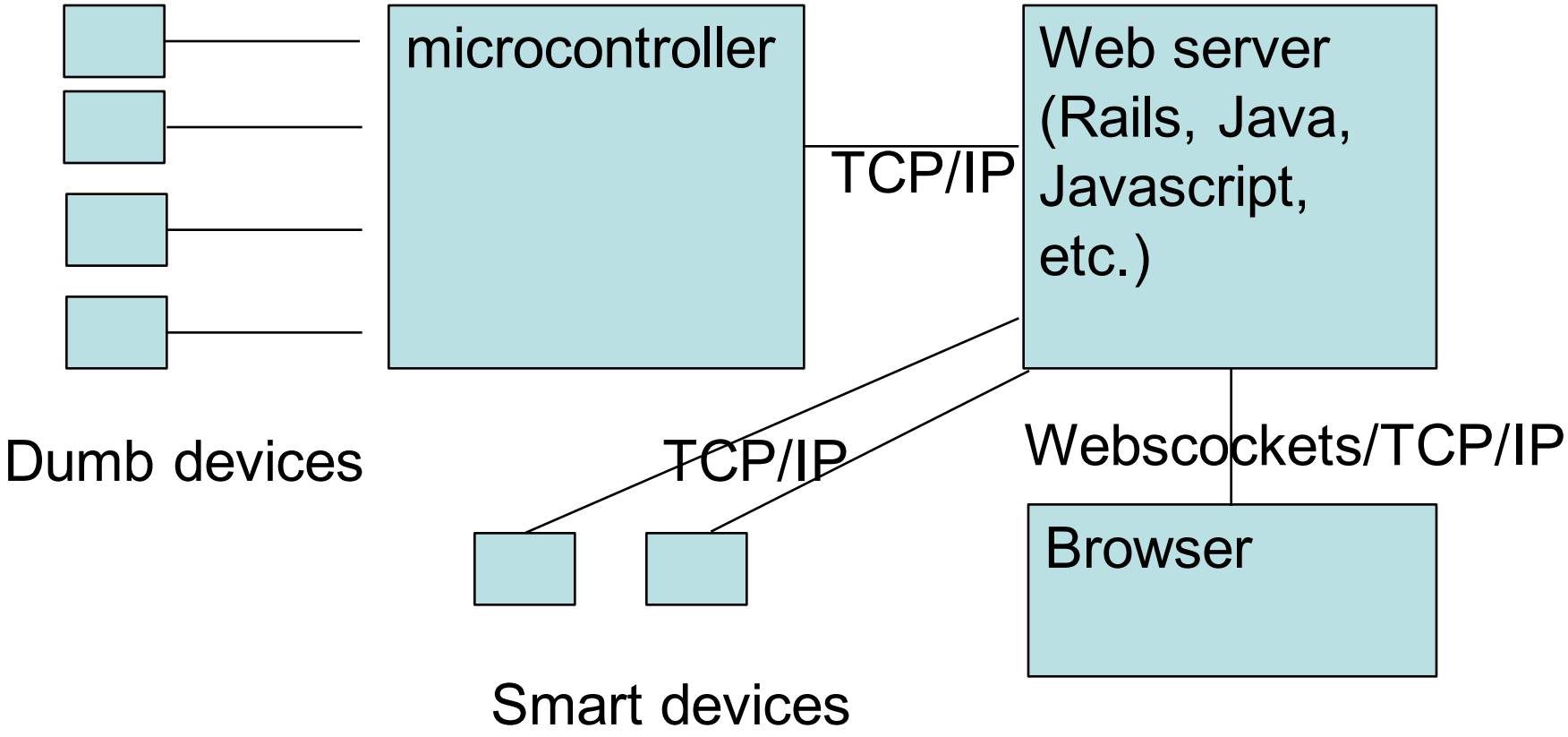
Internet Technologies

IoT, Ruby on Rails and REST

IoT: Jeff Jaffe from W3C

- Consider a person's watch (as an IoT device)
- It will participate in IoT wearable applications (since it is worn).
- It will participate in IoT medical applications (as it takes one's pulse and links into personal medical information).
- It will participate in IoT Smart Homes (used to control the home).
- It will contribute to IoT Smart Cities (as the municipal infrastructure relies on data about weather and traffic).
- It will be used in IoT Smart Factories (to track its usage and condition).
- But to participate across all silos, and for applications to be built which leverage all silos requires common data models, metadata, and an interoperable layered model.

