



Internet Technologies

Ruby and Ruby on Rails



Ruby on Rails

Material for this presentation was taken from Sebesta (PWWW, course text) and “Agile Web Development with Rails” by Ruby, Thomas and Hansson, third edition.

Notes on Ruby From Sebesta's "Programming The World Wide Web"

- Designed in Japan by Yukihiro Matsumoto
- Released in 1996
- Designed to replace Perl and Python
- Rails, a web application development framework , was written in and uses Ruby
- Ruby is general purpose but probably the most common use of Ruby is Rails
- Rails was developed by David Heinemeier and released in 2004
- Basecamp (project management) is written in RoR

General Notes on Ruby(1)

- To get started install RVM (Ruby Version Manager)
- Use ri command line tool to browse documentation (e.g., ri Integer).
- Use rdoc to create documentation (like Javadoc)
- Ruby is a pure object-oriented language.
- All variables reference objects.
- Every data value is an object.
- References are typeless.
- All that is ever assigned in an assignment statement is the address of an object.
- There is no way to declare a variable.
- A scalar variable that has not been assigned a value has the value nil.

General Notes on Ruby(2)

- Three categories of data types - scalars, arrays and hashes
- Two categories of scalars - numerics and character strings
- Everything (even classes) is an object.
- Numeric types inherit from the Numeric class
- Float and Integer inherit from Numeric
- Fixnum (32 bits) and Bignum inherit from Integer
- All string literals are String objects
- The null string may be denoted as "" or as "".
- The String class has over 75 methods

Interactive Environment

```
$irb
```

```
>> miles = 1000
```

```
=> 1000
```

```
>> milesPerHour = 100
```

```
=> 100
```

```
>> "Going #{miles} miles at #{milesPerHour} MPH takes #{1/milesPerHour.to_f*miles} hours"
```

```
=> "Going 1000 miles at 100 MPH takes 10.0 hours"
```

More interactive Ruby

```
$irb
```

```
>> miles = 1000
```

```
=> 1000
```

```
>> s = "The number of miles is #{miles}"
```

```
=> "The number of miles is 1000"
```

```
>> s
```

```
=> "The number of miles is 1000"
```

The Math Module

```
>> y = Math.sin(0)
```

```
=> 0.0
```

```
>> y = Math.sin(Math::PI/2.0)
```

```
=> 1.0
```

Non-Interactive Ruby

Save as one.rb and run with ruby one.rb

```
a = "hi"  
b = a  
puts a  
puts b  
b = "OK"  
puts a  
puts b
```

Output
=====
hi
hi
hi
OK

References are Typeless

```
a = 4  
puts a  
a = "hello"  
puts a
```

```
Output  
=====  
4  
hello
```

C Style Escapes

```
puts "Hello\nInternet\tTechnologies"
```

Hello

Internet

Technologies

Converting Case

```
a = "This is mixed case."  
puts a.upcase  
puts a  
puts a.upcase!  
puts a
```

```
THIS IS MIXED CASE.  
This is mixed case.  
THIS IS MIXED CASE.  
THIS IS MIXED CASE.
```

Testing Equality(1)

```
b = "Cool course" == "Cool course" # same content
puts b
b = "Cool course".equal?("Cool course") #same object
puts b
puts 7 == 7.0    # same value
puts 7.eql?(7.0) # same value and same type
```

Output

=====

true

false

true

false

Testing Equality(2)

```
a = "Ruby is cool."  
b = "Ruby is cool."  
c = b  
if a == b  
  puts "Cool"  
else  
  puts "Oops"  
end  
if c.equal?(b)  
  puts "Too cool"  
else  
  puts "Big Oops"  
end  
if c.equal?(a)  
  puts "Way cool"  
else  
  puts "Major Oops"  
end
```

What's the output?

```
$ruby test.rb  
Cool  
Too cool  
Major Oops
```

Reading The Keyboard

```
puts "Who are you?"  
name = gets #include entered newline  
name.chomp! #remove the newline  
puts "Hi " + name + ", nice meeting you."
```

Interaction

=====

Who are you?

Mike

Hi Mike, nice meeting you.

Reading Integers

```
#to_i returns 0 on strings that are not integers  
puts "Enter two integers on two lines and I'll add them"  
a = gets.to_i  
b = gets.to_i  
puts a + b
```

Interaction

```
=====
```

```
Enter two integers on two lines and I'll add them
```

```
2
```

```
4
```

```
6
```

Conditions (1)

```
a = 5
if a > 4
  puts "Inside the if"
  a = 2
end
puts "a == " + a.to_s(10)
```

Output

=====

Inside the if

a == 2

Conditions (2)

```
a = 5
unless a <= 4
  puts "Inside the if"
  a = 2
end
puts "a == " + a.to_s(10)
```

Output

=====

Inside the if

a == 2

Conditions (3)

```
a = 5
if a <= 4
  puts "Inside the if"
  a = 2
else
  puts "a == " + a.to_s(10)
end
```

Output

=====

```
a == 5
```

Conditions (4)

```
a = 5
if a <= 4
  puts "Inside the if"
  a = 2
elsif a <= 9
  puts "Inside the elsif"
else
  puts "Inside else"
end
```

Output

=====

Inside the elsif

Conditions (5)

```
a = 5
case a
when 4 then
  puts "The value is 4"
when 5
  puts "The value is 5"
end
```

Output

=====

The value is 5

Conditions (6)

```
a = 2
case a
when 4 then
  puts "The value is 4"
when 5
  puts "The value is 5"
else
  puts "OK"
end
```

Output

=====

OK

Statement Modifiers

Suppose the body of an if or while has a single statement. Then, you may code it as:

```
puts "This is displayed" if 4 > 3
j = 0
puts j+1 if j == 0
j = j + 1 while j < 100
puts j
```

```
This is displayed
1
100
```

Case/When with Range

```
a = 4
case a
when 4 then
  # after a match we are done
  puts "The value is 4"
when (3..500)
  puts "The value is between 3 and 500"
else
  puts "OK"
end
```

Output

=====

The value is 4

Value of Case/When (1)

```
year = 2009
leap = case
when year % 400 == 0 then true
when year % 100 == 0 then false
else year % 4 == 0
end
puts leap
```

Output

=====

false

Value of Case/When(2)

```
year = 2009
puts case
when year % 400 == 0 then true
when year % 100 == 0 then false
else year % 4 == 0
end
```

What's the output?

```
Output
=====
false
```

While

```
top = 100
now = 1
sum = 0
while now <= top
    sum = sum + now
    now += 1
end
puts sum
```

Output

=====

5050

Until

```
j = 100
until j < 0
  j = j - 1
end
puts j
```

Output

=====

-1

Arrays(1)

<code>a = [1,2,3,4,5]</code>	Output
<code>puts a[4]</code>	=====
<code>x = a[0]</code>	5
<code>puts x</code>	1
<code>a = ["To","be","or","not","to","be"]</code>	To
<code>j = 0</code>	be
<code>while j < 6</code>	or
<code>puts a[j]</code>	not
<code>j = j + 1</code>	to
<code>end</code>	be

Arrays(2)

```
a = [1,2,3,4,5]
j = 0
while j < 5
    a[j] = 0
    j = j + 1
end
puts a[1]
```

Output

=====

0

Arrays(3)

```
somedays = ["Friday","Saturday","Sunday","Monday"]  
puts somedays.empty?  
puts somedays.sort
```

Output

=====

```
false  
Friday  
Monday  
Saturday  
Sunday
```

Arrays(4)

```
a = [5,4,3,2,1]  
a.sort!  
puts a
```

What's the output?

1
2
3
4
5

Arrays(5) Set Intersection &

a = [5,4,3,2,1]

b = [5,4,1,2]

c = a & b

puts c

What's the output?

5
4
2
1

Arrays(6) Implement a Stack

```
x = Array.new
k = 0
while k < 5
  x.push(k)
  k = k + 1
end
```

```
while !x.empty?()
  y = x.pop
  puts y
end
```

What's the output?

4
3
2
1
0

Arrays and Ranges(1)

```
# Create an array from a Ruby range
```

```
# Create range
```

```
a = (1..7)
```

```
puts a
```

```
#create array
```

```
b = a.to_a
```

```
puts b
```

Output

=====

1..7

1

2

3

4

5

6

7

Arrays and Ranges(2)

#Ranges are objects with methods

```
v = 'aa'..'az'
```

```
u = v.to_a
```

```
puts v
```

```
puts u
```

Output

=====

aa..az

aa

ab

ac

:

:

aw

ax

ay

az

Arrays and Ranges(3)

```
a = 1..10;  
b = 10..20  
puts a  
puts b  
c = a.to_a & b.to_a  
puts c
```

What is the output?

```
1..10  
10..20  
10
```

Hashes (1)

```
# Hashes are associative arrays
# Each data element is paired with a key
# Arrays use small ints for indexing
# Hashes use a hash function on a string
```

```
kids_ages = {"Robert" => 16, "Cristina" =>14, "Sarah" => 12, "Grace" =>8}
puts kids_ages
```

Output

=====

Sarah12Cristina14Grace8Robert16

Hashes(2) Indexing

```
kids_ages = {"Robert" => 16, "Cristina" =>14, "Sarah" => 12, "Grace" =>8}  
puts kids_ages["Cristina"]
```

Output

=====

14

Hashes(3) Adding & Deleting

```
kids_ages = {"Robert" => 16, "Cristina" =>14, "Sarah" => 12, "Grace" =>8}  
kids_ages["Daniel"] = 15  
kids_ages.delete("Cristina")  
puts kids_ages
```

Output

=====

Daniel15Sarah12Grace8Robert16

Hashes (4) Taking The Keys

```
kids_ages = {"Robert" => 16, "Cristina" =>14, "Sarah" => 12, "Grace" =>8}  
m = kids_ages.keys  
kids_ages.clear  
puts kids_ages  
puts m
```

```
Output  
=====  
Sarah  
Cristina  
Grace  
Robert
```

Hashes (5)

```
grade = Hash.new  
grade["Mike"] = "A+"  
grade["Sue"] = "A-"  
puts grade["Mike"]
```

What's the output?

A+

Hashes with Symbols

(1) `s = {:u => 3, :t => 4, :xyz => "Cristina" }`
`puts s[:xyz]`
Cristina

(2) A Ruby symbol is an instance of the Symbol class.

