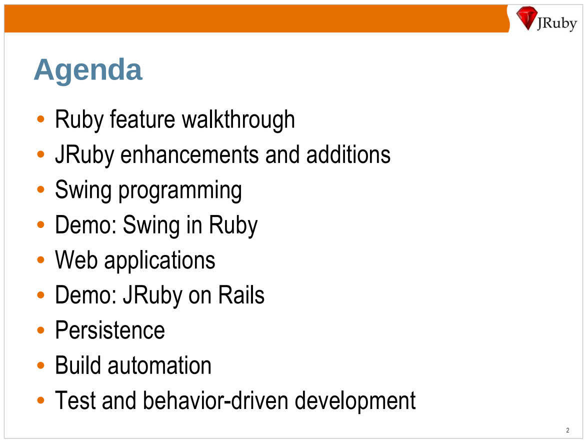
JRuby: Not Just Another JVM Language

Charles Nutter
Sun Microsystems

1



Agenda

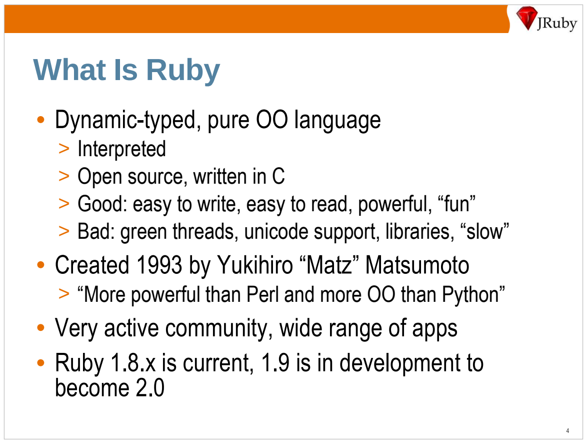
- Ruby feature walkthrough
- JRuby enhancements and additions
- Swing programming
- Demo: Swing in Ruby
- Web applications
- Demo: JRuby on Rails
- Persistence
- Build automation
- Test and behavior-driven development

2

Who am I

- Charles Oliver Nutter
- Longtime Java developer (10+ yrs)
- Engineer at Sun Microsystems for 1 yr
- Full-time JRuby developer
- Also working on JVM dynlang support
- Wide range of past experience
  - C, C++, C#, Perl, Python, Delphi, Lisp, Scheme
  - Java EE and ME, JINI, WS

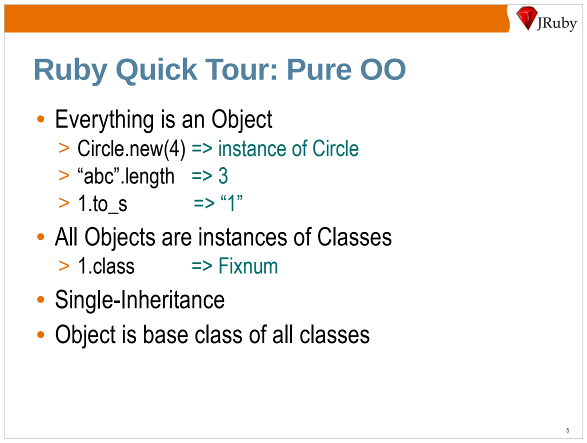
3



What Is Ruby

- Dynamic-typed, pure OO language
  - Interpreted
  - Open source, written in C
  - Good: easy to write, easy to read, powerful, “fun”
  - Bad: green threads, unicode support, libraries, “slow”
- Created 1993 by Yukihiro “Matz” Matsumoto
  - “More powerful than Perl and more OO than Python”
- Very active community, wide range of apps
- Ruby 1.8.x is current, 1.9 is in development to become 2.0

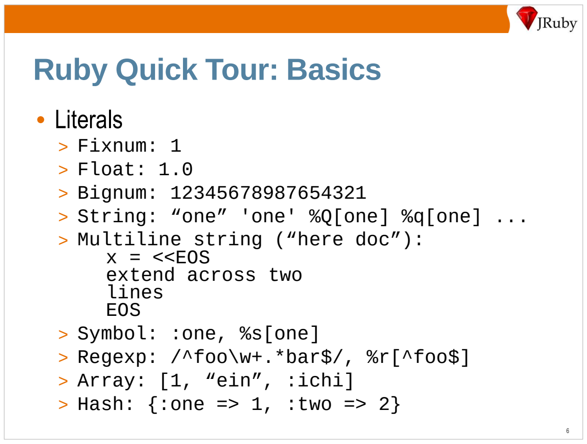
4



Ruby Quick Tour: Pure OO

- Everything is an Object
  - Circle.new(4) => instance of Circle
  - “abc”.length => 3
  - 1.to_s => “1”
- All Objects are instances of Classes
  - 1.class => Fixnum
- Single-Inheritance
- Object is base class of all classes

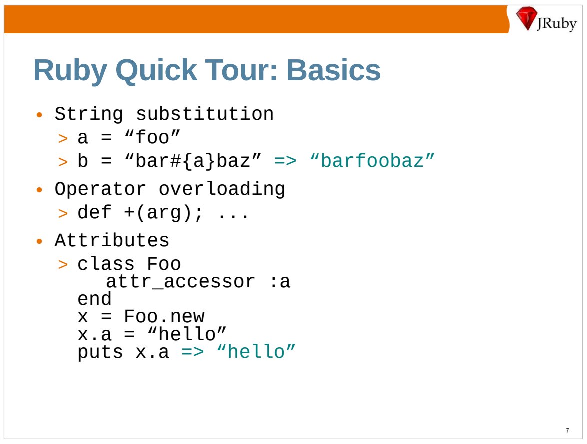
5



Ruby Quick Tour: Basics

- Literals
  - `Fixnum: 1`
  - `Float: 1.0`
  - `Bignum: 12345678987654321`
  - `String: “one” 'one' %Q[one] %q[one] ...`
  - `Multiline string (“here doc”):`
    ```
    x = <<EOS
    extend across two
    lines
    EOS
    ```
  - `Symbol: :one, %s[one]`
  - `Regexp: /^foo\w+.*bar$/, %r[^foo$]`
  - `Array: [1, “ein”, :ichi]`
  - `Hash: {:one => 1, :two => 2}`