IoT : Where might Rails fit ?



See video at https://www.youtube.com/watch?v=4FtnvyH0qq4

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), GitHub (web-based Git repository) are written in RoR

General Notes on Ruby(1)

- To get started install rbenv or 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.
- The 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

General Notes on Ruby(3)

- Ruby gems: “There is a gem for that”.
- A ruby gem provides functionality.
- May run on its own. A stand alone program. Rails is a gem.
- May be included in your code with require:
- require ‘aws/s3’ # to access Amazon Simple Storage Service
- require is the same as the c language’s include.
- How do you install a gem? From the command line enter:
- gem install GEM_NAME (usually from http://rubygems.org)
- gem install rails
- gem install jquery-rails
- gem install geocoder

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"
```

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 with if

```
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 with unless

```
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 with if else

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

Output

```
======

a == 5
```

Conditions with if/elsif/else

```
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 with case/when

```
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 with case/when/else

```
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)

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

```
Output
======
5
1
To
be
or
not
to
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.

Hashes and JSON (1)

```
# This programs demonstrates how Ruby may be used to parse
# JSON strings.
# Ruby represents the JSON object as a hash.

require 'net/http'
require 'json'

# Simple test example. Set up a string holding a JSON object.

s = '{"Pirates":{"CF" : "McCutchen","P" : "Bernett","RF" : "Clemente"}}'

# Get a hash from the JSON object. Same parse as in Javascript.
parsedData = JSON.parse(s)
```

Hashes and JSON (2)

```
# Display
print parsedData["Pirates"]   # returns a Ruby hash
print "\n"
print parsedData["Pirates"]["P"] + "\n"   #Bernett
print parsedData["Pirates"]["RF"] + "\n"  #Clemente
```

Hashes and JSON (3)

```
# Go out to the internet and collect some JSON from Northwind
require 'net/http'
require 'json'

url = "http://services.odata.org/Northwind/Northwind.svc/Products(2)?$format=json"

# Make an HTTP request and place the result in jsonStr
jsonStr = Net::HTTP.get_response(URI.parse(url))
data = jsonStr.body

jsonHash = JSON.parse(data)

# See if the product is discontinued
if (jsonHash["Discontinued"])
      print jsonHash["ProductName"].to_s + " is a discontinued product"
   else
      print jsonHash["ProductName"].to_s + " is an active product"
end
```

A Digression: Check out OData

Check out https://northwinddatabase.codeplex.com

What will this query do?
http://services.odata.org/Northwind/Northwind.svc/
Products(1)/Order_Details/?$format=json

What would you like to do with this data?
GET, PUT, DELETE, POST

The Northwind database is an Open Data Protocol (Odata) implementation.
Odata is based on REST. What is REST?

Open Data Protocol

- URL's taken seriously
- Service Document exposes collections:

  http://services.odata.org/V3/Northwind/Northwind.svc/
- $metadata describes content (entity data model types)

  http://services.odata.org/V3/Northwind/Northwind.svc/$metadata
- Each collection is like an RDBMS table

  http://services.odata.org/V3/Northwind/Northwind.svc/Customers

Open Data Protocol

- Visit http://services.odata.org/Northwind/Northwind.svc/Products(1)/?$format=json

The OData API is RESTful

- Representational State Transfer (REST)
- Roy Fielding's doctoral dissertation (2000)
- Fielding (along with Tim Berners-Lee) designed HTTP and URI's.
- The question he tried to answer in his thesis was "Why is the web so viral"? What is its architecture? What are its principles?
- REST is an architectural style – guidelines, best practices.

REST Architectural Principles

- The web has addressable resources.
  Each resource has a URI.
- The web has a uniform and constrained interface.
  HTTP, for example, has a small number of methods. Use these to manipulate resources.
- The web is representation oriented – providing diverse formats.
- The web may be used to communicate statelessly – providing scalability
- Hypermedia is used as the engine of application state.

Back to Ruby: 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

Scalers Are Pass By Value

