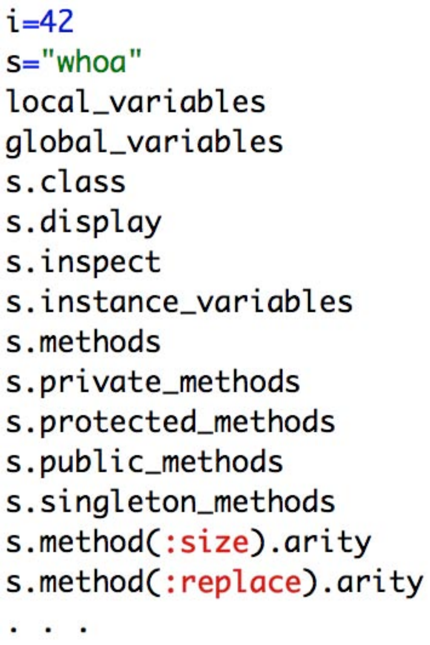
```
class Customer
  old_update = self.instance_method(:update)
  def save
     Log.instance.write("Saving")
     old_update.bind(self).call
  end
end
```

aop: introductions

add a new method to a class

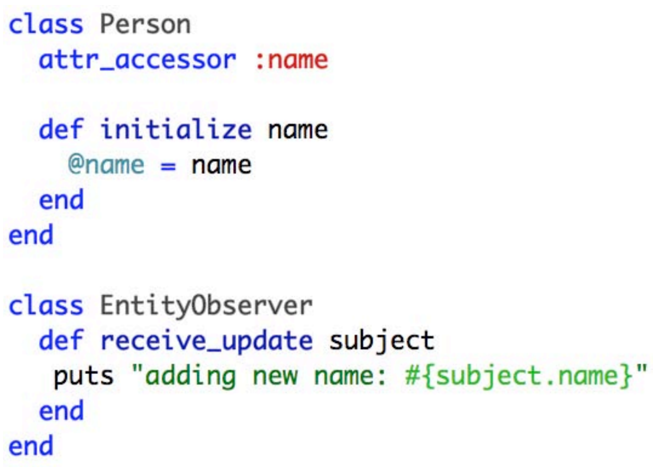
add a new method to an instance of a class (via the eigenclass)

aop: inspections

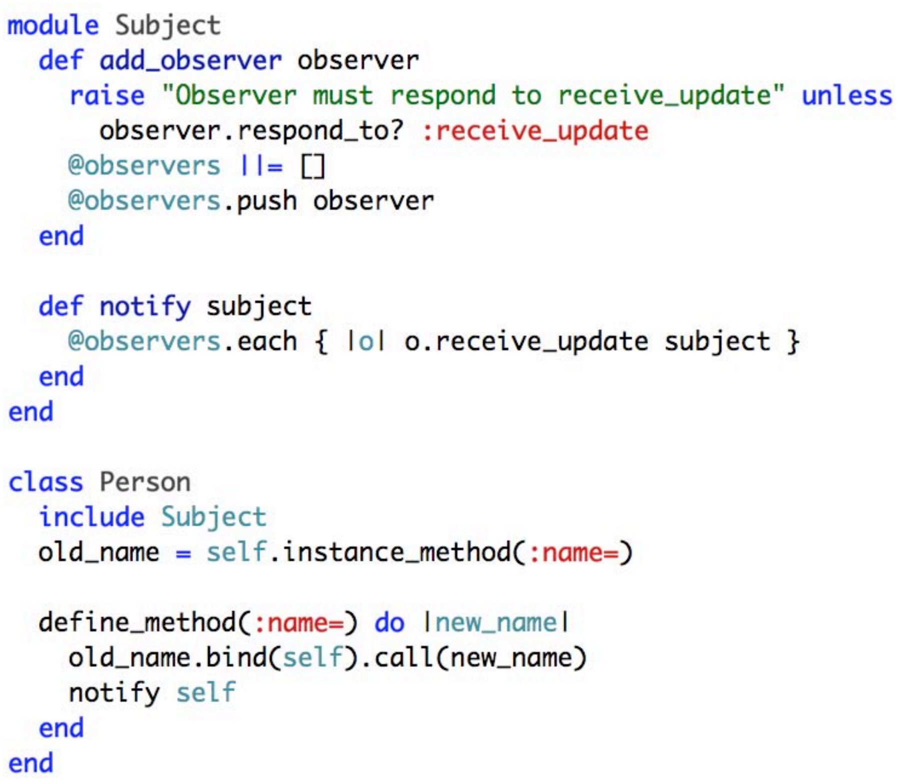


```
i=42
s="whoa"
local_variables
global_variables
s.class
s.display
s.inspect
s.instance_variables
s.methods
s.private_methods
s.protected_methods
s.public_methods
s.singleton_methods
s.method(:size).arity
s.method(:replace).arity
. . .
```