(3) In Rails we will see..

```
<%= link_to("Edit", :controller => "editcontroller", :action => "edit") %>
```

The first parameter is a label on the link and the second parameter is a hash.

(4) The `link_to` method checks if the symbol `:controller` maps to a value and if so, is able to find “editcontoller” . Same with `:action`.

Methods

```
# Methods may be defined outside classes
# to form functions or within classes to
# form methods. Methods must begin with lower case
# letters.
# If no parameters then parentheses are omitted.
```

```
def testMethod
  return Time.now
end
```

```
def testMethod2
  Time.now
end
```

```
puts testMethod
puts testMethod2
```

Output

=====

Tue Feb 10 22:12:44 -0500 2009

Tue Feb 10 22:12:44 -0500 2009

Methods Local Variables

```
def looper
  i = 0
  while i < 5
    puts i
    i = i + 1
  end
end
```

```
looper
```

What's the output?

```
Output
=====
0
1
2
3
4
```

Scalars Are Pass By Value

#scalars are pass by value

```
def looper(n)
  i = 0
  while i < n
    puts i
    i = i + 1
  end
end
```

Output
=====
0
1
2

```
looper(3)
```

Parenthesis Are Optional

#scalars are pass by value

```
def looper(n)
  i = 0
  while i < n
    puts i
    i = i + 1
  end
end
```

Output
=====
0
1
2

```
looper 3
```

Code Blocks (1)

```
def looper(n)
  i = 0
  while i < n
    yield i
    i = i + 1
  end
end
```

```
looper (3) do |x| puts x end
looper (4) {|x| puts x }
```

```
Output
=====
0
1
2
0
1
2
3
```

Think of the **code block** as a method with no name.

Code Blocks (2)

```
def looper
  i = 0
  n = 4
  while i < n
    yield i
    i = i + 1
  end
end
```

```
Value 0
Value 1
Value 2
Value 3
```

```
looper{|x| puts "Value #{x}" }
```

Think of the **code block** as a method with no name.

Code Blocks (3)

```
def interest(balance)
  yield balance
end
```

What's the output?

```
rate = 0.15
```

interest is 150.0

```
interestAmt = interest(1000.0) { |bal| bal * rate }
print "interest is #{interestAmt}"
```

```
rate = 0.12
```

```
interestAmt = interest(1000.0) { |bal| bal * rate + 1.0 }
print "interest is #{interestAmt}"
```

Code Blocks (4)

Many Ruby methods take blocks.

```
[1,2,3,4,5].each {|x| puts "Doubled = #{x*2}"}
```

```
Doubled = 2
```

```
Doubled = 4
```

```
Doubled = 6
```

```
Doubled = 8
```

```
Doubled = 10
```

Code Blocks (5)

Many Ruby methods take blocks.

Collect returns an array. What's the output?

```
t = [1,2,3,4,5].collect {|x| x*2}
```

```
puts t
```

```
t = [1,2,3,4,5].collect do |x| x + 1 end
```

```
puts t
```

2

4

6

8

10

2

3

4

5

6

Code Blocks (6)

XML Processing and XPATH predicates.

```
# We want to read the schedule for this class.  
# For command line processing use ARGV[0] rather than hard coding the name.  
  
require "rexml/document" # Ruby Electric XML comes with standard distribution  
file = File.new( "schedule.xml" )  
doc = REXML::Document.new(file)  
doc.elements.each("//Slides/Topic[.='Ruby and Ruby On Rails']") { |element| puts element }
```

<Topic>Ruby and Ruby On Rails</Topic>

Or Remotely

```
require "rexml/document"  
require 'open-uri'  
remoteFile = open('http://www.andrew.cmu.edu/user/mm6/95-733/schedule.xml') {|f| f.read }  
doc = REXML::Document.new(remoteFile)  
doc.elements.each("//Slides/Topic[.='Ruby and Ruby On Rails']") {|e| puts e }
```

Code Blocks(7)

integers are objects with methods that take code blocks.
4.times {puts "Yo!"}

Output

=====

Yo!

Yo!

Yo!

Yo!

Code Blocks (8) Closures

```
def print(x)
  x.call
end
```

What's the output?

```
z = "Mike"
a = lambda { puts z }
print(a)
```

Mike
Sue
Sue

```
z = "Sue"
b = lambda { puts z }
```

```
print(a)
print(b)
```

Arrays and Hashes Are Pass By Reference

```
def coolsorter(n)  
  n.sort!  
end
```

```
n = [5,4,3,2,1]  
coolsorter(n)  
puts n
```

What's the output?

Output

=====

1

2

3

4

5

Classes

```
# Classes and constants must begin with  
# an uppercase character.  
# Instance variable begin with an "@" sign.  
# The constructor is named initialize
```

```
class Student  
  def initialize(n = 5)  
    @course = Array.new(n)  
  end  
  def getCourse(i)  
    return @course[i]  
  end  
  def setCourse(c,i)  
    @course[i] = c  
  end  
end
```

```
individual = Student.new(3)  
individual.setCourse("Chemistry", 0)  
puts individual.getCourse(0)
```

```
Output  
=====  
Chemistry
```

Simple Inheritance

```
class Mammal
  def breathe
    puts "inhale and exhale"
  end
end
```

```
class Cat<Mammal
  def speak
    puts "Meow"
  end
end
```

```
class Dog<Mammal
  def speak
    puts "Woof"
  end
end
```

```
peanut = Dog.new
sam = Cat.new
peanut.speak
sam.speak
sam.breathe
```

Output
=====

Woof
Meow
inhale and exhale

Ruby has no multiple inheritance.

Self makes a method a class method. @@ is a class variable.

```
class Mammal
  @@total = 0
  def initialize
    @@total = @@total + 1
  end
  def breathe
    puts "inhale and exhale"
  end
  def self.total_created
    return @@total
  end
end
```

```
class Cat<Mammal
  def speak
    puts "Meow"
  end
end
class Dog<Mammal
  def speak
    puts "Woof"
  end
end
peanut = Dog.new
sam = Cat.new
peanut.speak
sam.speak
sam.breathe

puts Mammal.total_created
```

```
Woof
Meow
inhale and exhale
2
```

Public, Private and Protected

```
class Mammal
  def breathe # method is public
    puts "inhale and exhale"
  end

  protected

  def move # method available to inheritors
    puts "wiggle wiggle"
  end

  private

  def sleep # private method
    puts "quiet please"
  end
end
```

```
class Cat<Mammal
  def speak
    move
    puts "Meow"
  end

end

class Dog<Mammal
  def speak
    move
    puts "Woof"
  end

end

peanut = Dog.new
sam = Cat.new
peanut.speak
sam.speak
sam.breathe
```

Modules

Modules group together methods and constants.

A module has no instances or subclasses.

To call a module's method, use the module name, followed by a dot, followed by the name of the method.

To use a module's constant, use the module name, followed by two colons and the name of the constant.

Think “namespace”.

Modules

```
module Student
  MAXCLASSSIZE = 105
  class GradStudent
    def work
      puts "think, present, present,.."
    end
    def eat
      puts "pizza"
    end
    def sleep
      puts "zzzzz"
    end
  end
end
end
x = 6
mike = Student::GradStudent.new
mike.work if x <= Student::MAXCLASSSIZE
```

ruby onemodule.rb
think, present, present,..

Include this module with
require. Similar to Java's
import or C's #include.

Ruby Supports Closures

A **closure** is a first class function with free variables that are bound in the lexical environment.

(From Wikipedia)

Put another way: A **closure** is a method with two properties:

1. It can be passed around and can be called at a later time and
2. It has access to variables that were in scope at the time the method was created.

Closures in Javascript

```
<html>
  <head>
    <script type="text/javascript">
      // define printMessage to point to a function
      var printMessage = function (s) {
        alert("In printMessage() for " + s)
        var f = function () {
          alert(s + ' was pressed.');
```

```
        }
        return f;
      }
      // call function pointed to be printMessage
      // with a parameter.
      // A pointer to a function is returned.
      // The inner function has a copy of s.
      buttonA = printMessage('A')
      buttonB = printMessage("B")
      buttonC = printMessage("C")
```

Closures in Javascript

```
</script>
  <title>Closure example</title>
</head>

<body>

  <!-- call the function pointed to by the variable -->
  <button type="button" onClick = "buttonA()">A Button Click Me!</button>
  <button type="button" onClick = "buttonB()" >B Button Click Me!</button>
  <button type="button" onClick = "buttonC()" >C Button Click Me!</button>

</body>
</html>
```

What's the output?

Closures in Javascript

On page load:

In printMessage() for A

In printMessage() for B

In printMessage() for C

Three buttons appear

Click A => “A was pressed”

Click B=> “B was pressed”

Another Ruby Closure

```
class ACoolClass
  def initialize(value1)
    @value1 = value1
  end
  def set(i)
    @value1 = i
  end
  def display(value2)
    lambda { puts "Value1: #{@value1}, Value2: #{value2}" }
  end
end
def caller(some_closure)
  some_closure.call
end
obj1 = ACoolClass.new(5)
printer = obj1.display("some values")
caller(printer)
printer.call
obj1.set(3)
printer.call
```

Quiz: What's the output?

Another Ruby Closure (2)

```
class ACoolClass
  def initialize(value1)
    @value1 = value1
  end
  def set(i)
    @value1 = i
  end
  def display(value2)
    lambda { puts "Value1: #{@value1}, Value2: #{value2}" }
  end
end
def caller(some_closure)
  some_closure.call
end
obj1 = ACoolClass.new(5)
printer = obj1.display("some values")
caller(printer)
printer.call
obj1.set(3)
printer.call
```

ruby closure.rb

Value1: 5, Value2: some values

Value1: 5, Value2: some values

Value1: 3, Value2: some values

How about this one?

```
class SomeClass
  def initialize(value1)
    @value1 = value1
  end

  def value_printer(value2)
    i = 65
    lambda {puts "Value1: #{@value1}, Value2: #{value2}, Value3: #{i}"; i = i + 1}
  end
end

def caller(some_closure)
  some_closure.call
end
```

What's the output?

```
some_class = SomeClass.new(5)
```

```
printer = some_class.value_printer("some value")  
printer2 = some_class.value_printer("another value")
```

```
puts "Nothing called yet"  
caller(printer)  
caller(printer)  
caller(printer)
```

```
caller(printer2)
```

What's the output?

```
some_class = SomeClass.new(5)
```

```
printer = some_class.value_printer("some value")  
printer2 = some_class.value_printer("another value")
```

```
puts "Nothing called yet"
```

```
caller(printer)
```

```
caller(printer)
```

```
caller(printer)
```

```
caller(printer2)
```

```
ruby testClosure.rb
```

```
Nothing called yet
```

```
Value1: 5, Value2: some value, Value3: 65
```

```
Value1: 5, Value2: some value, Value3: 66
```

```
Value1: 5, Value2: some value, Value3: 67
```

```
Value1: 5, Value2: another value, Value3: 65
```

Pattern Matching

#Pattern matching using regular expressions

```
line = "http://www.andrew.cmu.edu"  
loc = line =~ /www/  
puts "www is at position #{loc}"
```

Output

=====

www is at position 7

Regular Expressions

This split is based on a space, period or comma followed
by zero or more whitespace.

```
line2 = "www.cmu.edu is where it's at."  
arr = line2.split(/[ .,]\s*/)  
puts arr
```

Output
=====
www
cmu
edu
is
where
it's
at

Ruby On Rails(1)

“A *framework* is a system in which much of the more or less standard parts are furnished by the framework, so that they do not need to be written by the application developer.” Source: Sebesta

Like Tapestry and Struts, Rails is based on the Model View Controller architecture for applications.