```
#scalers are pass by value

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

looper(3)
```

Output

```
======
0
1
2
```

Parenthesis Are Optional

```
#scalers are pass by value

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

looper 3
```

Output

```
======
0
1
2
```

Passing 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.

Only one code block may be passed.

Use procs or lambdas if you need more.

Passing Code Blocks (2)

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

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

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

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

Passing Code Blocks (3)

```
def interest(balance)
   yield balance
end

rate = 0.15
interestAmt = interest(1000.0) { |bal| bal * rate }
print "interest is #{interestAmt}"

rate = 0.12
total = interest(1000.0) { |bal| bal * (rate + 1.0)}
print "interest is #{total}"
```

What's the output?

interest is 150.0interest is 1120.0

Passing 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
```

Passing 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
```

Passing 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 }
```

Passing Code Blocks(7)

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

Output

```
======
Yo!
Yo!
Yo!
Yo!
```

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
```

Duck Typing

```
class Duck
  def quack
    puts "Quaaaaaack!"
  end

  def feathers
    puts "The duck has white and gray feathers."
  end
end

class Person
  def quack
    puts "The person imitates a duck."
  end
```

From Wikipedia

Duck Typing (2)

```
 def feathers
    puts "The person takes a feather from the ground and shows it."
  end
end

def in_the_forest duck     # takes anything that quacks with feathers
  duck.quack
  duck.feathers
end
```

From Wikipedia

Duck Typing (3)

```
 def game
  donald = Duck.new
  john = Person.new
  in_the_forest donald
  in_the_forest john
end

game
```

From Wikipedia

Reflection

```
class Dog
  def bark
    puts "woof woof"
  end

  def fur
    puts "This dog likes you to pat her fur."
  end
end

scout = Dog.new

if(scout.respond_to?("name"))
   puts "She responds to name"
end
if(scout.respond_to?("bark"))
   puts "She responds to bark"
   puts scout.bark
end
```

She responds to bark
woof woof

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
  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.

Mixins

```
module SomeCoolMethods

def foo
    puts "foo is running"
end

def foo2
    puts "foo2 is running"
end

end

class CoolClass

   include SomeCoolMethods

end
x = CoolClass.new
x.foo2
```

The methods of a module become members of a class. Think “multiple inheritance” in Ruby.

If this were an external module it would be ‘required’ first. Then ‘included’.

‘require’ is like C’s include. ‘include’ is used for mixins.

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.

From: Alan Skorkin's "Closures – A simple explanation

Javascript has Closures Too!

```
function foo(x) {
  return function() { alert("Hi " + x); }
}

var t = foo("Mike");
var m = foo("Sue");

t();
m();
```

Javascript has Closures Too!

```
<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”

A Closure in Ruby

```
def foo (p)
   p.call              #call the proc
end

x = 24
#create a proc to pass
p = Proc.new { puts x }

foo(p)

x = 19
foo(p)
```

Quiz: What's the output?

Note: It is easy to pass two or more procs. Only one code block may be passed.

Note: x is not within the scope of foo.

Note: a reference to x is used. Not a value.

A Closure in Ruby

```
def foo (p)
   p.call              #call the proc
end

x = 24
#create a proc to pass
p = Proc.new { puts x }

foo(p)

x = 19
foo(p)
```

```
24
19
```

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)
p = obj1.display("some values")
caller(p)
p.call()
obj1.set(3)
p.call
```

Lambdas are procs but with arity checking and different return semantics.

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)
p = obj1.display("some values")
caller(p)
p.call()
obj1.set(3)
p.call
```

```
ruby closure.rb
Value1: 5, Value2: some values
Value1: 5, Value2: some values
Value1: 3, Value2: some values
```

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
```

Passing Hashes

```
def foo(a,hash)

  hash.each_pair do |key, val|
        puts "#{key} -> #{val}"
  end
end

foo("Hello",{:cool => "Value", :tooCool => "anotherValue" })