aop: modularization

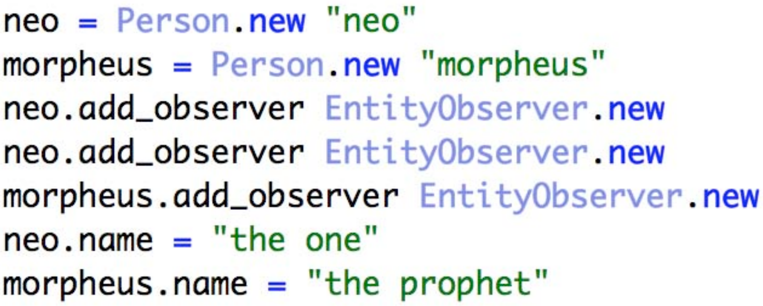


```ruby
class Person
  attr_accessor :name

  def initialize name
    @name = name
  end
end

class EntityObserver
  def receive_update subject
   puts "adding new name: #{subject.name}"
  end
end
```



```ruby
module Subject
  def add_observer observer
    raise "Observer must respond to receive_update" unless
      observer.respond_to? :receive_update
    @observers ||= []
    @observers.push observer
  end

  def notify subject
    @observers.each { |o| o.receive_update subject }
  end
end

class Person
  include Subject
  old_name = self.instance_method(:name=)

  define_method(:name=) do |new_name|
    old_name.bind(self).call(new_name)
    notify self
  end
end
```

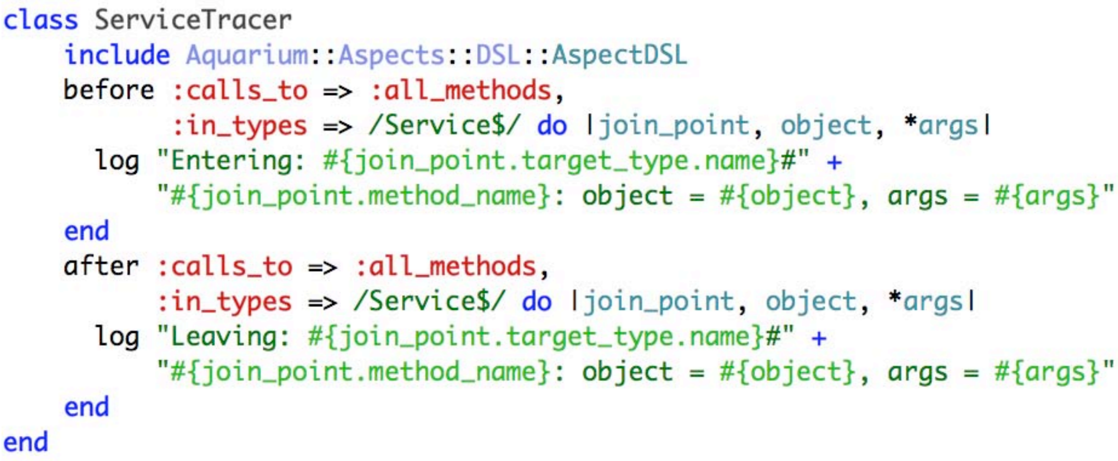
aop: modularization



```ruby
neo = Person.new "neo"
morpheus = Person.new "morpheus"
neo.add_observer EntityObserver.new
neo.add_observer EntityObserver.new
morpheus.add_observer EntityObserver.new
neo.name = "the one"
morpheus.name = "the prophet"
```