MVC developed at XeroxPARC by the Smalltalk group.

Ruby On Rails (2)

- Two fundamental principles:
 - DRY (Don' t Repeat Yourself)
 - Convention over configuration
- Rails is a product of a software development paradigm called *agile development*.
- Part of being agile is quick development of working software rather than the creation of elaborate documentation and *then* software.

Model View Controller

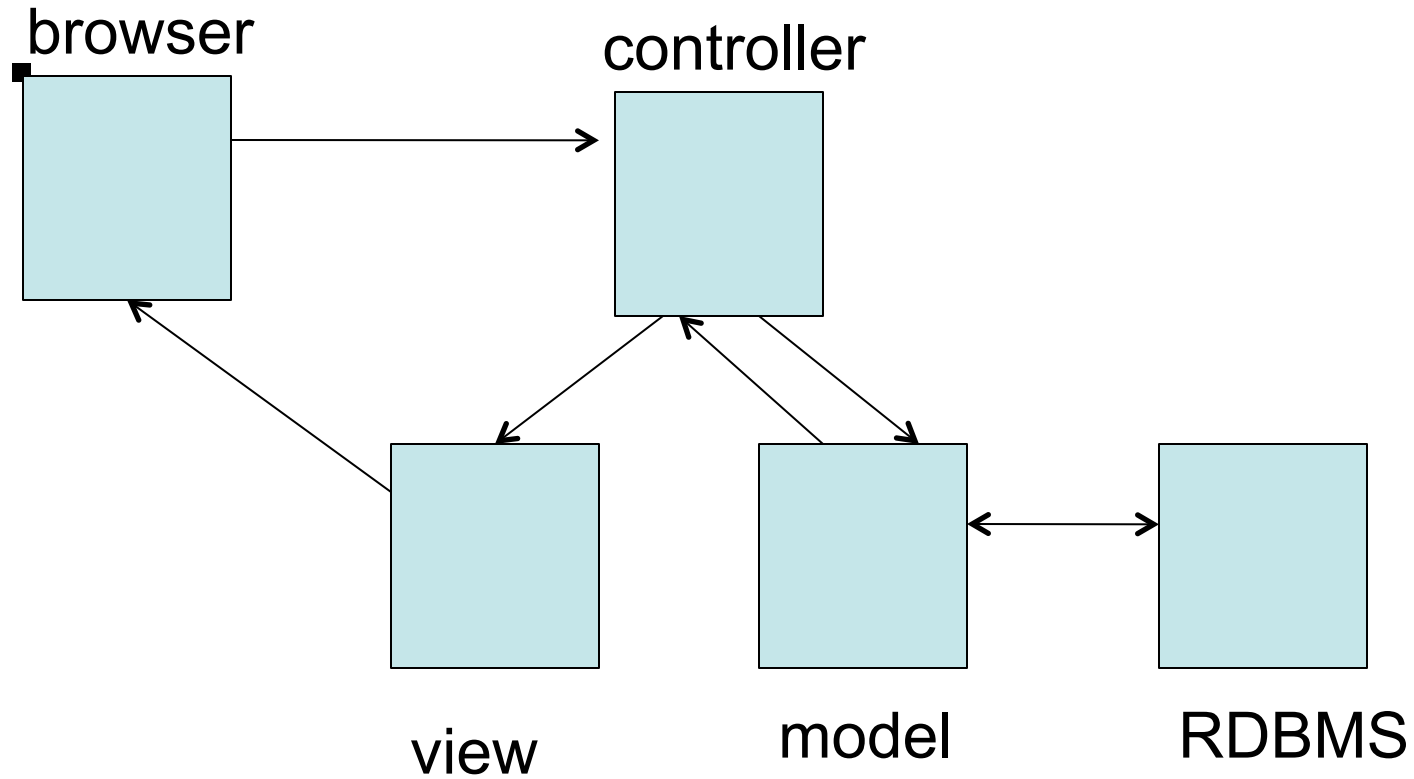
- The *Model* is the data and any enforced constraints on the data. Rails uses Object Relationship Mapping. A class corresponds to a table. An object corresponds to a row.
- The *View* prepares and presents results to the user.
- The *Controller* performs required computations and controls the application.

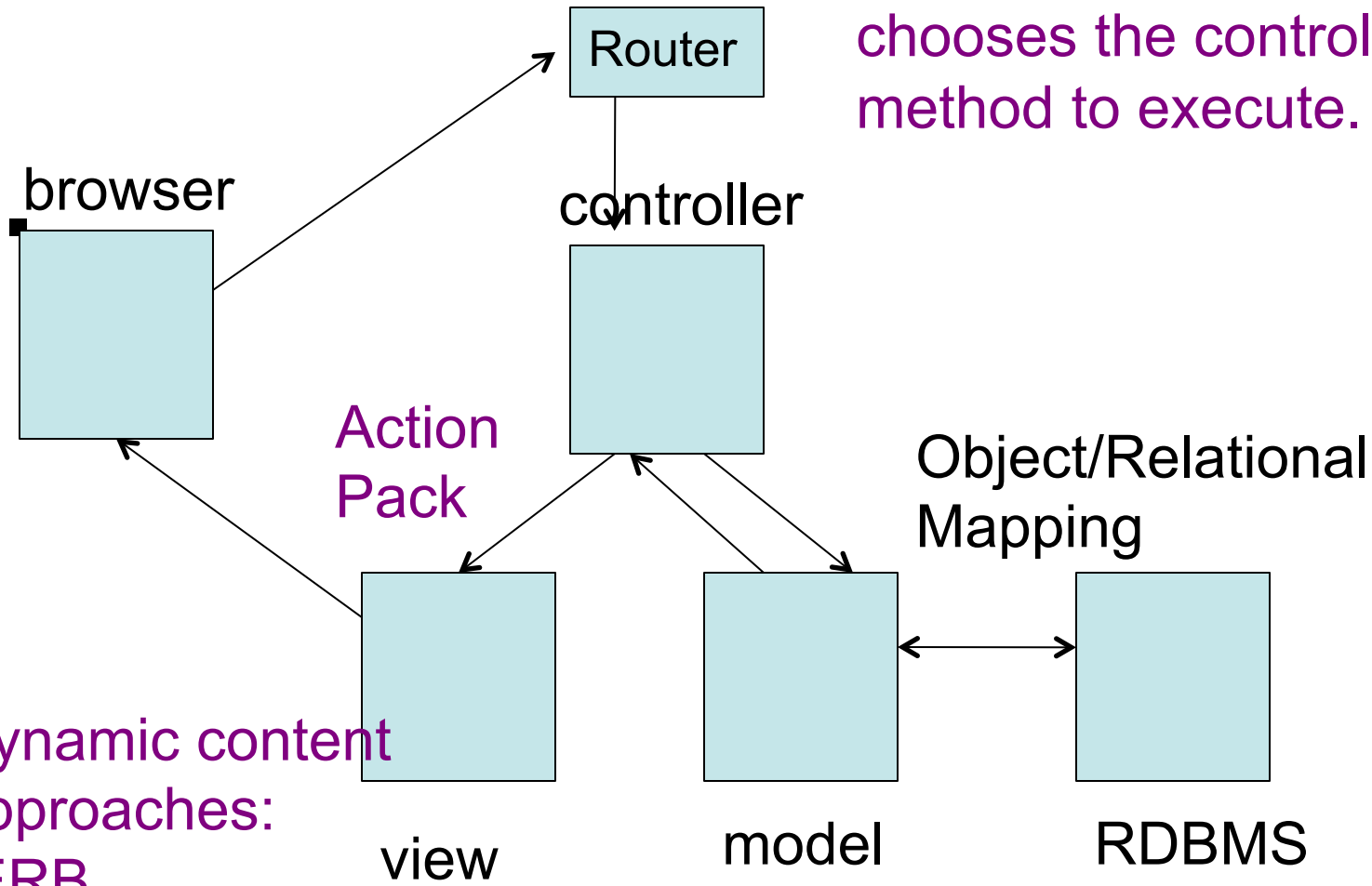
Source: Sebesta

Model View Controller

- Rails is a web-application and persistence framework.
- MVC splits the view into "dumb" templates that are primarily responsible for inserting pre-built data in between HTML tags.
- The model contains the "smart" domain objects (such as Account, Product, Person).
- The model holds all the business logic and knows how to persist itself to a database.
- The controller handles the incoming requests (such as Save New Account, Update Product, Show Person) by manipulating the model and directing data to the view.

Model View Controller





Recognizes URL's and chooses the controller and method to execute.