# Or, we may drop the parens…

foo "Hello" ,{:cool => "Value", :tooCool => "anotherValue" }
```

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.

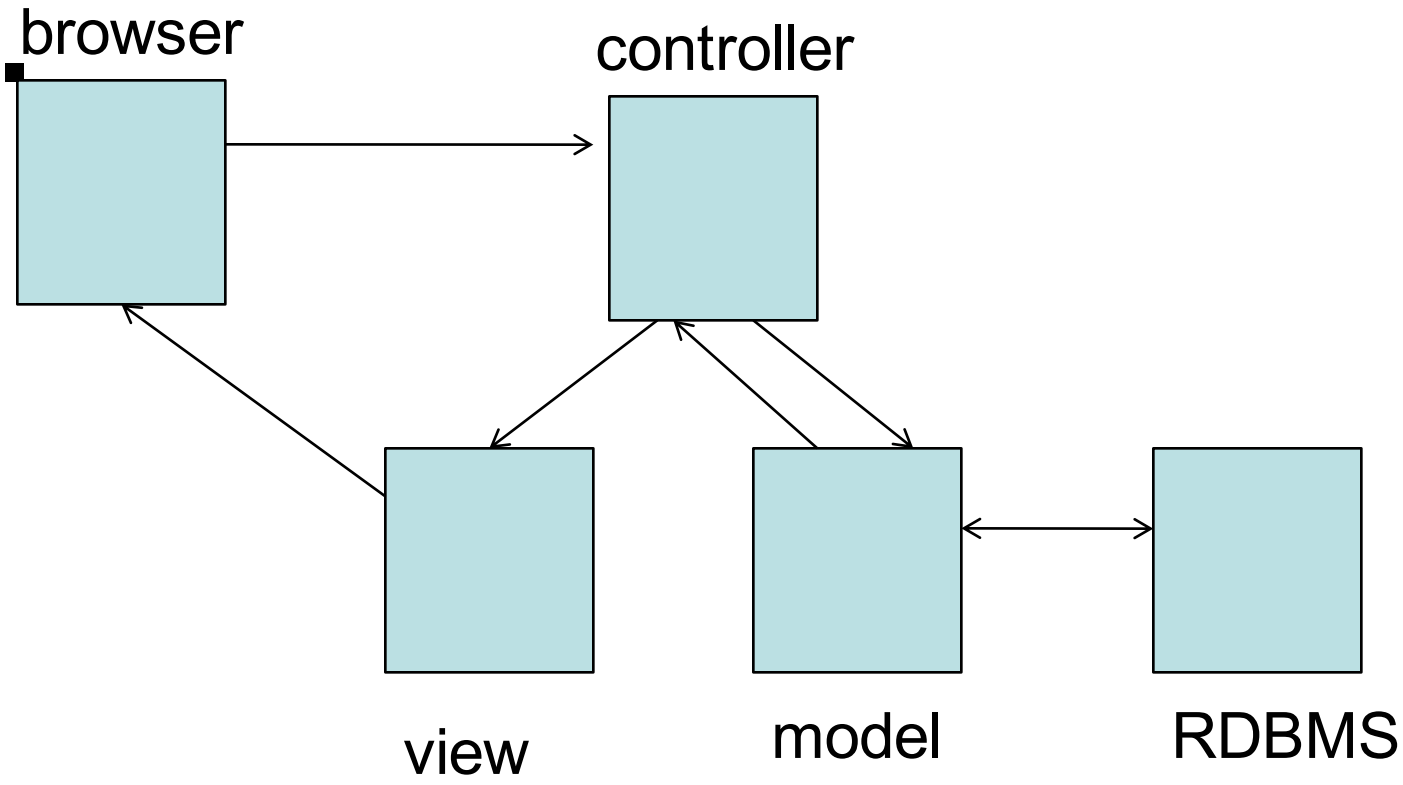
Source: Sebesta

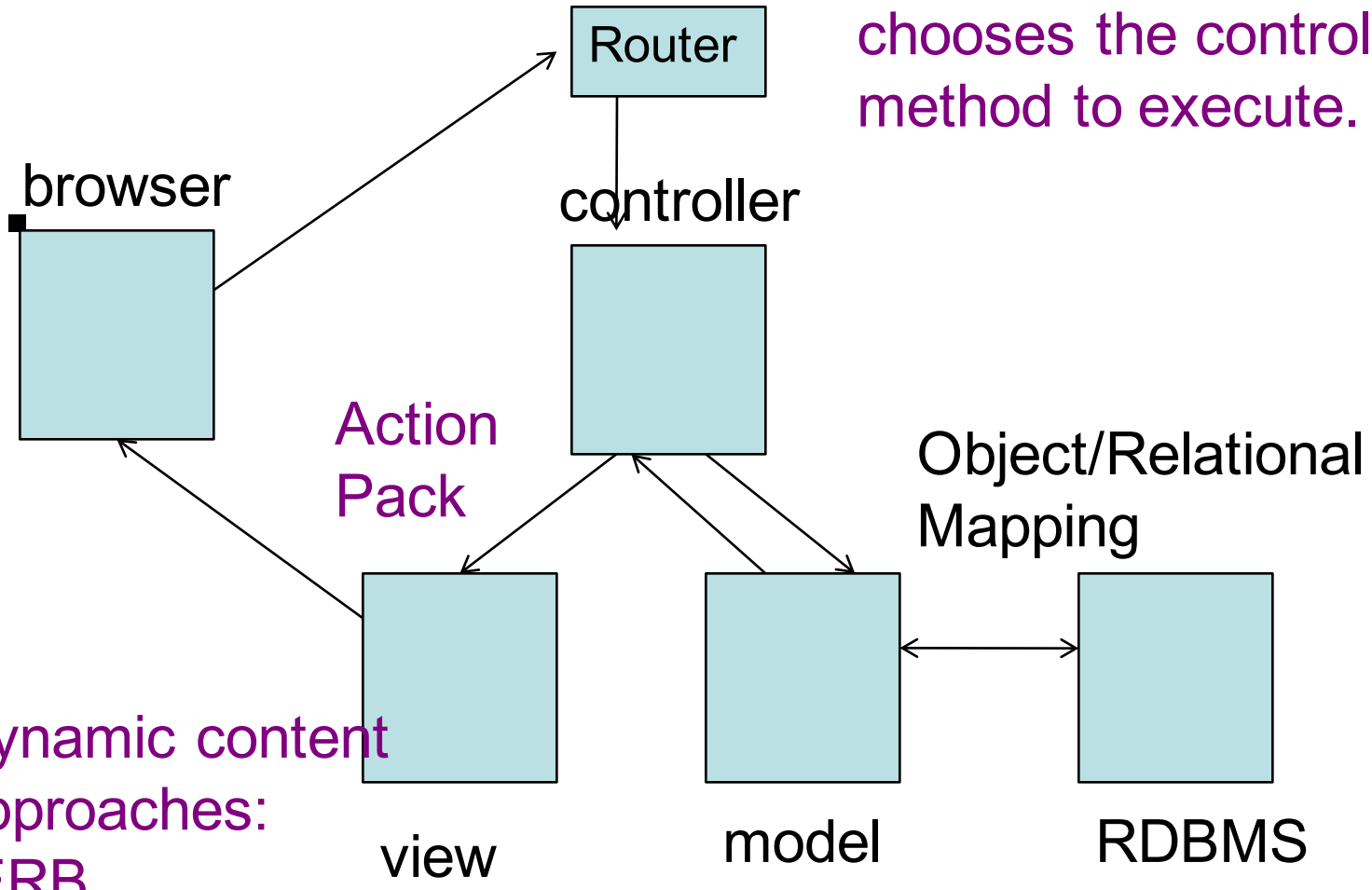
Model View Controller

- 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.

From the Rails README

Model View Controller