aquarium

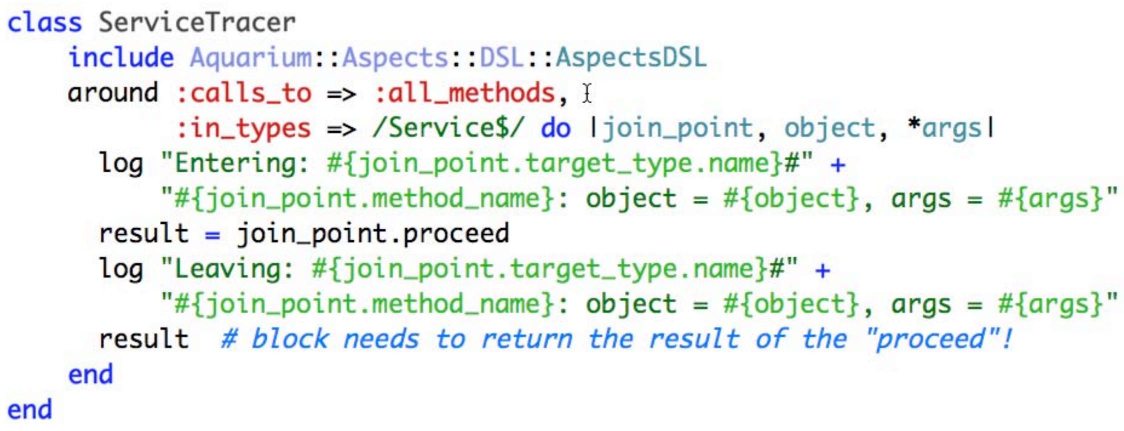
trace all invocations of the public instance methods in all classes whose names end with "Service"



```ruby
class ServiceTracer
    include Aquarium::Aspects::DSL::AspectDSL
    before :calls_to => :all_methods,
           :in_types => /Service$/ do |join_point, object, *args|
      log "Entering: #{join_point.target_type.name}#" +
          "#{join_point.method_name}: object = #{object}, args = #{args}"
    end
    after :calls_to => :all_methods,
          :in_types => /Service$/ do |join_point, object, *args|
      log "Leaving: #{join_point.target_type.name}#" +
          "#{join_point.method_name}: object = #{object}, args = #{args}"
    end
end
```

aquarium

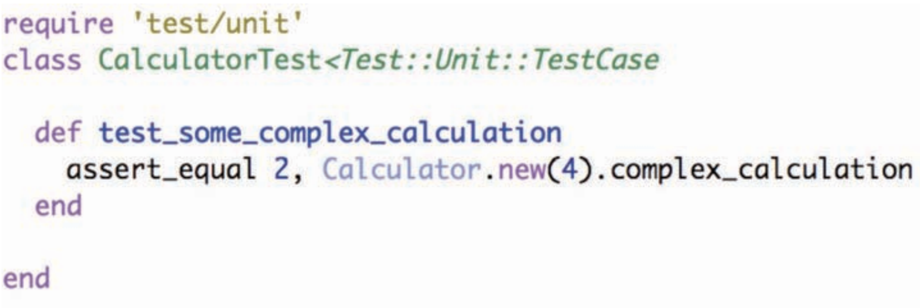
using *around* advice



```
class ServiceTracer
    include Aquarium::Aspects::DSL::AspectsDSL
    around :calls_to => :all_methods,
           :in_types => /Service$/ do |join_point, object, *args|
      log "Entering: #{join_point.target_type.name}#" +
          "#{join_point.method_name}: object = #{object}, args = #{args}"
      result = join_point.proceed
      log "Leaving: #{join_point.target_type.name}#" +
          "#{join_point.method_name}: object = #{object}, args = #{args}"
      result  # block needs to return the result of the "proceed"!
    end
end
```