Dynamic content approaches:

- ERB
- XML Builder
- RJS for Javascript

ActiveRecord

Rails Tools

- Rails provides command line tools.
The following command creates many directories and subdirectories including models, views, and controllers:

```
$rails new greet
```

```
$cd greet
```

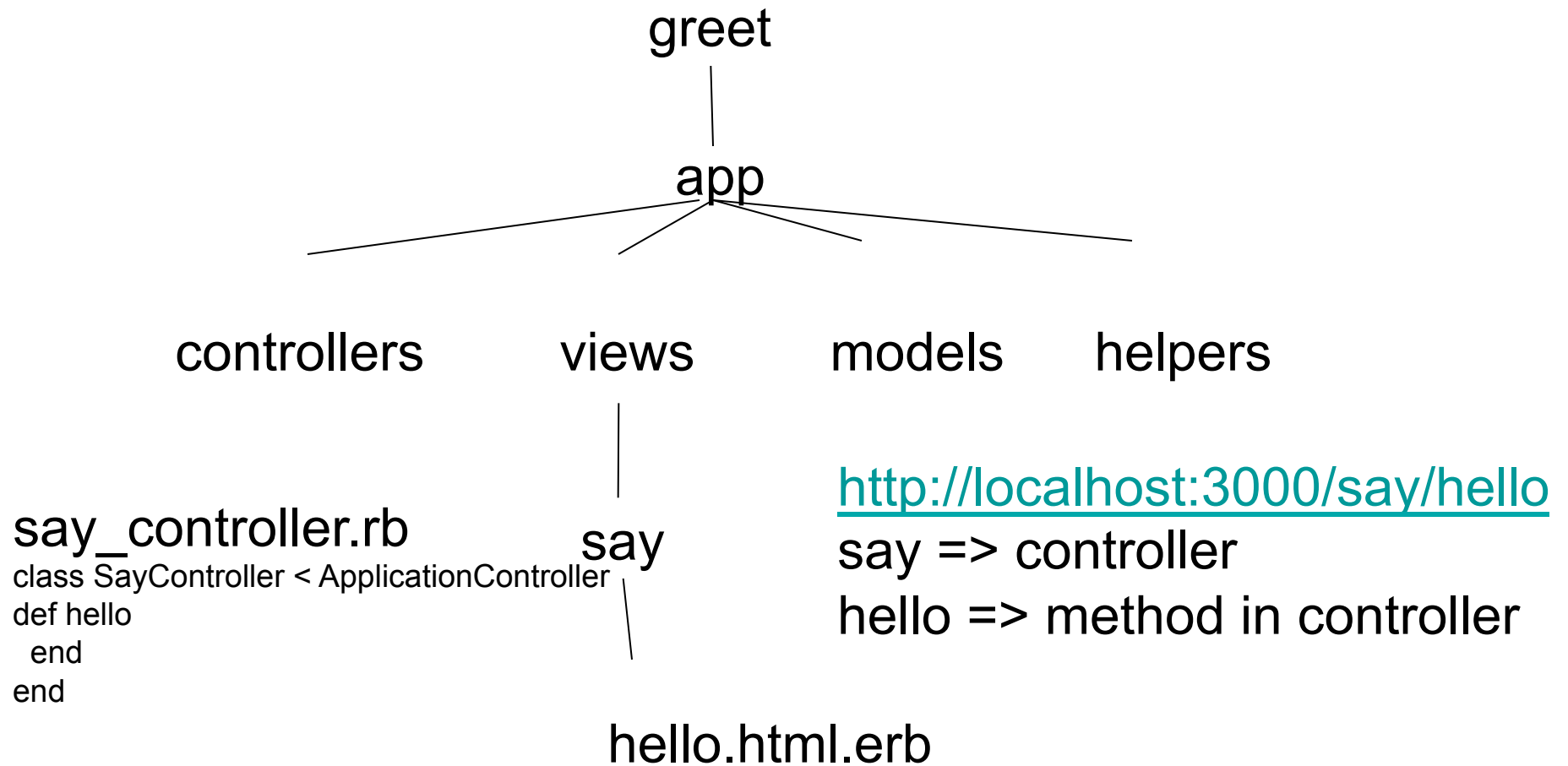
```
$rails generate controller say
```

```
$rails server
```

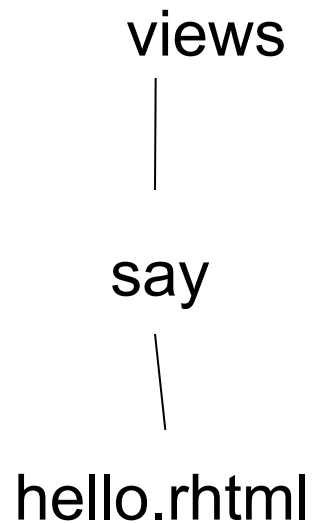
```
Add match '/say/hello' => 'say#hello' to the end  
of greet/config/routes.rb
```

```
Add an HTML file named hello.html.erb to  
greet/app/views/say
```

Rails Directories



hello.html.erb

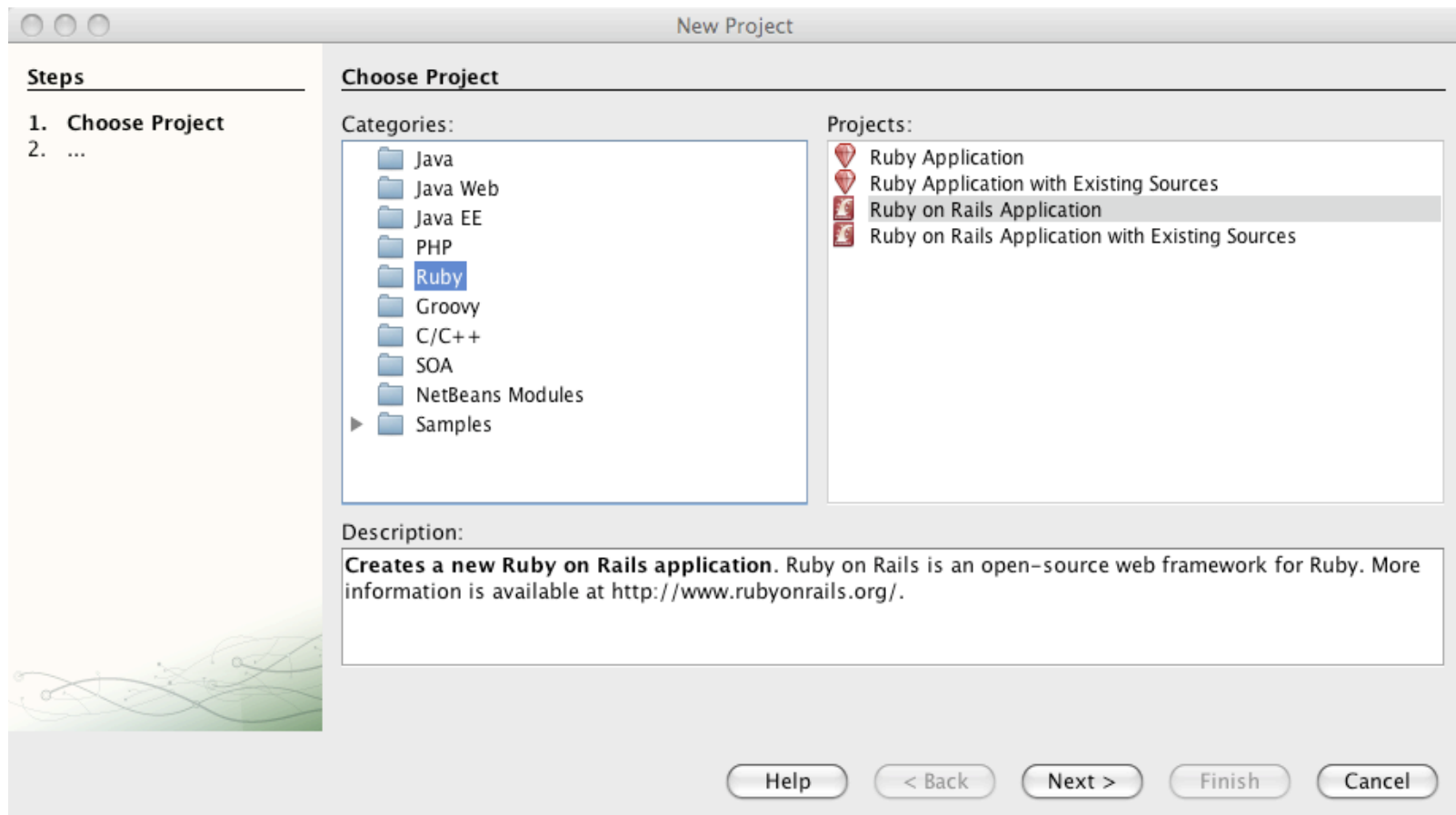


```
<html>
  <!-- all instance variables of the
        controller are visible here. - - >
  <body>
    <b>Ruby says "Yo Mike".</b>
    <%a = 32%>Ruby is <%=a%> degrees cool.
  </body>
</html>
```

Three Examples From Sebesta

- Hello world application
- Processing Forms
- Rails and Databases

Using Netbeans



Create an RoR Project

Steps

1. Choose Project
2. **Name and Location**
3. Database Configuration
4. Install Rails

Name and Location

Project Name:

Project Location:

Project Folder:

Ruby Platform:

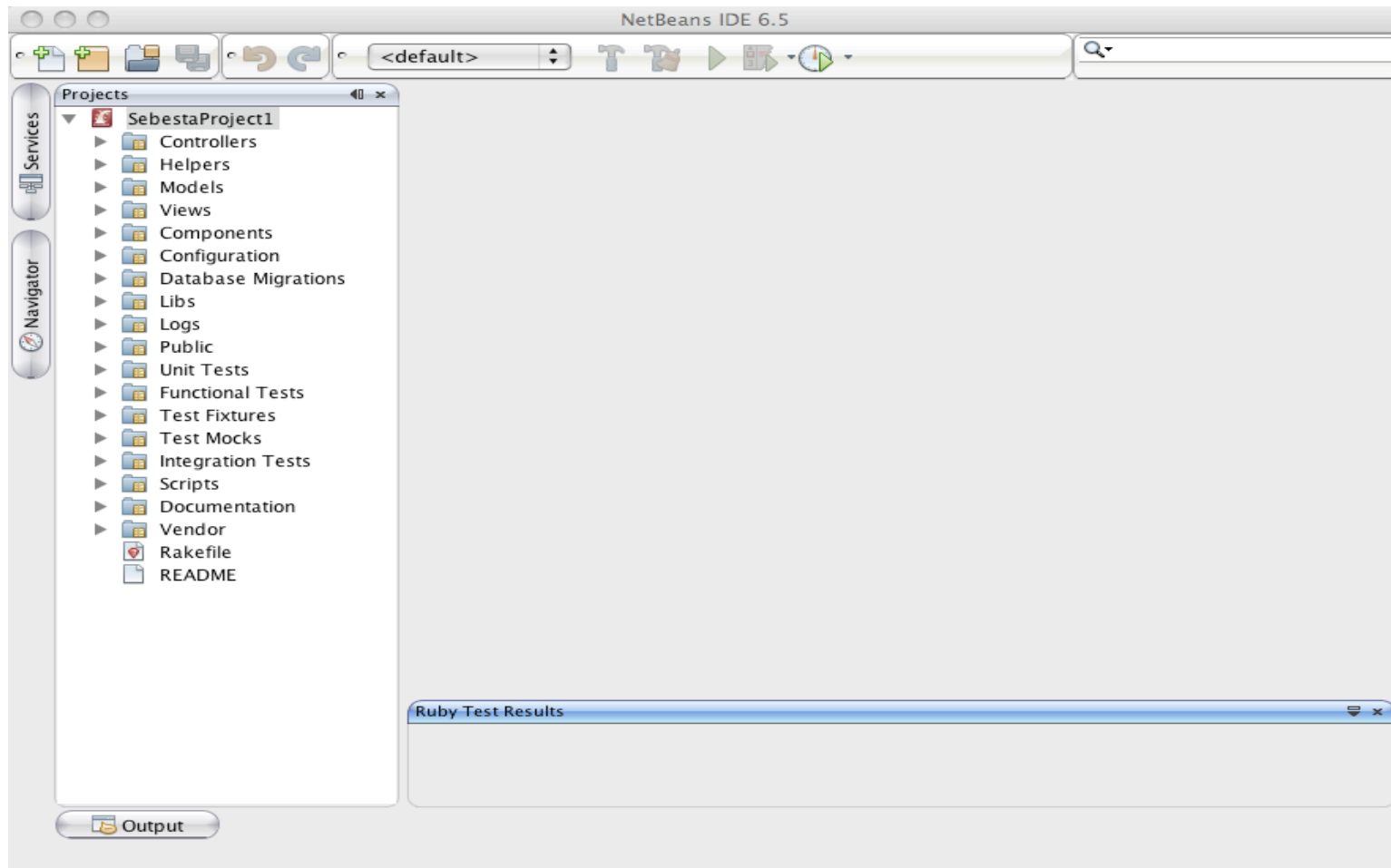
Server:

Add Rake Targets to Support App Server Deployment (.war)

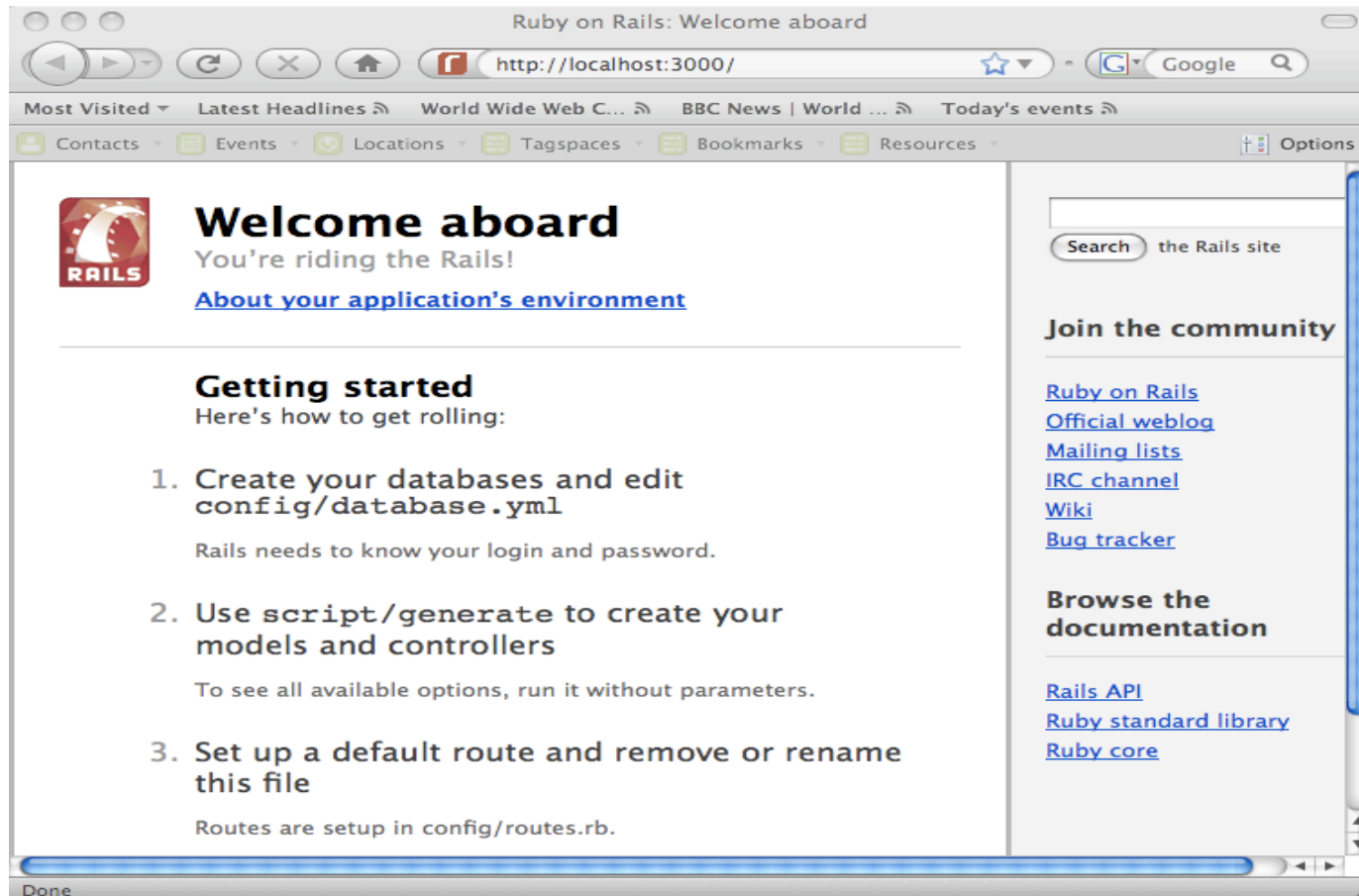
Select MySQL

The screenshot shows a window titled "New Ruby on Rails Application" with a sidebar on the left and a main configuration area on the right. The sidebar, under the heading "Steps", lists four steps: 1. Choose Project, 2. Name and Location, 3. Database Configuration (which is highlighted in bold), and 4. Install Rails. The main area is titled "Database Configuration" and contains "Database Access Information". There are two radio button options: "Configure Using IDE Connections" (which is unselected) and "Specify Database Information Directly" (which is selected). Under the selected option, there are four fields: "Database Adapter" (a dropdown menu with "mysql" selected), "Database Name" (a text box containing "SebestaProject1_development"), "User Name" (an empty text box), and "Password" (an empty text box). Above these fields, there are three rows for "Development:", "Test:", and "Production:", each with a dropdown menu and a "Create DB..." button. At the bottom of the window, there are five buttons: "Help", "< Back", "Next >" (which is highlighted in blue), "Finish", and "Cancel".

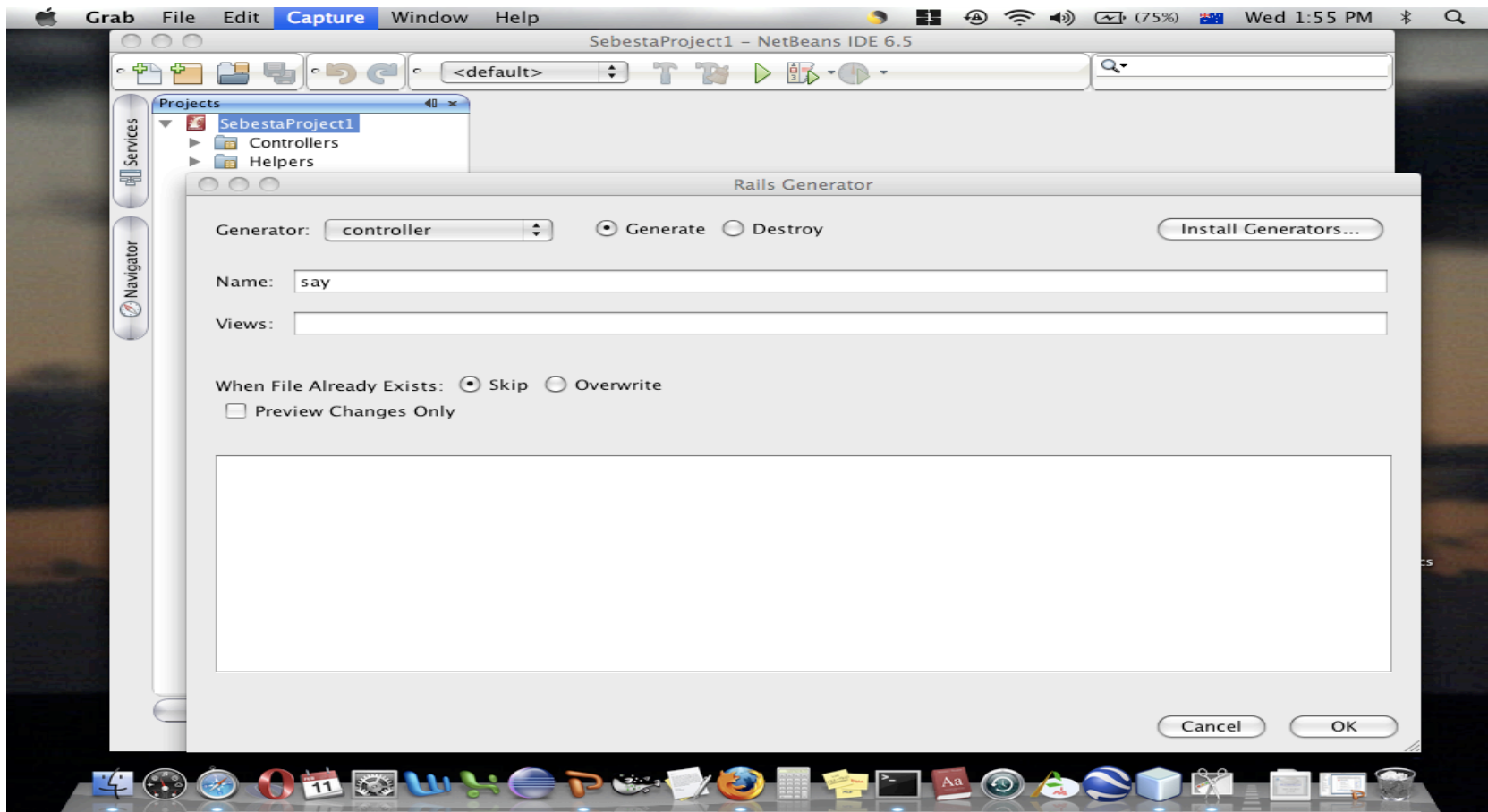
Models Views and Controllers



Run And Visit Rails



Generate A Controller



Modify The Default Controller

```
# The program say_controller.rb is the specific controller  
# for the SebestaProject1 project.  
# Add the definition of the hello method.
```

```
class SayController < ApplicationController  
  def hello  
  end  
end
```