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

Dynamic content approaches:
-ERB
-XML Builder
-RJS for Javascript

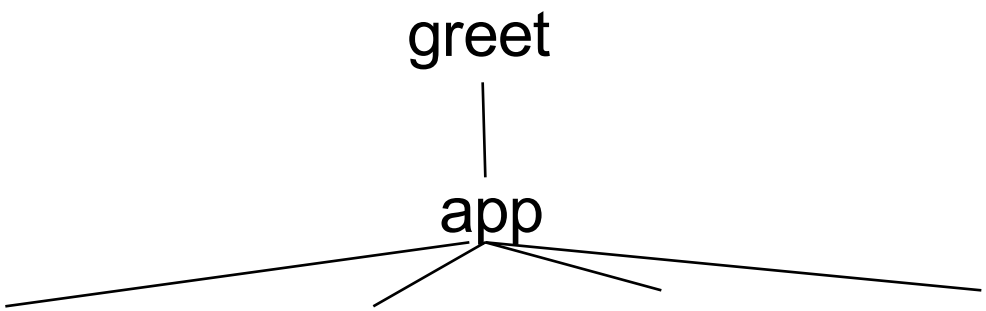
ActiveRecord

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
Add  get '/say/hello', to: 'say#hello' to the end
of greet/config/routes.rb
Or, add get '/say/hello', :to => 'say#hello'
Add an HTML file named hello.html.erb to
greet/app/views/say
$rails server
```