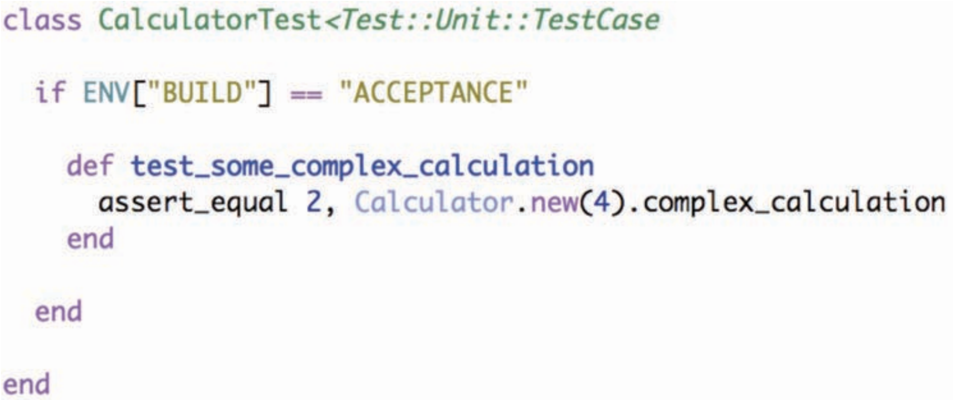
sticky attributes in ruby

limiting testing



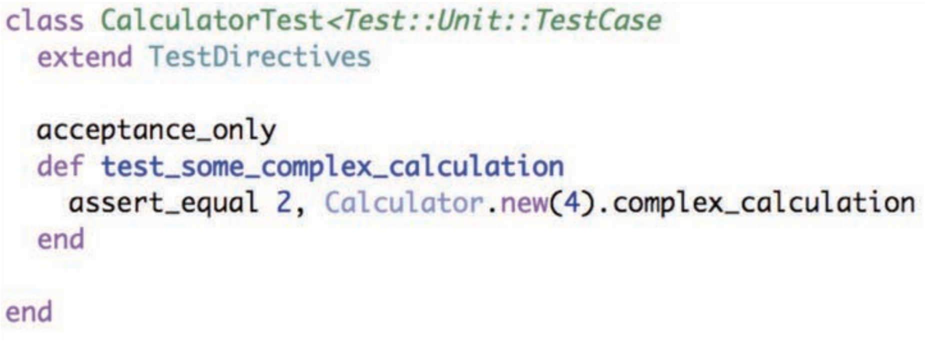
```ruby
require 'test/unit'
class CalculatorTest<Test::Unit::TestCase

  def test_some_complex_calculation
    assert_equal 2, Calculator.new(4).complex_calculation
  end

end
```

conditional method definition



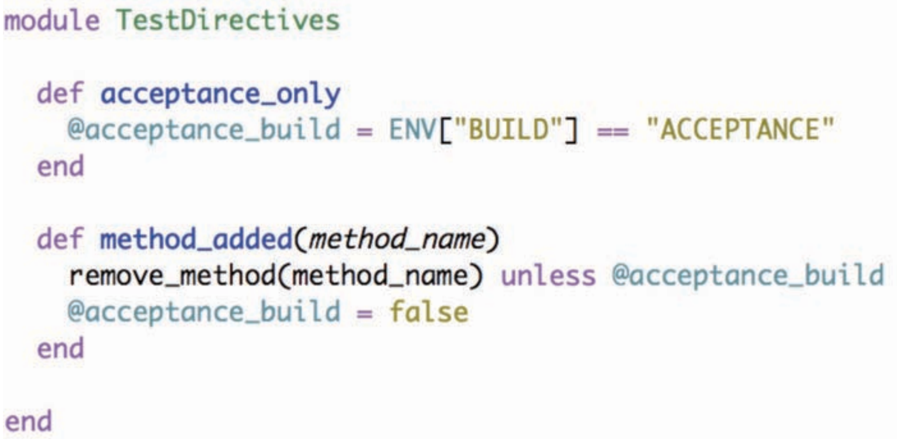
```ruby
class CalculatorTest<Test::Unit::TestCase

  if ENV["BUILD"] == "ACCEPTANCE"

    def test_some_complex_calculation
      assert_equal 2, Calculator.new(4).complex_calculation
    end

  end

end
```