6

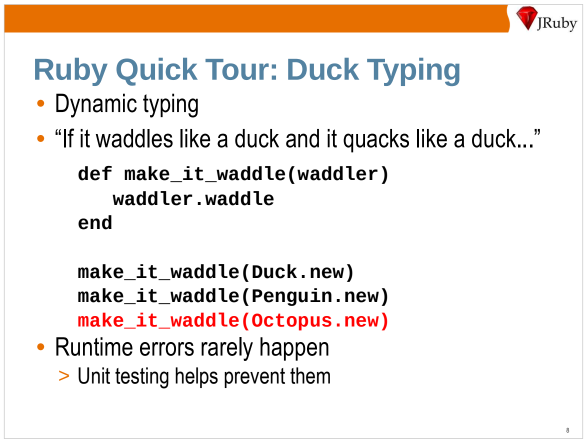


JRuby

Ruby Quick Tour: Basics

- String substitution
  - a = “foo”
  - b = “bar#{a}baz” => “barfoobaz”
- Operator overloading
  - def +(arg); ...
- Attributes
  -
```
class Foo
  attr_accessor :a
end
x = Foo.new
x.a = “hello”
puts x.a => “hello”
```

7



JRuby

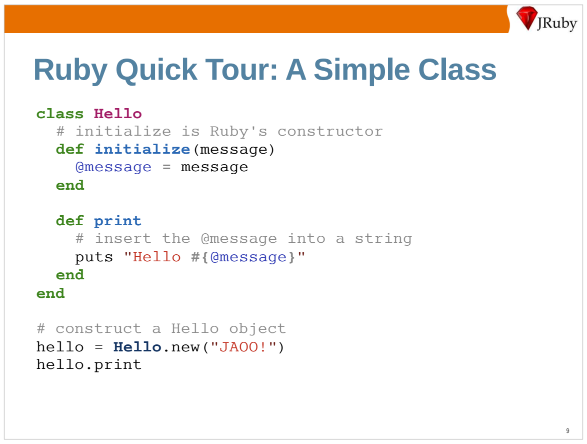
Ruby Quick Tour: Duck Typing

- Dynamic typing
- “If it waddles like a duck and it quacks like a duck...”

```
def make_it_waddle(waddler)
   waddler.waddle
end

make_it_waddle(Duck.new)
make_it_waddle(Penguin.new)
make_it_waddle(Octopus.new)
```

- Runtime errors rarely happen
  - Unit testing helps prevent them

8

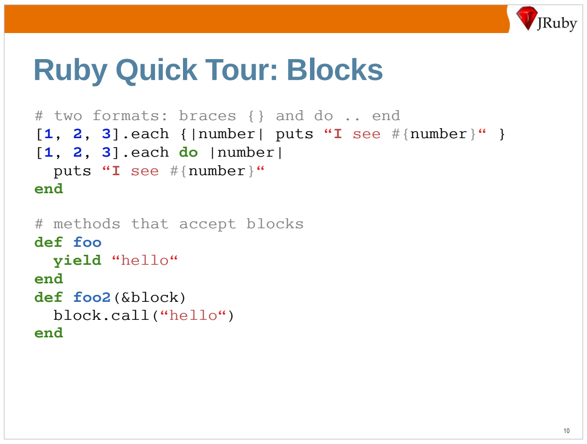


JRuby

Ruby Quick Tour: A Simple Class

```
class Hello
  # initialize is Ruby's constructor
  def initialize(message)
    @message = message
  end

  def print
    # insert the @message into a string
    puts "Hello #{@message}"
  end
end

# construct a Hello object
hello = Hello.new("JAOO!")
hello.print
```

9

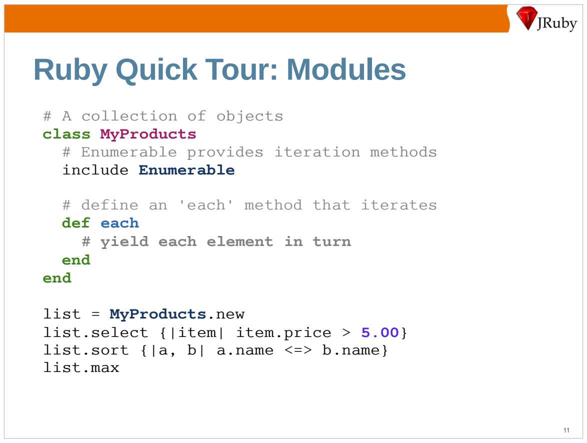


JRuby

Ruby Quick Tour: Blocks

```
# two formats: braces {} and do .. end
[1, 2, 3].each {|number| puts “I see #{number}“ }
[1, 2, 3].each do |number|
  puts “I see #{number}“
end

# methods that accept blocks
def foo
  yield “hello“
end
def foo2(&block)
  block.call(“hello“)
end
```

10

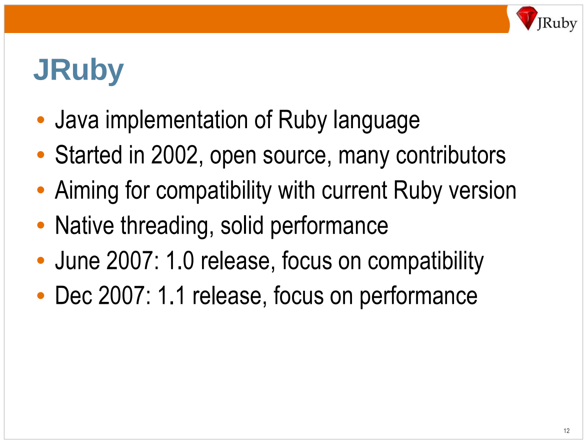


JRuby

Ruby Quick Tour: Modules

```
# A collection of objects
class MyProducts
  # Enumerable provides iteration methods
  include Enumerable

  # define an 'each' method that iterates
  def each
    # yield each element in turn
  end
end

list = MyProducts.new
list.select {|item| item.price > 5.00}
list.sort {|a, b| a.name <=> b.name}
list.max
```