Rails Directories



controllers views models helpers

say_controller.rb

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

say

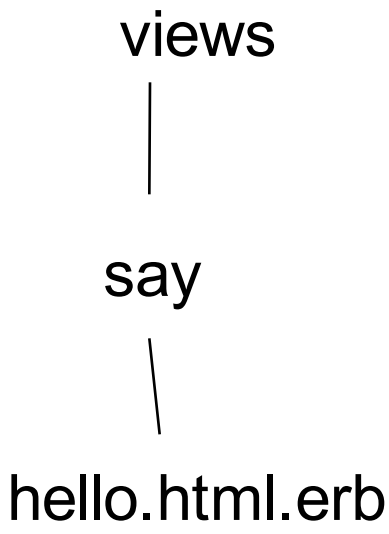
http://localhost:3000/say/hello

say => controller

hello => method in controller

hello.html.erb

hello.html.erb

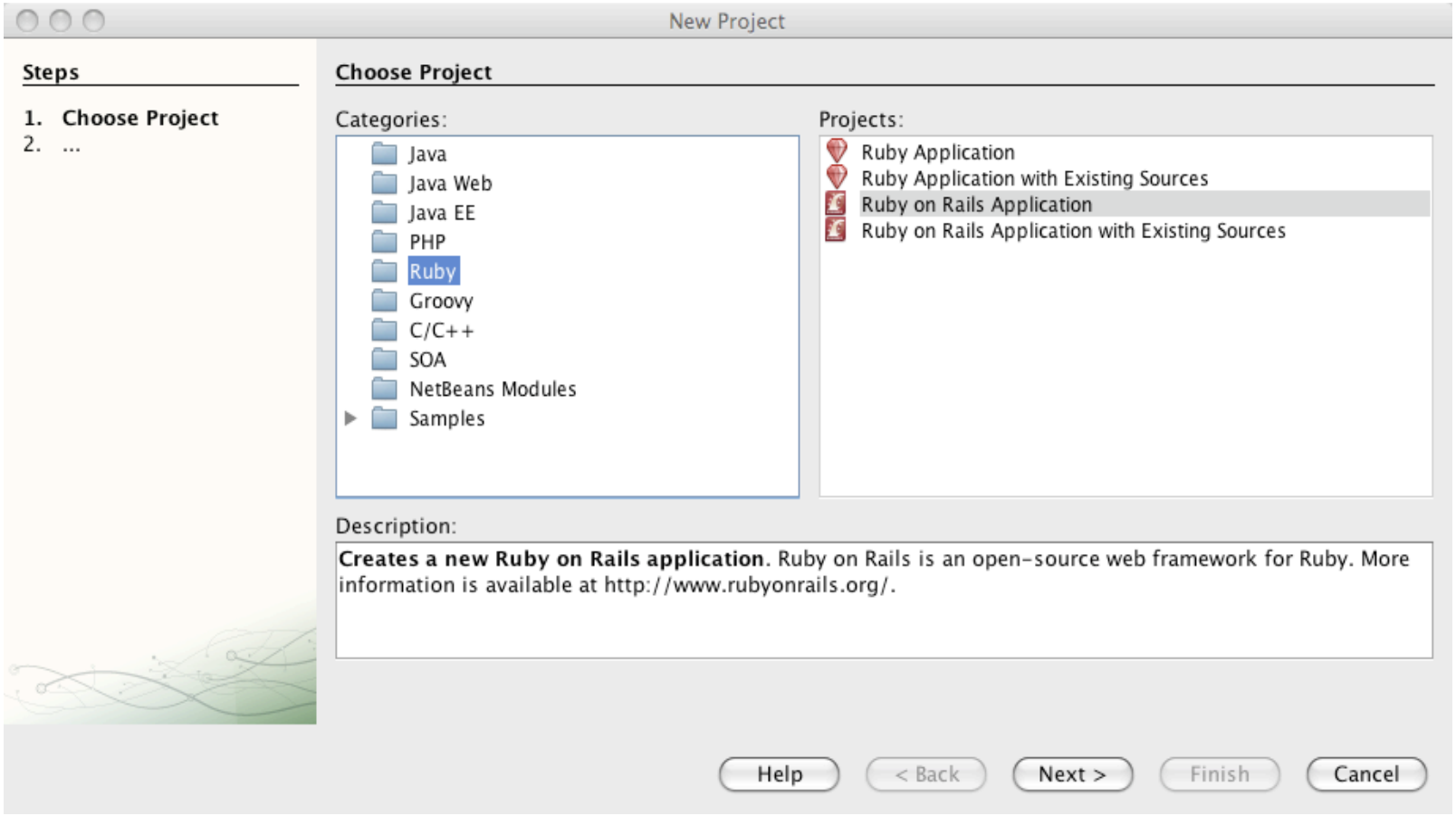



```
<html>
  <!– all instance variables of the
       controller are visible here. - - >
  <body>
    <b>Ruby says "Yo Mike".</b>
    <%a = 32%>Ruby is <%=a%> degrees cool.
  </body>
</html>
```