attribute



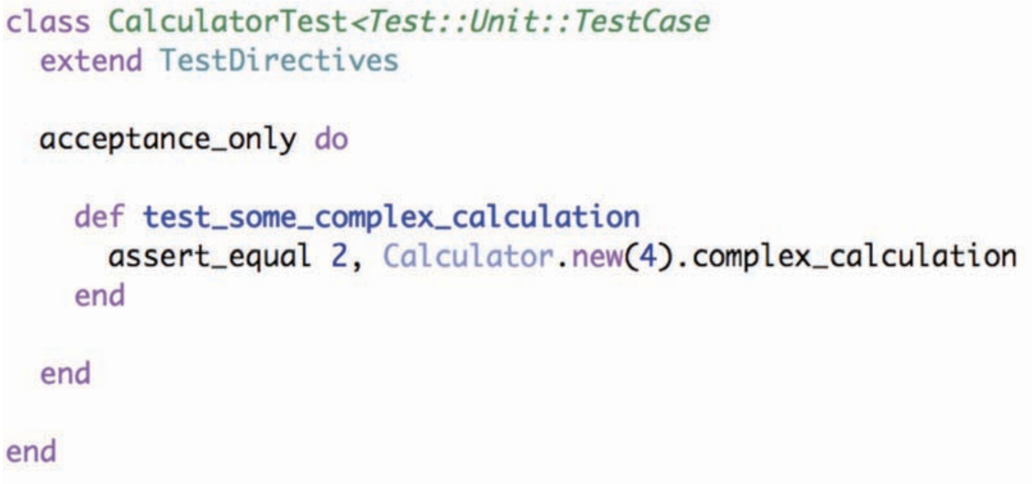
```
class CalculatorTest<Test::Unit::TestCase
  extend TestDirectives

  acceptance_only
  def test_some_complex_calculation
    assert_equal 2, Calculator.new(4).complex_calculation
  end

end
```

using hook methods



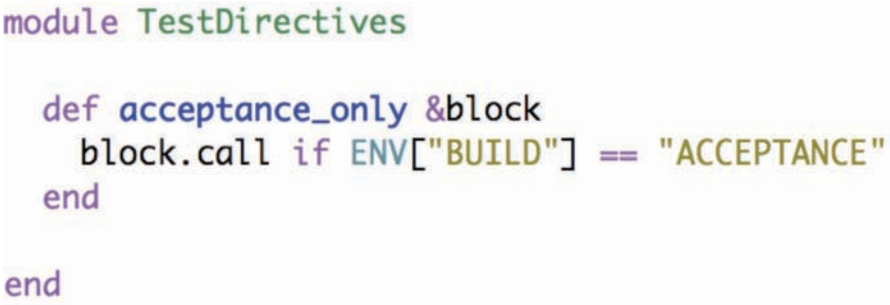
```
module TestDirectives

  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
```