11

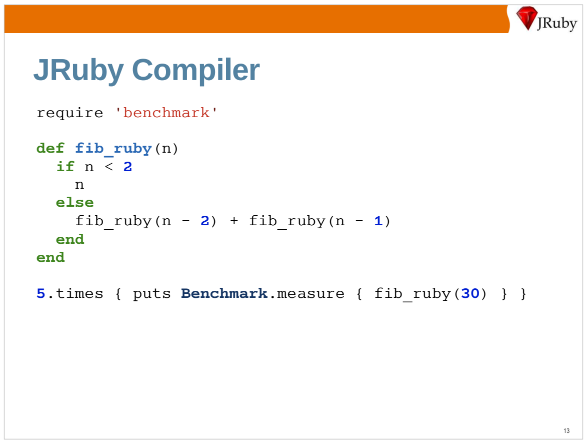


JRuby

JRuby

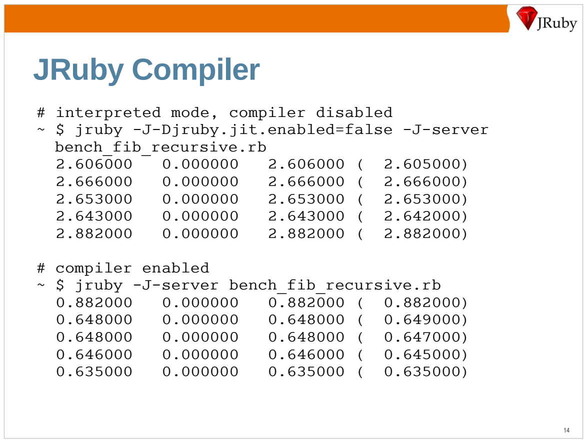
- Java implementation of Ruby language
- Started in 2002, open source, many contributors
- Aiming for compatibility with current Ruby version
- Native threading, solid performance
- June 2007: 1.0 release, focus on compatibility
- Dec 2007: 1.1 release, focus on performance

12



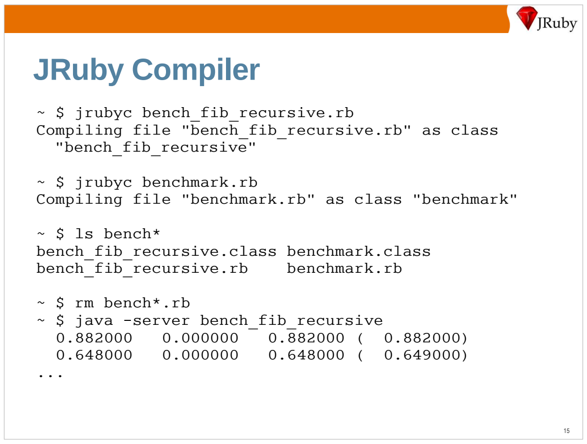
JRuby Compiler

```ruby
require 'benchmark'

def fib_ruby(n)
  if n < 2
    n
  else
    fib_ruby(n - 2) + fib_ruby(n - 1)
  end
end

5.times { puts Benchmark.measure { fib_ruby(30) } }
```



JRuby Compiler

```
# interpreted mode, compiler disabled
~ $ jruby -J-Djruby.jit.enabled=false -J-server
  bench_fib_recursive.rb
  2.606000   0.000000   2.606000 (  2.605000)
  2.666000   0.000000   2.666000 (  2.666000)
  2.653000   0.000000   2.653000 (  2.653000)
  2.643000   0.000000   2.643000 (  2.642000)
  2.882000   0.000000   2.882000 (  2.882000)

# compiler enabled
~ $ jruby -J-server bench_fib_recursive.rb
  0.882000   0.000000   0.882000 (  0.882000)
  0.648000   0.000000   0.648000 (  0.649000)
  0.648000   0.000000   0.648000 (  0.647000)
  0.646000   0.000000   0.646000 (  0.645000)
  0.635000   0.000000   0.635000 (  0.635000)
```



JRuby Compiler

```
~ $ jrubyc bench_fib_recursive.rb
Compiling file "bench_fib_recursive.rb" as class
  "bench_fib_recursive"

~ $ jrubyc benchmark.rb
Compiling file "benchmark.rb" as class "benchmark"

~ $ ls bench*
bench_fib_recursive.class benchmark.class
bench_fib_recursive.rb    benchmark.rb

~ $ rm bench*.rb
~ $ java -server bench_fib_recursive
  0.882000   0.000000   0.882000 (  0.882000)
  0.648000   0.000000   0.648000 (  0.649000)
...
```

JRuby

Calling Ruby from Java (Java 6)

```
// One-time load Ruby runtime
ScriptEngineManager factory =
    new ScriptEngineManager();
ScriptEngine engine =
    factory.getEngineByName("jruby");

// Evaluate JRuby code from string.
try {
    engine.eval("puts('Hello')");
} catch (ScriptException exception) {
    exception.printStackTrace();
}
```

JRuby

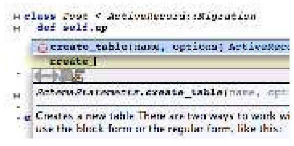
Calling Java from Ruby

```
# pull in Java support (require 'java' works too)
include Java

# import classes you need
import java.util.ArrayList
import javax.swing.JFrame

# use them like normal Ruby classes
list = ArrayList.new
frame = JFrame.new("Ruby SWINGS!")

# ...but with Ruby features added
list << frame
list.each {|f| f.set_size(200,200) }
```

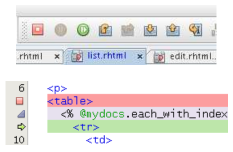
JRuby

JRuby Enables Tooling

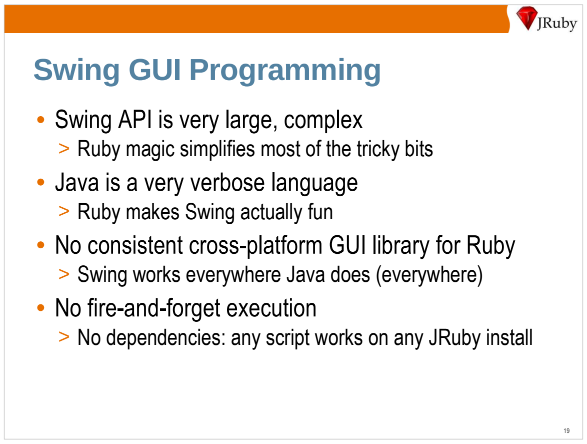
- JRuby's parser used by most Ruby IDEs
  - > NetBeans Ruby Support
  - > Eclipse RDT/RadRails/Aptana, DLTK, 3rd Rail
  - > IntelliJ
  - > JEdit



```
class Post < ActiveRecord::Migration
  def self.up
    create_table(name, options) ActiveRec
    create_
SchemaStatements.create_table(name, opt
Creates a new table There are two ways to work wi
use the block form or the regular form, like this:
```



```
<p>
<table>
  <% @mydocs.each_with_index
    <tr>
      <td>
```