Two Examples From Sebesta

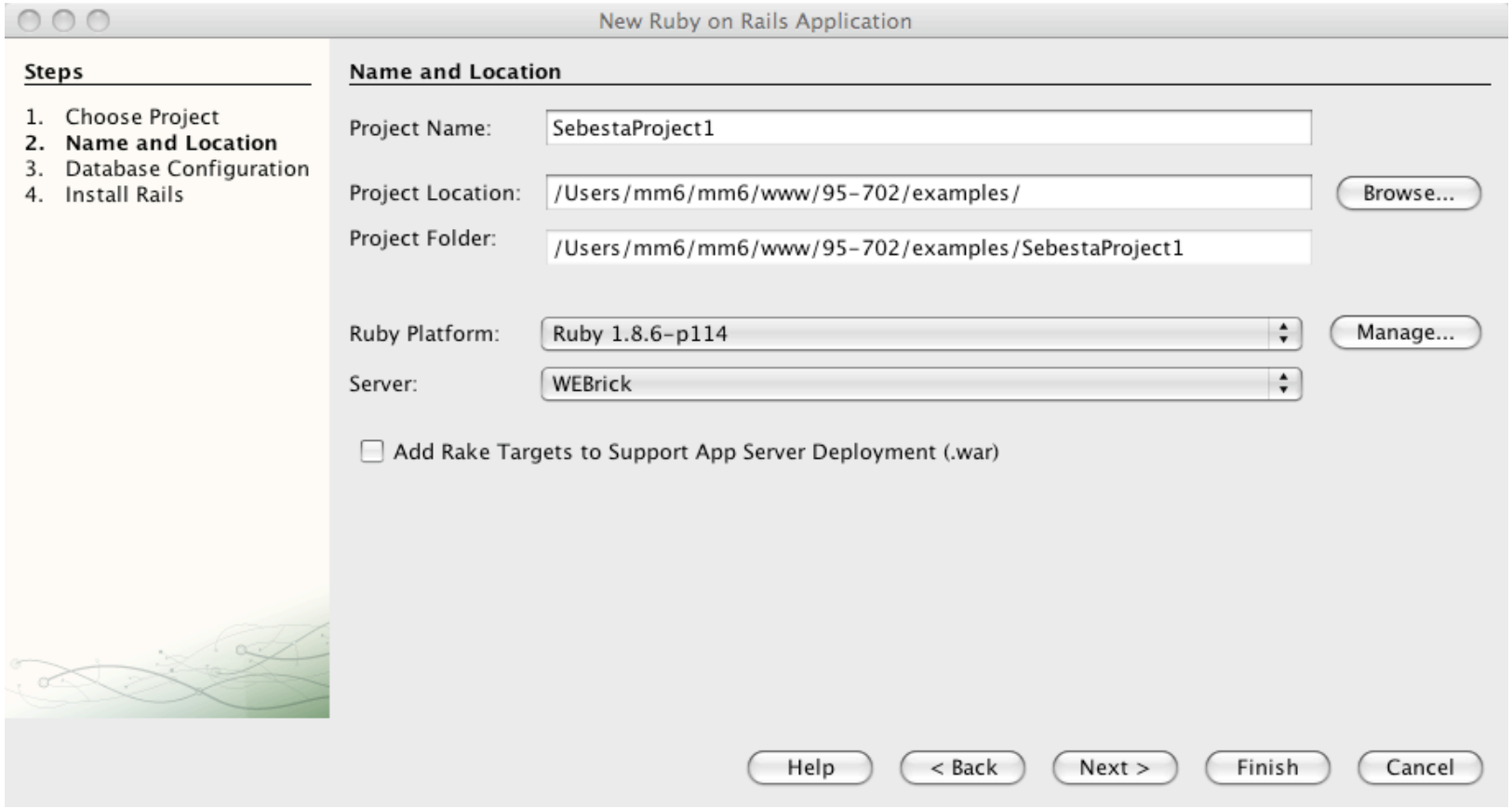
- Hello world application
- Processing a Popcorn Form