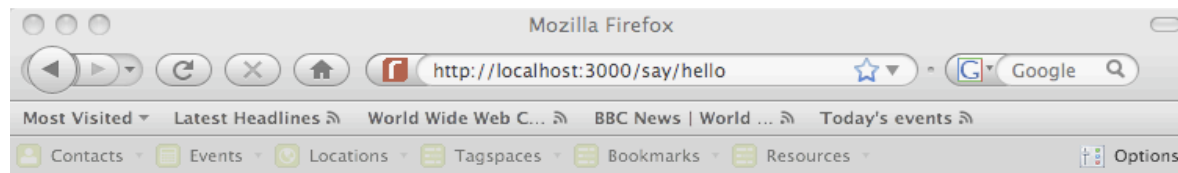
“hello” becomes part of the URL and tells the controller about the view.

Enter The View

1. Select SebestaProject1/Views/Say
2. Right Click
3. New RHTML file
4. File name hello.rhtml

```
<html>
  <!-- all instance variables of the controller are visible here. - - >
  <body>
    <b>Ruby says "Yo Mike".</b>
    <%a = 32%>Ruby is <%=a%> degrees cool.
  </body>
</html>
```

Run And Visit The Application



Ruby says "Yo Mike". Ruby is 32 degrees cool!

So far, no model.

Done

Processing Forms

Steps

1. Choose Project
2. **Name and Location**
3. Database Configuration
4. Install Rails

Name and Location

Project Name:

Project Location:

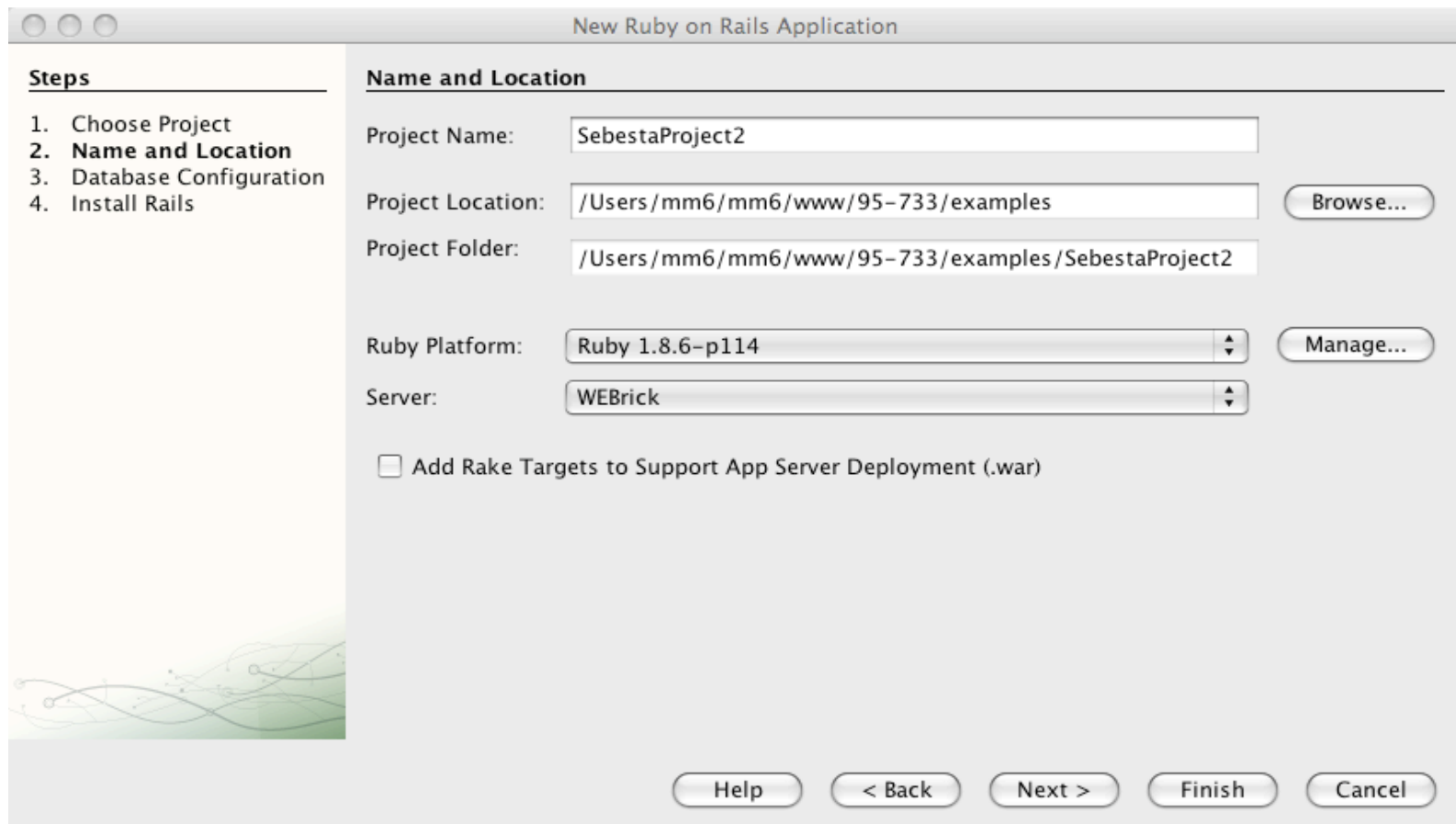
Project Folder:

Ruby Platform:

Server:

Add Rake Targets to Support App Server Deployment (.war)

Select Database



The screenshot shows a window titled "New Ruby on Rails Application" with a sidebar on the left and a main configuration area on the right. The sidebar, under the heading "Steps", lists four steps: 1. Choose Project, 2. Name and Location (which is highlighted), 3. Database Configuration, and 4. Install Rails. The main area is titled "Name and Location" and contains several input fields and buttons. The "Project Name" field contains "SebestaProject2". The "Project Location" field contains "/Users/mm6/mm6/www/95-733/examples" and has a "Browse..." button to its right. The "Project Folder" field contains "/Users/mm6/mm6/www/95-733/examples/SebestaProject2". The "Ruby Platform" field is a dropdown menu showing "Ruby 1.8.6-p114" and has a "Manage..." button to its right. The "Server" field is a dropdown menu showing "WEBrick". At the bottom of the main area, there is a checkbox labeled "Add Rake Targets to Support App Server Deployment (.war)" which is currently unchecked. At the very bottom of the window, there are five buttons: "Help", "< Back", "Next >", "Finish", and "Cancel".

Steps

1. Choose Project
2. **Name and Location**
3. Database Configuration
4. Install Rails

Name and Location

Project Name:

Project Location:

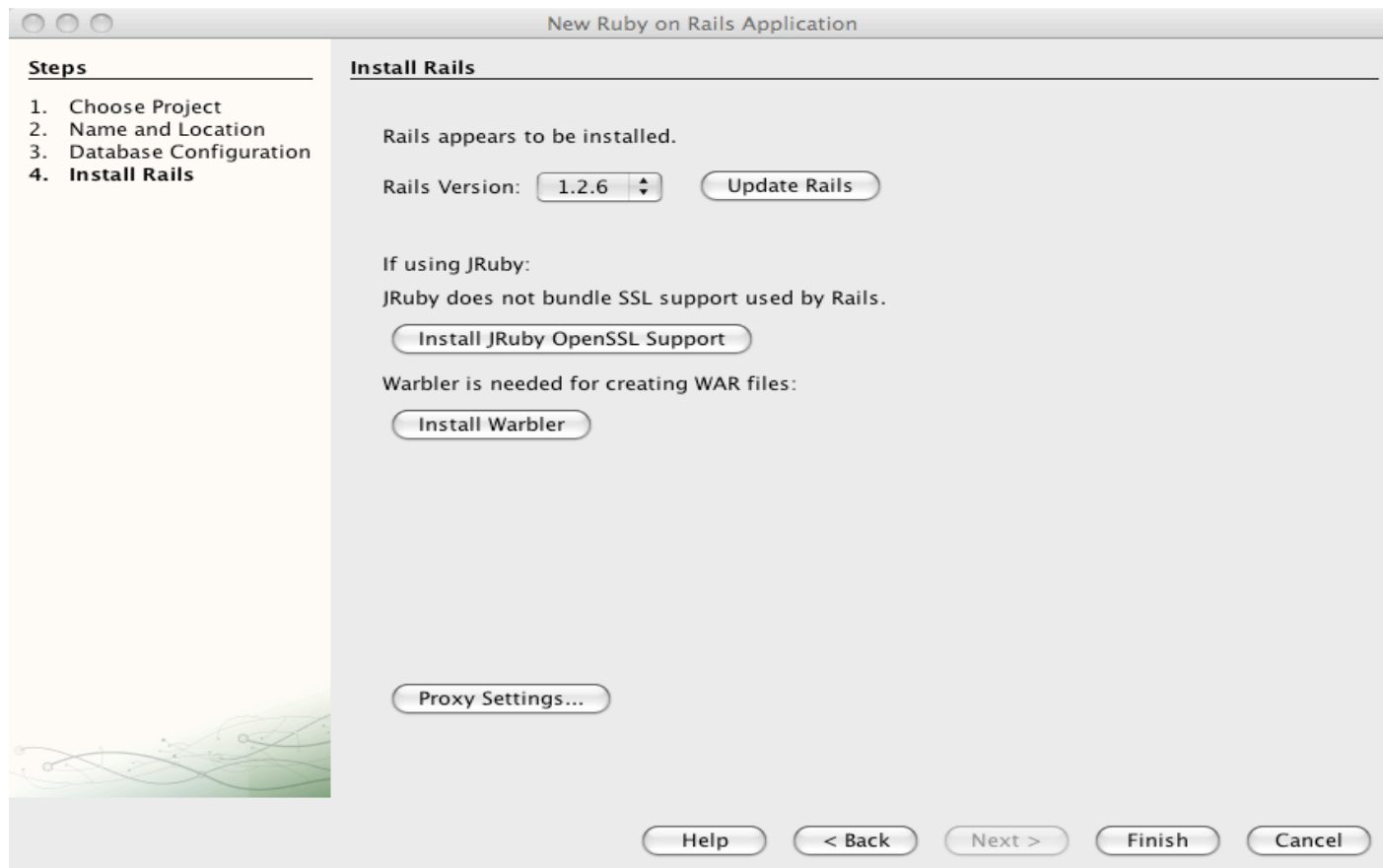
Project Folder:

