ThoughtWorks®

JRuby For The Win

Ola Bini

computational metalinguist

ola.bini@gmail.com

http://olabini.com/blog

698E 2885 C1DE 74E3 2CD5 03AD 295C 7469 84AF 7F0C

Logistics and Demographics

LAST MINUTE DEMO

JRuby

Implementation of the Ruby language

Java 1.6+

1.8.7 and 1.9.3 compatible (experimental 2.0 support)

Open Source

Created 2001

Embraces testing

Current version: 1.7.4

Support from EngineYard, RedHat & ThoughtWorks

Why JRuby?

Threading

Unicode

Performance

Memory

Explicit extension API and OO internals

Libraries and legacy systems

Politics

InvokeDynamic

JRuby Differences

Most compatible alternative implementation

Native threads vs Green threads

No C extensions (well, some)

No continuations

No fork

ObjectSpace disabled by default

Simple JRuby

Java integration

Java types == Ruby types

Call methods, construct instances

Static generation of classes

camelCase or snake_case

.getFoo(), setFoo(v) becomes .foo and .foo = v

Interfaces can be implemented

Classes can be inherited from

Implicit closure conversion

Extra added features to Rubyfy Java

Ant+Rake

Clojure STM

Web

Rails

Sinatra

Trinidad

Swing

- Swing API == large and complex
  - Ruby magic simplifies most of the tricky bits
- Java is a very verbose language
  - Ruby makes Swing fun (more fun at least)
- No consistent cross-platform GUI library for Ruby
  - Swing works everywhere Java does

Swing - the direct approach

```
java_import javax.swing.JFrame
java_import javax.swing.JButton

frame = JFrame.new("Swing is easy now!")
frame.set_size 300, 300
frame.always_on_top = true

button = JButton.new("Press me!")
button.add_action_listener do |evt|
  evt.source.text = "Don't press me again!"
  evt.source.enabled = false
end

frame.add(button)
frame.show
```

Swing - Cheri (builder)

```
include Cheri::Swing

frame = swing.frame("Swing builders!") { |form|
  size 300, 300
  box_layout form, :Y_AXIS
  content_pane { background :WHITE }

  button("Event binding is nice") { |btn|
    on_click { btn.text = "You clicked me!" }
  }
}

frame.visible = true
```

Swing - Profligacy

```
class ProfligacyDemo
  java_import javax.swing.*
  include Profligacy

  def initialize
    layout = "[<translate][*input][>result]"
    @ui = Swing::LEL.new(JFrame, layout) {|cmps, ints|
      cmps.translate = JButton.new("Translate")
      cmps.input = JTextField.new
      cmps.result = JLabel.new

      translator = proc {|id, evt|
        original = @ui.input.text
        translation = MyTranslator.translate(original)
        @ui.result.text = translation
      }

      ints.translate = {:action => translator}
    }
  end
end
```

Swing - MonkeyBars (tools)

GUI editor friendly (e.g. NetBeans “Matisse”)

Simple Ruby MVC based API

Combines best of both worlds

Testing

Ruby frameworks

Cucumber

JtestR

The problem with mocking

It works fine while staying in Ruby-land

Setting expectations to be consumed by Java is problematic

The interface

```
package org.test;

public interface Quux {
    String one();
    String two();
}
```

The class

```
package org.test;

public class Foo implements Quux {
    public String one() {
        return "hello";
    }

    public String two() {
        return "goodbye";
    }
}
```

The consumer

```
package org.test;

public class Bar {
    public void doFoo(Foo foo) {
        System.out.println("Foo one: " + foo.one());
        System.out.println("Foo two: " + foo.two());
    }

    public void doQuux(Quux q) {
        System.out.println("Quux one: " + q.one());
        System.out.println("Quux two: " + q.two());
    }
}
```

Mocking an interface

```
describe "Foo and Bar with RSpec mocking" do
  it "can mock out parts of interface methods with RSpec" do
    f = org.test.Quux.new
    b = org.test.Bar.new
    f.should_receive(:one).and_return "Canned answer 1 from RSpec"
    f.should_receive(:two).and_return "Canned answer 2 from RSpec"

    b.do_quux f
  end
end
```