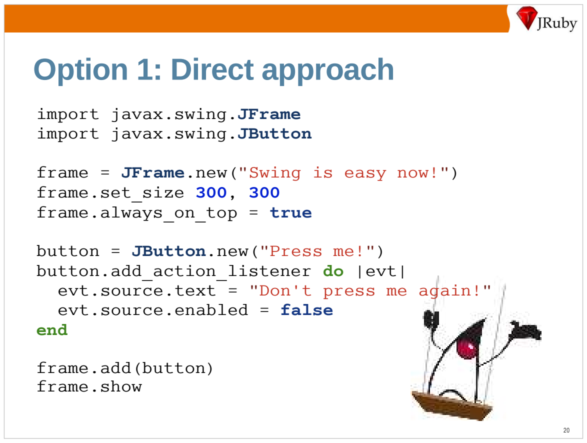


JRuby

Swing GUI Programming

- Swing API is very large, complex
  - > Ruby magic simplifies most of the tricky bits
- Java is a very verbose language
  - > Ruby makes Swing actually fun
- No consistent cross-platform GUI library for Ruby
  - > Swing works everywhere Java does (everywhere)
- No fire-and-forget execution
  - > No dependencies: any script works on any JRuby install

19

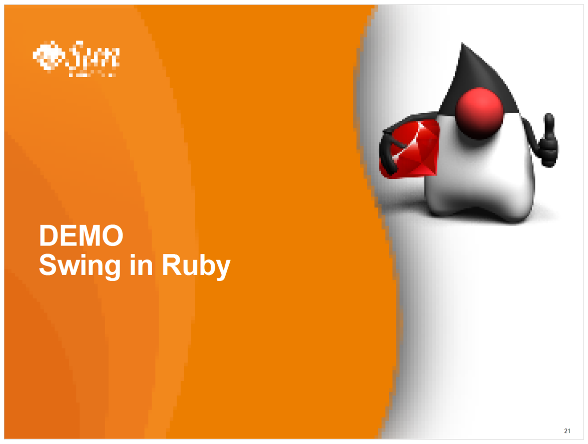


JRuby

Option 1: Direct approach

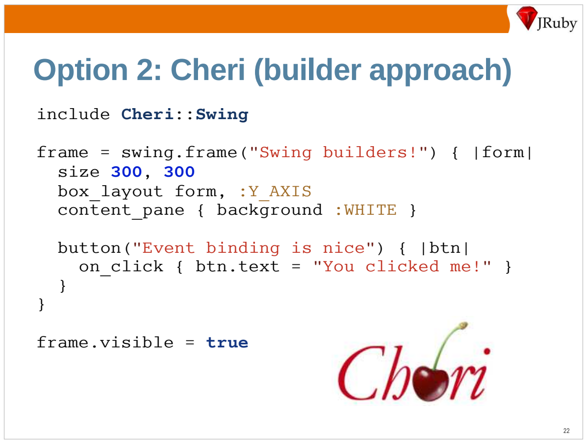
```
import javax.swing.JFrame
import javax.swing.JButton

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
```

20



DEMO
Swing in Ruby

21



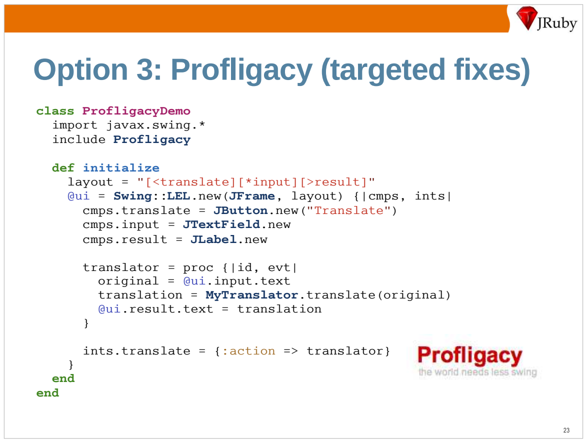
JRuby

Option 2: Cheri (builder approach)

```
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
```

Cheri

22



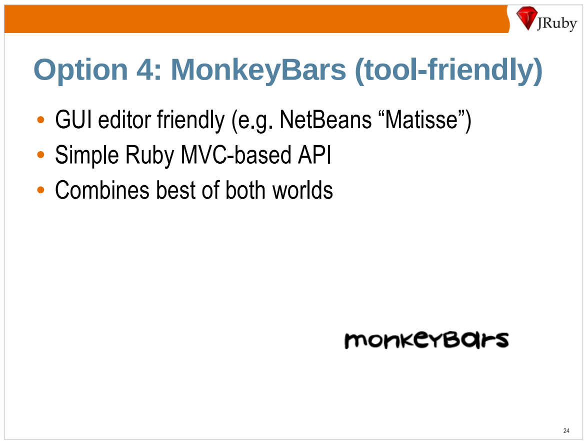
JRuby

Option 3: Profligacy (targeted fixes)

```
class ProfligacyDemo
  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
```

Profligacy
the world needs less swing

23



JRuby

Option 4: MonkeyBars (tool-friendly)

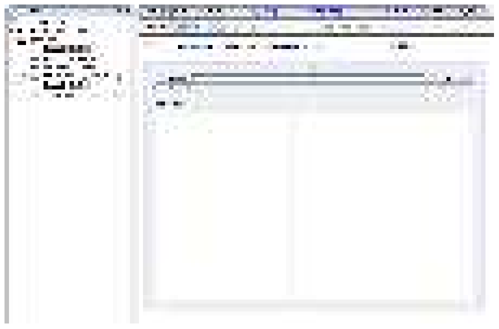
- GUI editor friendly (e.g. NetBeans “Matisse”)
- Simple Ruby MVC-based API
- Combines best of both worlds

monkeyBars

24

JRuby

MonkeyBars + NetBeans Matisse



JRuby

MonkeyBars Controller

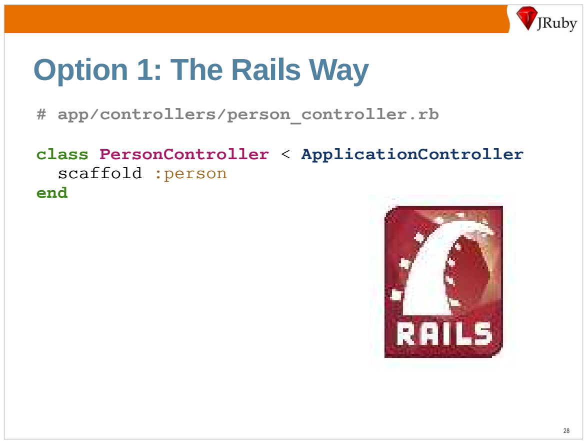
```
class RssController < Monkeybars::Controller
  set_view "RssView"
  set_model "RssModel"

  close_action :exit
  add_listener :type => :mouse,
     :components => ["goButton", "articleList"]

  def go_button_mouse_released(view_state, event)
    model.feed_url = view_state.feed_url
    content = Kernel.open(model.feed_url).read
    @rss = RSS::Parser.parse(content, false)

    model.articles = @rss.items.map {|art| art.title}
    model.article_text =
        CGI.unescapeHTML(@rss.items[0].description)
    update_view
  end
  ...
```