Ruby Platform:

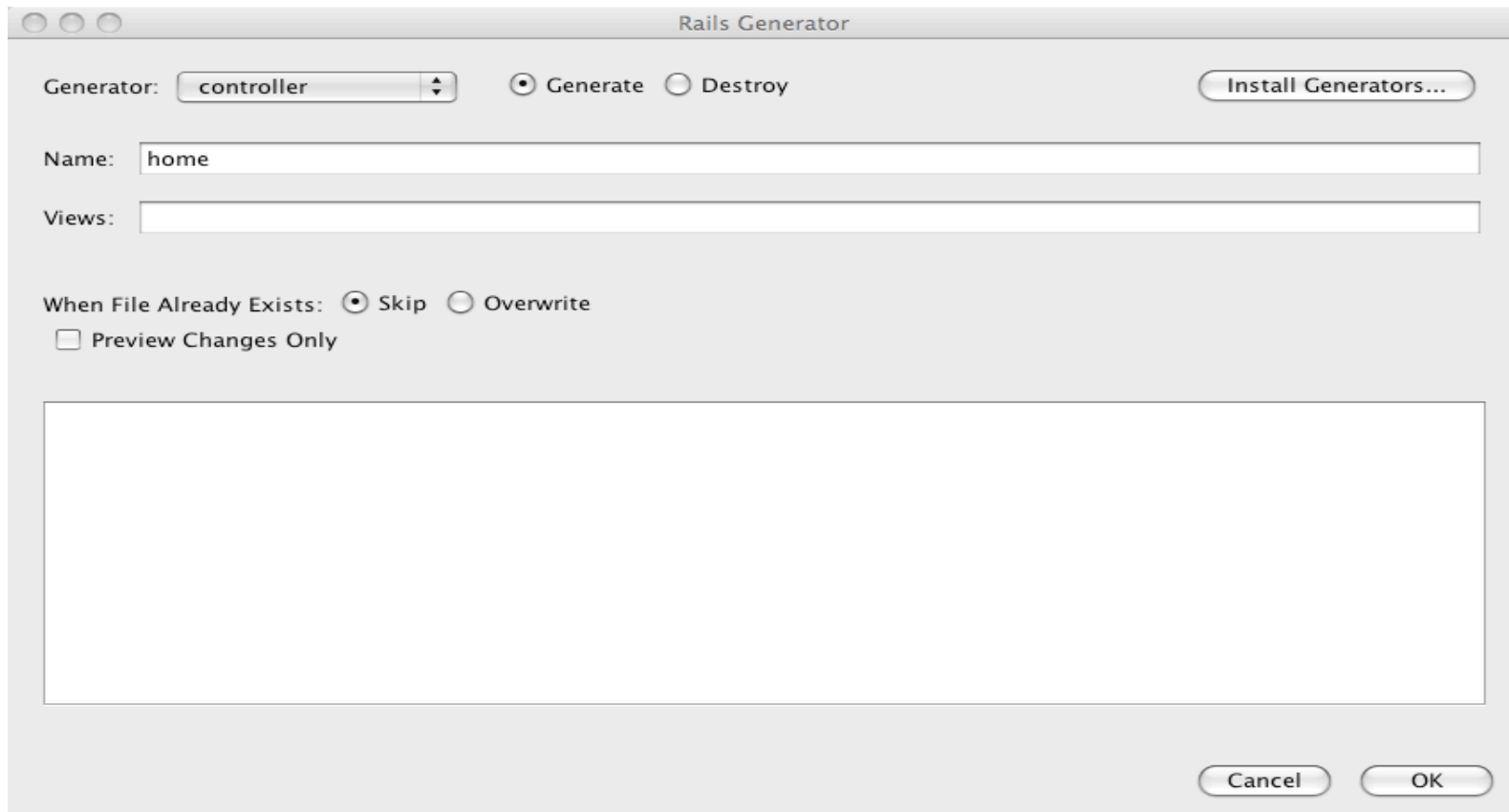
Server:

Add Rake Targets to Support App Server Deployment (.war)

Select Rails Version



From Project Generate Controller



The image shows a screenshot of the 'Rails Generator' dialog box. The window title is 'Rails Generator'. It features a 'Generator:' dropdown menu set to 'controller', with radio buttons for 'Generate' (selected) and 'Destroy'. An 'Install Generators...' button is on the right. Below, there are text input fields for 'Name:' (containing 'home') and 'Views:'. A section for 'When File Already Exists:' includes radio buttons for 'Skip' (selected) and 'Overwrite', and a checkbox for 'Preview Changes Only'. A large empty text area is at the bottom. 'Cancel' and 'OK' buttons are at the bottom right.

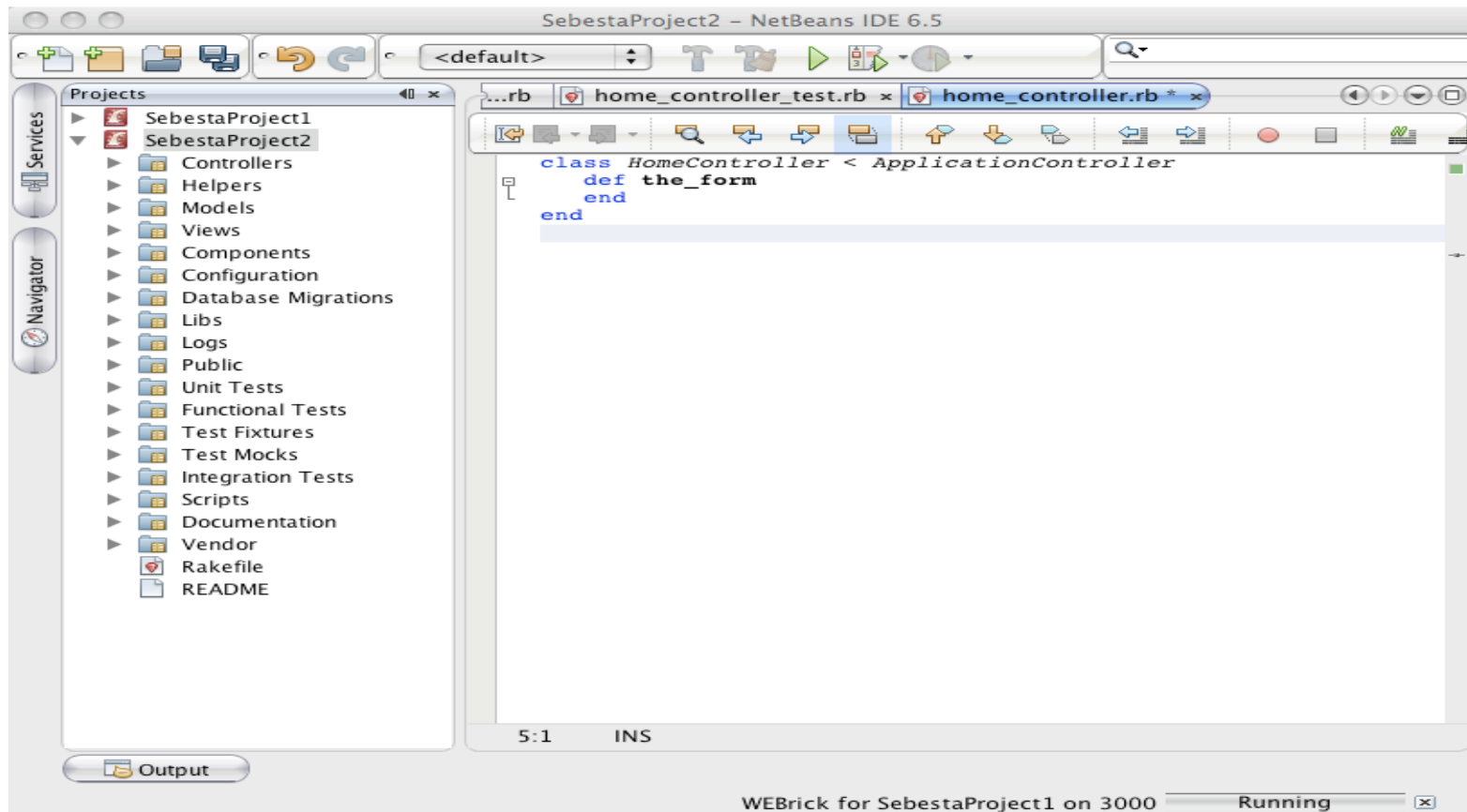
Generator: Generate Destroy

Name:

Views:

When File Already Exists: Skip Overwrite
 Preview Changes Only

The First Action Method is the_form



Add A View

Select Views/Home Right Click for a New RHTML document.

Use the RHTML code from chapter 2 but with a different opening form tag. The data will be arriving with names on the form.

```
<!-- Popcorn.html
  Popcorn sales form document
-->

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Popcorn Sales Form</title>
  </head>
  <body>
    <h2>Welcome to popcorn sales</h2>
    <form action = "result" method = "post">
      <table>
        <tr>
          <td>Buyer's Name:</td>
          <td>
            <input type="text" name="name" size ="30" />
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

Add A View

Select Views/Home Right Click for a New RHTML document.

Use the RHTML code from chapter 2 but with a different opening form tag. The data will be arriving with names on the form. This form (the `_form.rhtml`) is submitted to **result**.

```
<!-- Popcorn.html
  Popcorn sales form document
-->

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Popcorn Sales Form</title>
  </head>
  <body>
    <h2>Welcome to popcorn sales</h2>
    <form action = "result" method = "post">
      <table>
        <tr>
          <td>Buyer's Name:</td>
          <td>
            <input type="text" name="name" size ="30" />
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

We need to handle this in the controller.