Mocking a class

```
describe "Foo and Bar with RSpec mocking" do
  it "can mock out parts of instance methods with RSpec" do
    f = org.test.Foo.new
    b = org.test.Bar.new
    f.should_receive(:one).and_return "Canned answer 1 from RSpec"
    f.should_receive(:two).and_return "Canned answer 2 from RSpec"

    b.do_foo f
  end
end
```

The output

```
Quux one: Canned answer 1 from RSpec
Quux two: Canned answer 2 from RSpec
.Foo one: hello
Foo two: goodbye
F

1)
Spec::Mocks::MockExpectationError in 'Foo and Bar with RSpec mocking can mock out parts of
instance methods with RSpec'
org.test.Foo@7570b819 expected :one with (any args) once, but received it 0 times
./rspec_mocking_spec.rb:18:

Finished in 0.321 seconds

2 examples, 1 failure
```

The JtestR solution

Cucumber

Behavior driven development

Plain text stories

Used to describe high level behavior

“Business-readable domain specific language”

Serves as documentation, automated tests and dev aid

Supports table-based tests

Erlang+Ruby

Google AppEngine

JRuby runs on it

JRuby-rack supports it

Google gems

Startup time

Merb, Ramaze and Sinatra easy options

Rails works

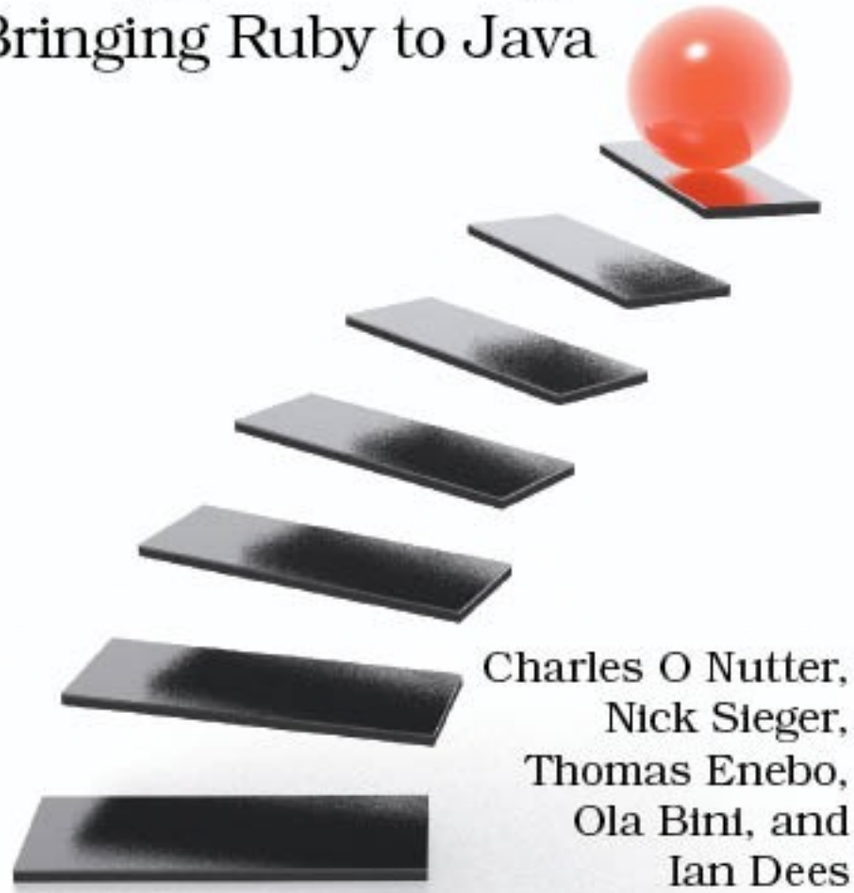
Mobile

Usage

The Pragmatic Programmers

Using JRuby

Bringing Ruby to Java



Charles O Nutter,
Nick Sieger,
Thomas Enebo,
Ola Bini, and
Ian Dees

Edited by Jacquelyn Carter

Questions?

Ola Bini

ThoughtWorks®

http://olabini.com
obini@thoughtworks.com
@olabini