JRuby

Web applications

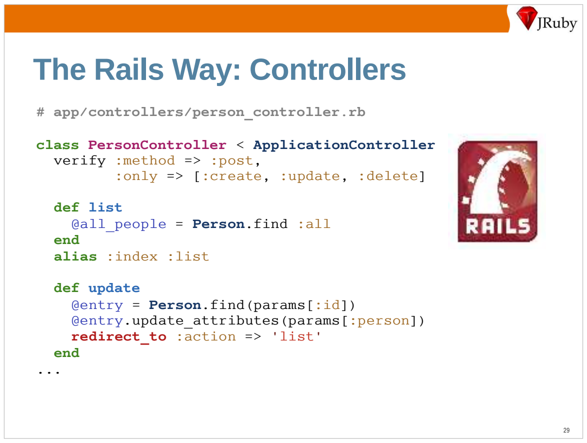
- Classic Java web dev is too complicated
  - > Modern frameworks follow Rails' lead
- Over-flexible, over-configured
  - > Conventions trump repetition and configuration
- Java is often too verbose for agile work
  - > Ruby makes even raw servlets look easy

Footnote position, 12 pts.



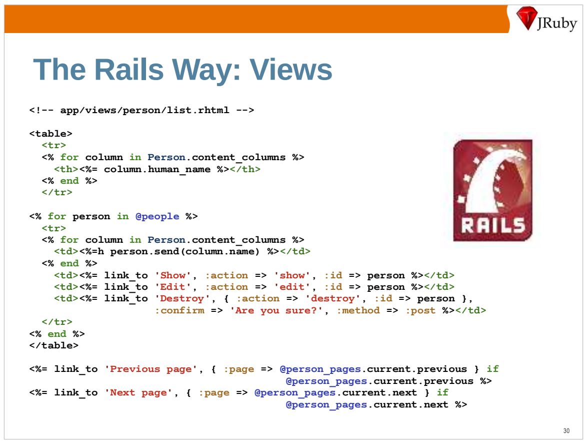
Option 1: The Rails Way

```
# app/controllers/person_controller.rb

class PersonController < ApplicationController
  scaffold :person
end
```



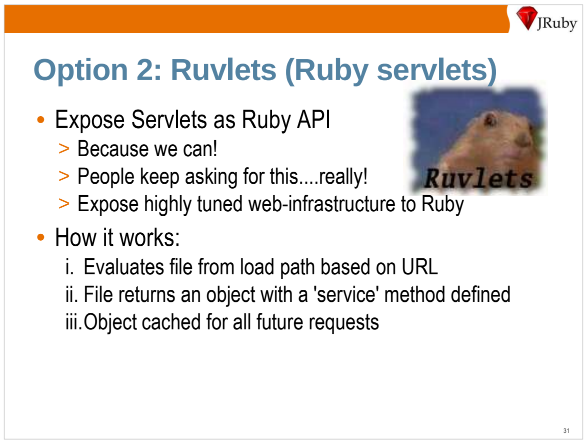
The Rails Way: Controllers

```
# app/controllers/person_controller.rb

class PersonController < ApplicationController
  verify :method => :post,
         :only => [:create, :update, :delete]

  def list
    @all_people = Person.find :all
  end
  alias :index :list

  def update
    @entry = Person.find(params[:id])
    @entry.update_attributes(params[:person])
    redirect_to :action => 'list'
  end
...
```



The Rails Way: Views

```
<!-- app/views/person/list.rhtml -->

<table>
  <tr>
  <% for column in Person.content_columns %>
    <th><%= column.human_name %></th>
  <% end %>
  </tr>

<% for person in @people %>
  <tr>
  <% for column in Person.content_columns %>
    <td><%=h person.send(column.name) %></td>
  <% end %>
    <td><%= link_to 'Show', :action => 'show', :id => person %></td>
    <td><%= link_to 'Edit', :action => 'edit', :id => person %></td>
    <td><%= link_to 'Destroy', { :action => 'destroy', :id => person },
                    :confirm => 'Are you sure?', :method => :post %></td>
  </tr>
<% end %>
</table>

<%= link_to 'Previous page', { :page => @person_pages.current.previous } if
                                         @person_pages.current.previous %>
<%= link_to 'Next page', { :page => @person_pages.current.next } if
                                         @person_pages.current.next %>
```

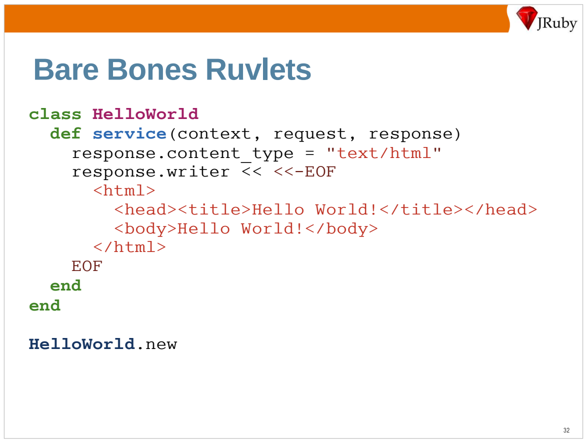


JRuby

Option 2: Ruvlets (Ruby servlets)

- Expose Servlets as Ruby API
  - > Because we can!
  - > People keep asking for this....really!
  - > Expose highly tuned web-infrastructure to Ruby
- How it works:
  - i. Evaluates file from load path based on URL
  - ii. File returns an object with a 'service' method defined
  - iii. Object cached for all future requests

31

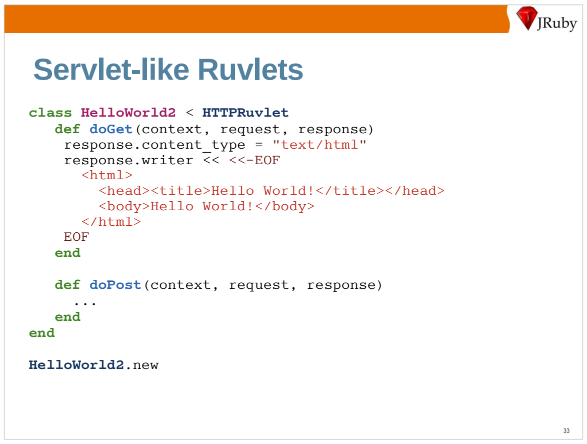


JRuby

Bare Bones Ruvlets

```
class HelloWorld
  def service(context, request, response)
    response.content_type = "text/html"
    response.writer << <<-EOF
      <html>
        <head><title>Hello World!</title></head>
        <body>Hello World!</body>
      </html>
    EOF
  end
end

HelloWorld.new
```