delineating blocks



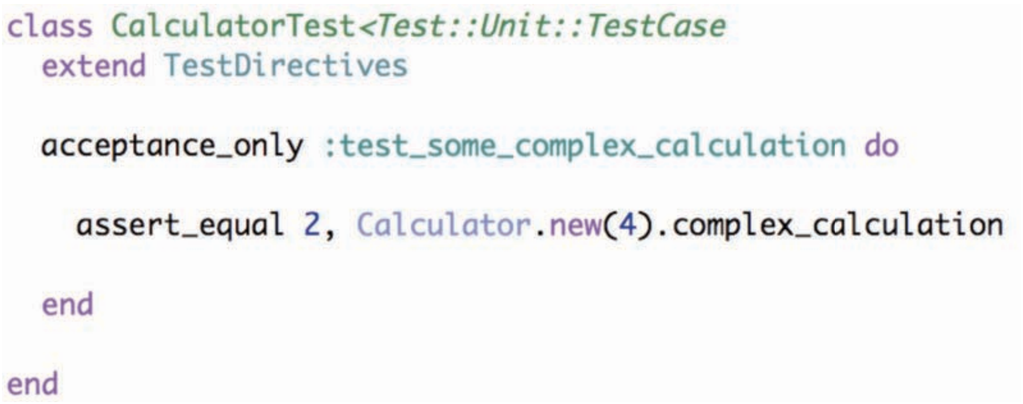
```
class CalculatorTest<Test::Unit::TestCase
  extend TestDirectives

  acceptance_only do

    def test_some_complex_calculation
      assert_equal 2, Calculator.new(4).complex_calculation
    end

  end

end
```

building block container



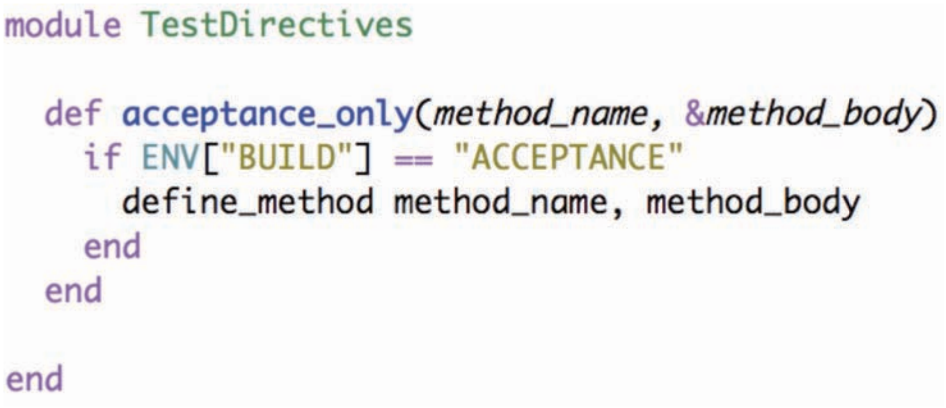
```
module TestDirectives

  def acceptance_only &block
    block.call if ENV["BUILD"] == "ACCEPTANCE"
  end

end
```

named blocks



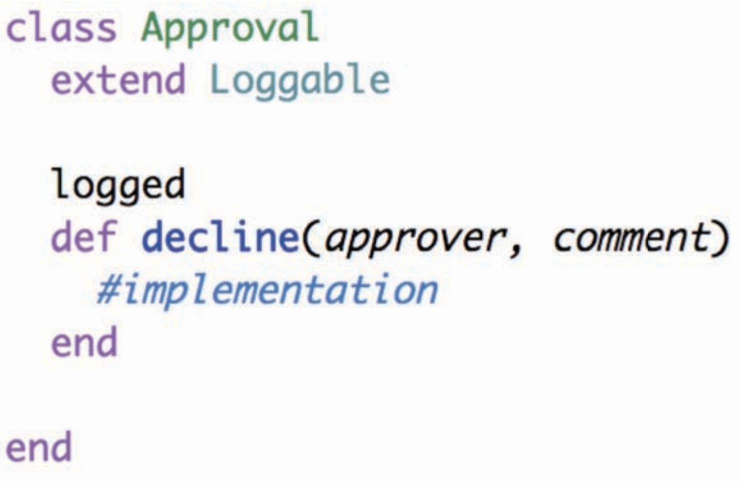
```
class CalculatorTest<Test::Unit::TestCase
  extend TestDirectives

  acceptance_only :test_some_complex_calculation do

    assert_equal 2, Calculator.new(4).complex_calculation

  end

end
```