Data is named.

Modify The Controller(1)

```
class HomeController < ApplicationController
  def the_form
  end
  def result
    @unpop = params[:unpop].to_i
    @caramel = params[:caramel].to_i
    @caramelnut = params[:caramelnut].to_i
    @toffeynut = params[:toffeynut].to_i
    @name = params[:name]
    @street = params[:street]
    @city = params[:city]
    puts @city
    @payment = params[:payment]
```

After these computations pass the results to the view (result.rhtml).

Modify The Controller(2)

```
@unpop_cost = 3.0 * @unpop
@caramel_cost = 3.5 * @caramel
@caramelnut_cost = 4.5 * @caramelnut
@toffeynut_cost = 5.0 * @toffeynut
@total_price = @unpop_cost + @caramel_cost + @caramelnut_cost +
  @toffeynut_cost
@total_items = @unpop + @caramel + @caramelnut + @toffeynut

@total_price = sprintf("%5.2f",@total_price)
@unpop_cost = sprintf("%5.2f",@unpop_cost)
@caramel_cost = sprintf("%5.2f",@caramel_cost)
@caramelnut_cost = sprintf("%5.2f",@caramelnut_cost)
@toffeynut_cost = sprintf("%5.2f",@toffeynut_cost)
```

```
end
end
```

result.rhtml (1)

```
<?xml version="1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>result.rhtml</head>
  <body>
    <h4>Customer:</h4>
    <%= @name %> <br/>
    <%= @street %> <br/>
    <%= @city %>
  <p/><p/>
```

Values are read from the controller.

result.rhtml (2)

```
<table border="border">
  <caption>Order Information</caption>
  <tr>
    <th>Product</th>
    <th>Unit Price</th>
    <th>Quantity</th>
    <th>Item Cost</th>
  </tr>
  <tr align="center">
    <td>Unpopped Corn</td>
    <td>$3.00</td>
    <td><%= @unpop %> </td>
    <td><%= @unpop_cost %> </td>
```

See Sebesta for rest of code...

Initial Visit on the_form

Popcorn Sales Form

http://localhost:3000/home/the_form

Most Visited | Latest Headlines | World Wide Web C... | BBC News | World ... | Today's events

Contacts | Events | Locations | Tagspaces | Bookmarks | Resources | Options

Welcome to popcorn sales

Buyer's Name:

Street Address:

City, State, Zip:

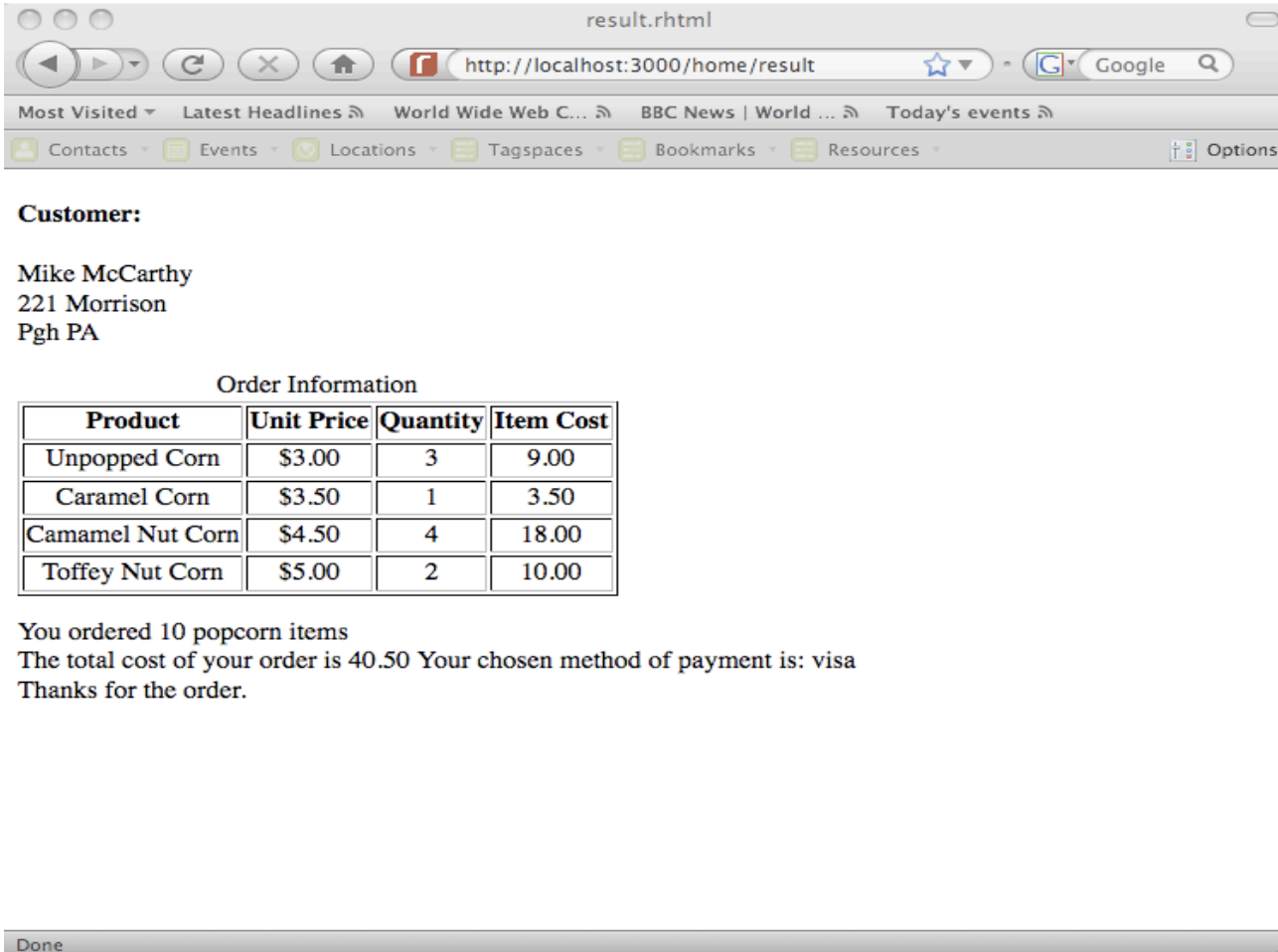
Product Name	Price	Quantity
Unpopped Corn 1 LB	\$3.00	<input type="text" value="3"/>
Caramel Corn 2 LB	\$3.50	<input type="text" value="1"/>
Caramel Nut Corn 2 LB	\$4.50	<input type="text" value="4"/>
Toffey Nut Corn 2 LB	\$5.00	<input type="text" value="2"/>

Payment Method

Visa
 Master Card
 Discover
 Check

Done

Form Submitted to result



The screenshot shows a web browser window titled "result.rhtml" with the address bar displaying "http://localhost:3000/home/result". The browser interface includes navigation buttons, a search bar with "Google", and a bookmarks bar. The main content area displays the following information:

Customer:

Mike McCarthy
221 Morrison
Pgh PA

Order Information

Product	Unit Price	Quantity	Item Cost
Unpopped Corn	\$3.00	3	9.00
Caramel Corn	\$3.50	1	3.50
Camamel Nut Corn	\$4.50	4	18.00
Toffey Nut Corn	\$5.00	2	10.00

You ordered 10 popcorn items
The total cost of your order is 40.50 Your chosen method of payment is: visa
Thanks for the order.

Done

The Model (1)

- Rails uses **Active Record** for object-relational mapping.
- Database rows are mapped to objects with methods.
- In Java's Hibernate, you work from Java's object model.
- Hibernate is a **mapping framework**.
- Active Record is a **wrapping framework**.
- In Active Record, you work from an SQL schema.
- Active Record exploits metaprogramming and convention over configuration.

The Model (2)

- This example is from Bruce Tate at IBM.
- See <http://www.ibm.com/developerworks/java/library/j-cb03076/index.html>.

The Model (3)

Beginning from a database schema:

```
CREATE TABLE people ( id int(11) NOT NULL auto_increment,  
                      first_name varchar(255),  
                      last_name varchar(255),  
                      email varchar(255),  
                      PRIMARY KEY (id) );
```

Create a Ruby class:

```
class Person < ActiveRecord::Base  
  
end
```

The Model (4)

This type of programming is now possible:

```
person = Person.new ;  
person.first_name = "Bruce" ;  
person.last_name = "Tate";  
person.email = bruce.tate@nospam.j2life.com;  
person.save ;  
person = Person.new;  
person.first_name = "Tom";  
person.save
```

The Base class adds attributes to your person class for every column in the database. This is adding code to your code – metaprogramming.

Convention Over Configuration

Model class names such as `Person` are in CamelCase and are English singulars.

Database table names such as `people` use underscores between words and are English plurals.

Primary keys uniquely identify rows in relational databases. Active Record uses `id` for primary keys.

Foreign keys join database tables. Active Record uses foreign keys such as `person_id` with an English singular and an `_id` suffix.

Model Based Validation

```
class Person < ActiveRecord::Base  
  validates_presence_of :email  
end
```

Relationships(1)

```
CREATE TABLE addresses ( id int(11) NOT NULL auto_increment,  
    person_id int(11),  
    address varchar(255),  
    city varchar(255),  
    state varchar(255),  
    zip int(9),  
    PRIMARY KEY (id) );
```

We are following the conventions, so we write...

Relationships(2)

```
class Person < ActiveRecord::Base
  has_one :address          # add an instance variable
                           # of type address
  validates_presence_of :email
end
```

```
class Address < ActiveRecord::Base
  belongs_to :person
end
```

Note that “belongs_to:person” is a metaprogramming method with a symbol parameter.

Relationships(3)

```
person = Person.new;  
person.email = bruce@tate.com;  
address = Address.new ;  
address.city = "Austin";  
person.address = address;  
person.save;  
person2 = Person.find_by_email "bruce@tate.com";  
puts person2.address.city;
```

Output "Austin" ;

Relationships(4)

Other relationships are possible:

```
class Person < ActiveRecord::Base
  has_many :addresses          # must be plural
  validates_presence_of :email
End
```

`has_many` adds an array of addresses to Person.

Relationships(5)

```
load 'app/models/person.rb' ;  
person = Person.find_by_email bruce@tate.com;  
address = Address.new;  
address.city = "New Braunfels";  
person.addresses << address;  
person.save;  
puts Address.find_all.size
```

Output => 2