32

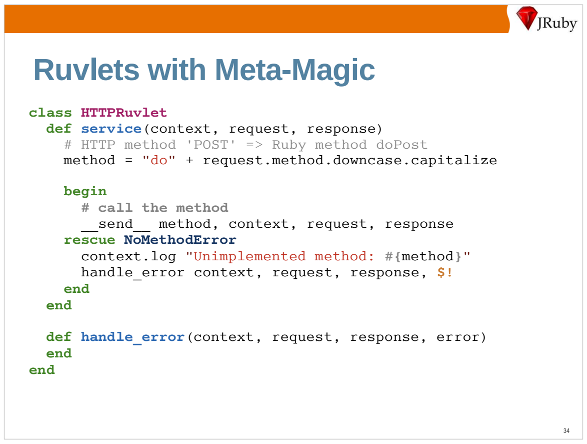


JRuby

Servlet-like Ruvlets

```
class HelloWorld2 < HTTPRuvlet
   def doGet(context, request, response)
    response.content_type = "text/html"
    response.writer << <<-EOF
      <html>
        <head><title>Hello World!</title></head>
        <body>Hello World!</body>
      </html>
    EOF
   end

   def doPost(context, request, response)
     ...
   end
end

HelloWorld2.new
```

33

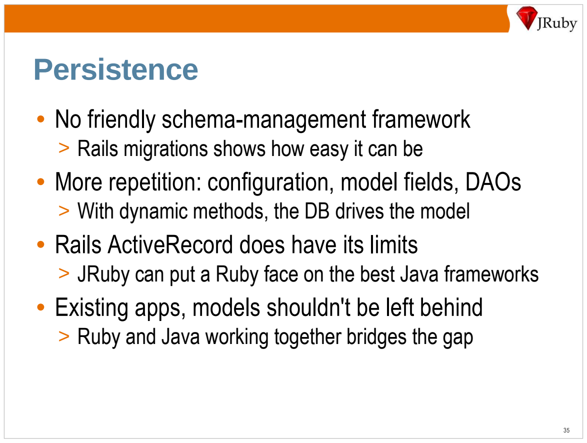


JRuby

Ruvlets with Meta-Magic

```
class HTTPRuvlet
  def service(context, request, response)
    # HTTP method 'POST' => Ruby method doPost
    method = "do" + request.method.downcase.capitalize

    begin
      # call the method
      __send__ method, context, request, response
    rescue NoMethodError
      context.log "Unimplemented method: #{method}"
      handle_error context, request, response, $!
    end
  end

  def handle_error(context, request, response, error)
  end
end
```

34

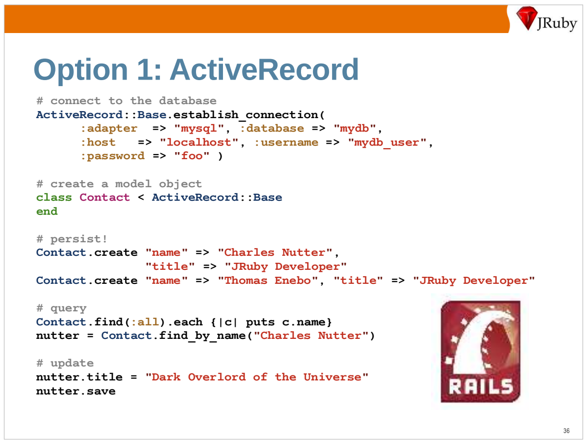


JRuby

Persistence

- No friendly schema-management framework
  - > Rails migrations shows how easy it can be
- More repetition: configuration, model fields, DAOs
  - > With dynamic methods, the DB drives the model
- Rails ActiveRecord does have its limits
  - > JRuby can put a Ruby face on the best Java frameworks
- Existing apps, models shouldn't be left behind
  - > Ruby and Java working together bridges the gap

35

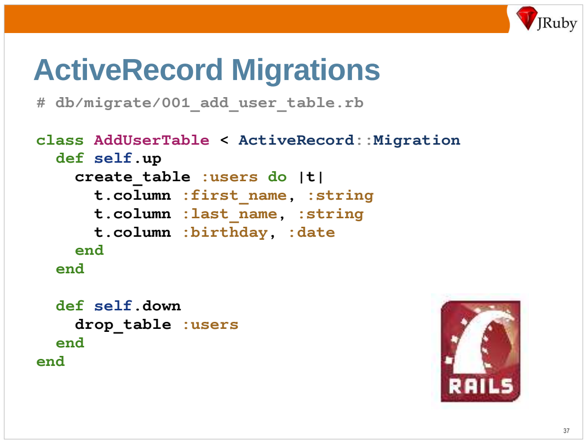


JRuby

Option 1: ActiveRecord

```
# connect to the database
ActiveRecord::Base.establish_connection(
      :adapter  => "mysql", :database => "mydb",
      :host     => "localhost", :username => "mydb_user",
      :password => "foo" )

# create a model object
class Contact < ActiveRecord::Base
end

# persist!
Contact.create "name" => "Charles Nutter",
               "title" => "JRuby Developer"
Contact.create "name" => "Thomas Enebo", "title" => "JRuby Developer"

# query
Contact.find(:all).each {|c| puts c.name}
nutter = Contact.find_by_name("Charles Nutter")

# update
nutter.title = "Dark Overlord of the Universe"
nutter.save
```

36



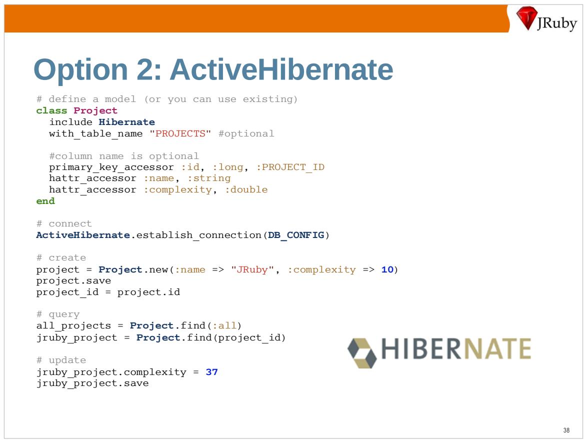
JRuby

ActiveRecord Migrations

```ruby
# db/migrate/001_add_user_table.rb

class AddUserTable < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.column :first_name, :string
      t.column :last_name, :string
      t.column :birthday, :date
    end
  end

  def self.down
    drop_table :users
  end
end
```