implementing named blocks

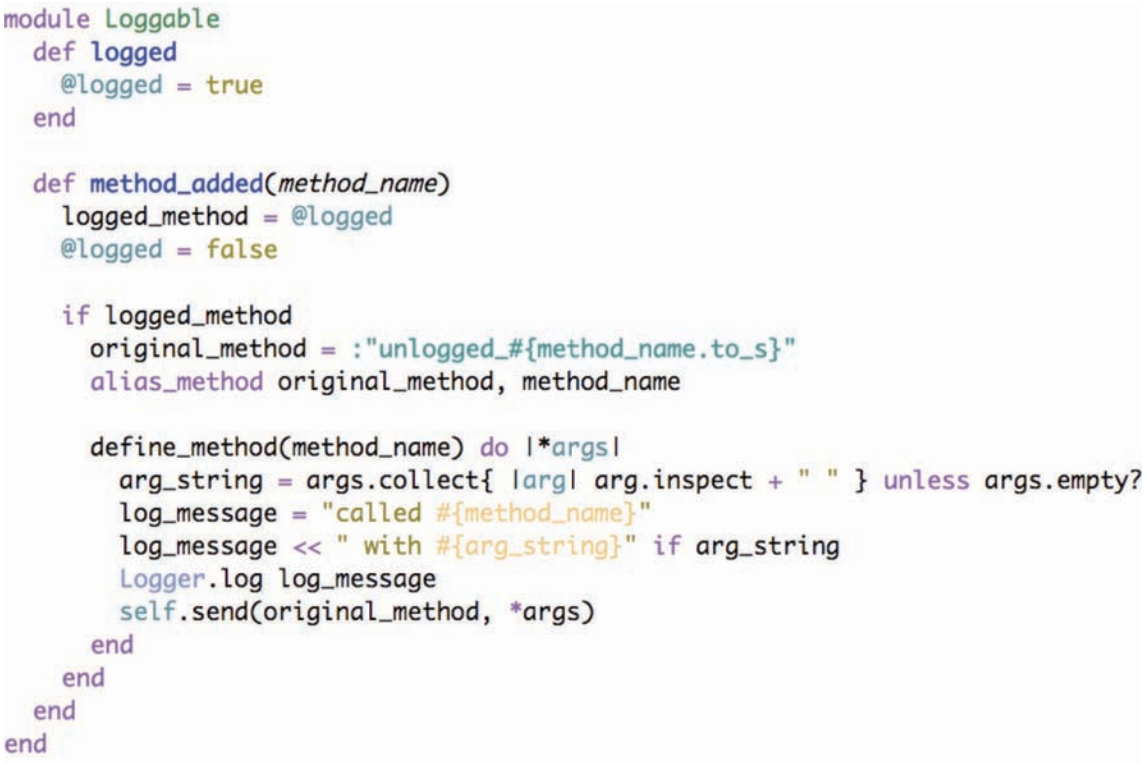


```
module TestDirectives

  def acceptance_only(method_name, &method_body)
    if ENV["BUILD"] == "ACCEPTANCE"
      define_method method_name, method_body
    end
  end

end
```

attributes for cross-cutting concerns



```
class Approval
  extend Loggable

  logged
  def decline(approver, comment)
    #implementation
  end

end
```



```
module Loggable
  def logged
    @logged = true
  end

  def method_added(method_name)
    logged_method = @logged
    @logged = false

    if logged_method
      original_method = :"unlogged_#{method_name.to_s}"
      alias_method original_method, method_name

      define_method(method_name) do |*args|
        arg_string = args.collect{ |arg| arg.inspect + " " } unless args.empty?
        log_message = "called #{method_name}"
        log_message << " with #{arg_string}" if arg_string
        Logger.log log_message
        self.send(original_method, *args)
      end
    end
  end
end
```

meta-programming makes hard problems easier... ...and the impossible merely improbable. whoa.

?'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
memeagora.blogspot.com

resources

jruby site
http://jruby.codehaus.org/

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

ola bini’s blog
http://ola-bini.blogspot.com/

benchmarks
http://shootout.alioth.debian.org/gp4sandbox/benchmark.php