RAILS

37



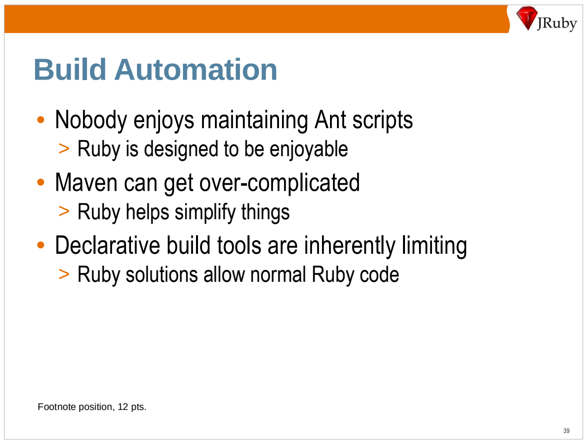
JRuby

Option 2: ActiveHibernate

```ruby
# define a model (or you can use existing)
class Project
  include Hibernate
  with_table_name "PROJECTS" #optional

  #column name is optional
  primary_key_accessor :id, :long, :PROJECT_ID
  hattr_accessor :name, :string
  hattr_accessor :complexity, :double
end

# connect
ActiveHibernate.establish_connection(DB_CONFIG)

# create
project = Project.new(:name => "JRuby", :complexity => 10)
project.save
project_id = project.id

# query
all_projects = Project.find(:all)
jruby_project = Project.find(project_id)

# update
jruby_project.complexity = 37
jruby_project.save
```

HIBERNATE

38



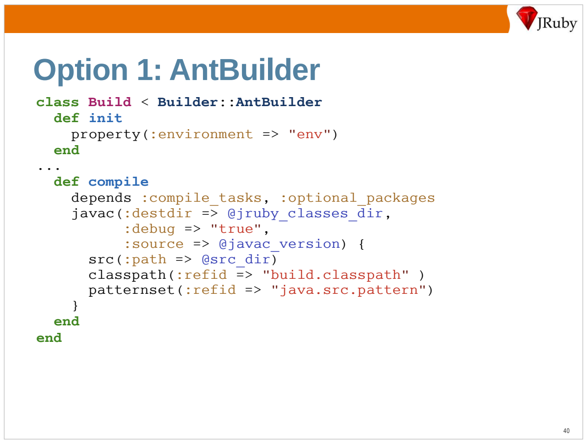
JRuby

Build Automation

- Nobody enjoys maintaining Ant scripts
  - > Ruby is designed to be enjoyable
- Maven can get over-complicated
  - > Ruby helps simplify things
- Declarative build tools are inherently limiting
  - > Ruby solutions allow normal Ruby code

Footnote position, 12 pts.

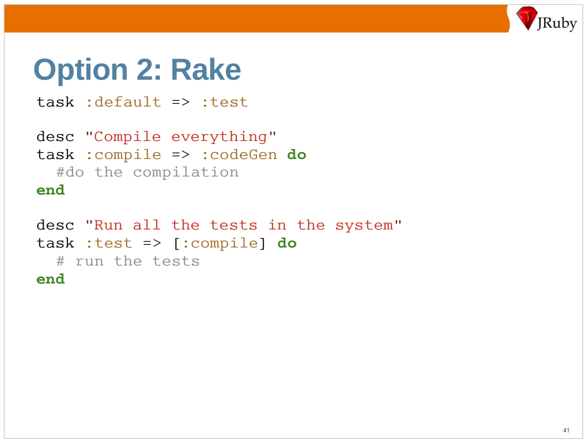
39



JRuby

Option 1: AntBuilder

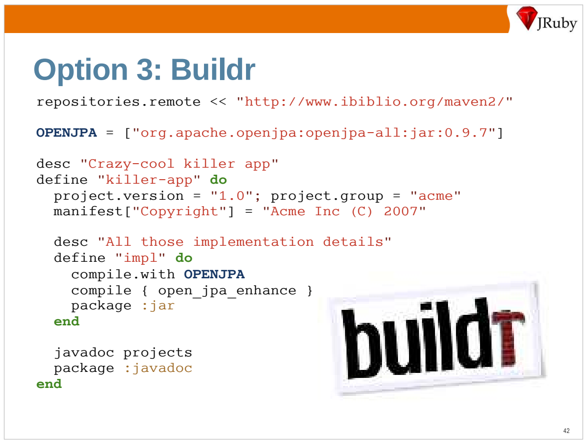
```
class Build < Builder::AntBuilder
  def init
    property(:environment => "env")
  end
...
  def compile
    depends :compile_tasks, :optional_packages
    javac(:destdir => @jruby_classes_dir,
          :debug => "true",
          :source => @javac_version) {
      src(:path => @src_dir)
      classpath(:refid => "build.classpath" )
      patternset(:refid => "java.src.pattern")
    }
  end
end
```



JRuby

Option 2: Rake

```
task :default => :test

desc "Compile everything"
task :compile => :codeGen do
  #do the compilation
end

desc "Run all the tests in the system"
task :test => [:compile] do
  # run the tests
end
```

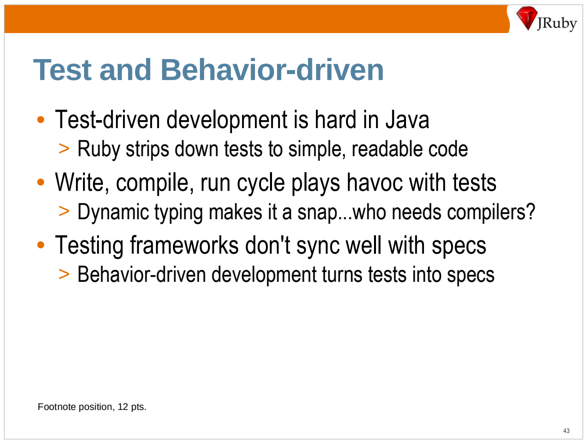


JRuby

Option 3: Buildr

```
repositories.remote << "http://www.ibiblio.org/maven2/"

OPENJPA = ["org.apache.openjpa:openjpa-all:jar:0.9.7"]

desc "Crazy-cool killer app"
define "killer-app" do
  project.version = "1.0"; project.group = "acme"
  manifest["Copyright"] = "Acme Inc (C) 2007"

  desc "All those implementation details"
  define "impl" do
    compile.with OPENJPA
    compile { open_jpa_enhance }
    package :jar
  end

  javadoc projects
  package :javadoc
end
```

buildr



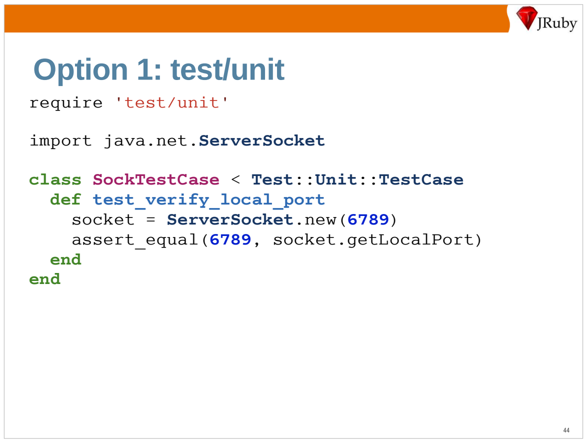
JRuby

Test and Behavior-driven

- Test-driven development is hard in Java
  - > Ruby strips down tests to simple, readable code
- Write, compile, run cycle plays havoc with tests
  - > Dynamic typing makes it a snap...who needs compilers?
- Testing frameworks don't sync well with specs
  - > Behavior-driven development turns tests into specs

Footnote position, 12 pts.

43

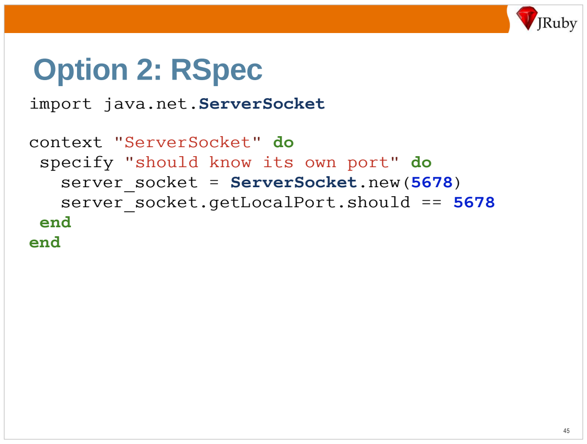


JRuby

Option 1: test/unit

```
require 'test/unit'

import java.net.ServerSocket

class SockTestCase < Test::Unit::TestCase
  def test_verify_local_port
    socket = ServerSocket.new(6789)
    assert_equal(6789, socket.getLocalPort)
  end
end
```

44

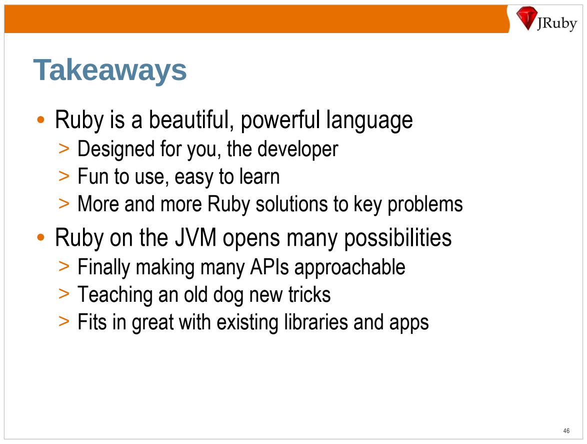


JRuby

Option 2: RSpec

```
import java.net.ServerSocket

context "ServerSocket" do
 specify "should know its own port" do
   server_socket = ServerSocket.new(5678)
   server_socket.getLocalPort.should == 5678
 end
end
```

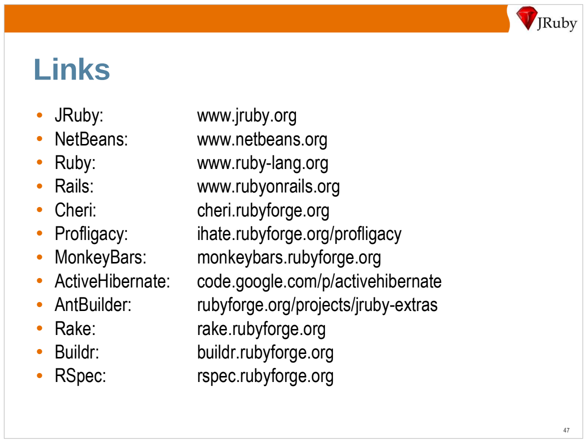
45



Takeaways

- Ruby is a beautiful, powerful language
  - > Designed for you, the developer
  - > Fun to use, easy to learn
  - > More and more Ruby solutions to key problems
- Ruby on the JVM opens many possibilities
  - > Finally making many APIs approachable
  - > Teaching an old dog new tricks
  - > Fits in great with existing libraries and apps

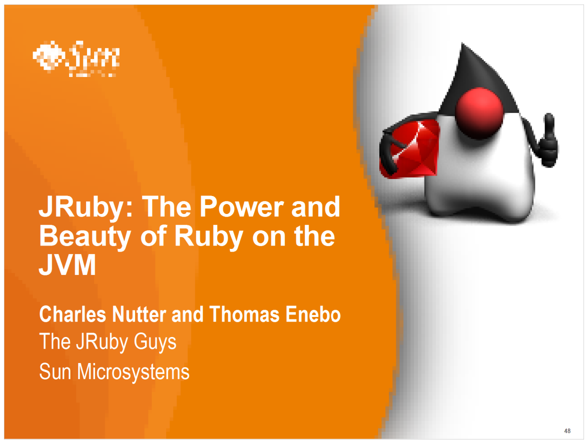
46



Links

JRuby:	www.jruby.org
NetBeans:	www.netbeans.org
Ruby:	www.ruby-lang.org
Rails:	www.rubyonrails.org
Cheri:	cheri.rubyforge.org
Profligacy:	ihate.rubyforge.org/profligacy
MonkeyBars:	monkeybars.rubyforge.org
ActiveHibernate:	code.google.com/p/activehibernate
AntBuilder:	rubyforge.org/projects/jruby-extras
Rake:	rake.rubyforge.org
Buildr:	buildr.rubyforge.org
RSpec:	rspec.rubyforge.org

47



JRuby: The Power and Beauty of Ruby on the JVM

Charles Nutter and Thomas Enebo
The JRuby Guys
Sun Microsystems

48