Using Netbeans

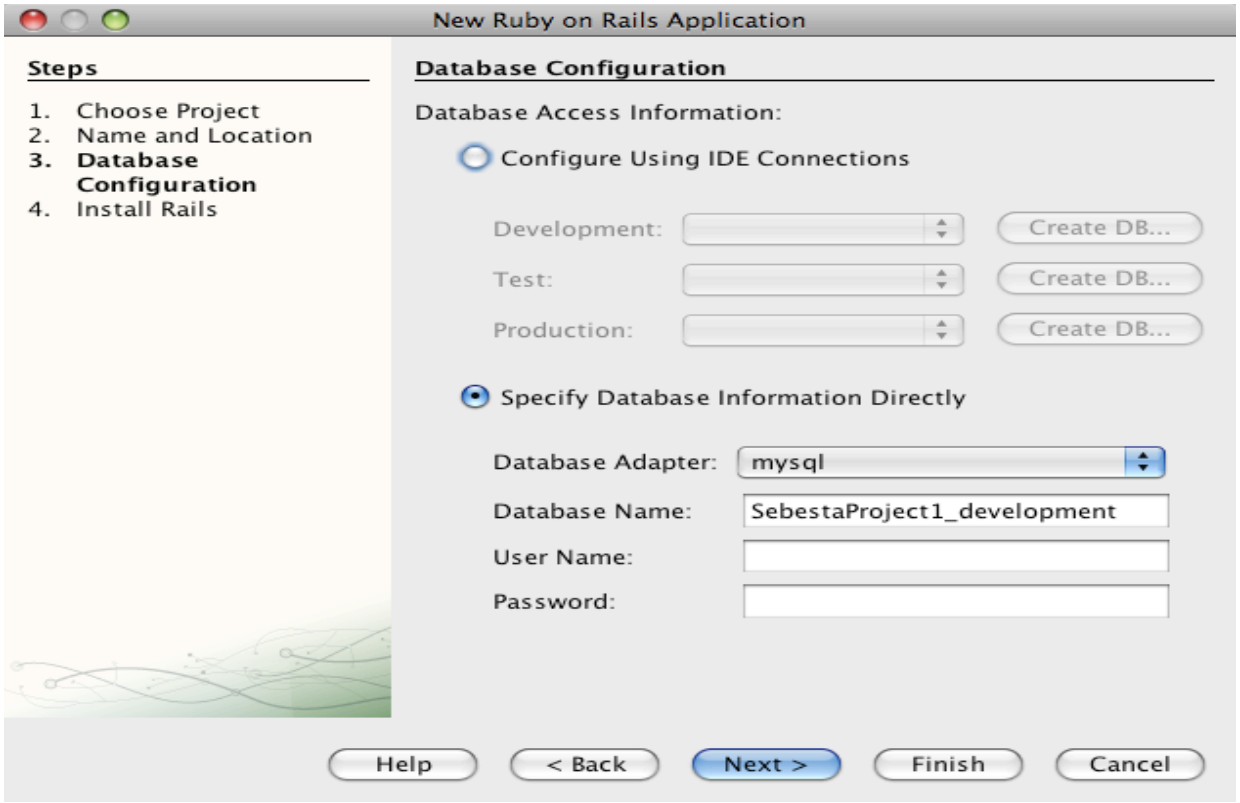


See Tom Enebo's NetBeans Ruby Project

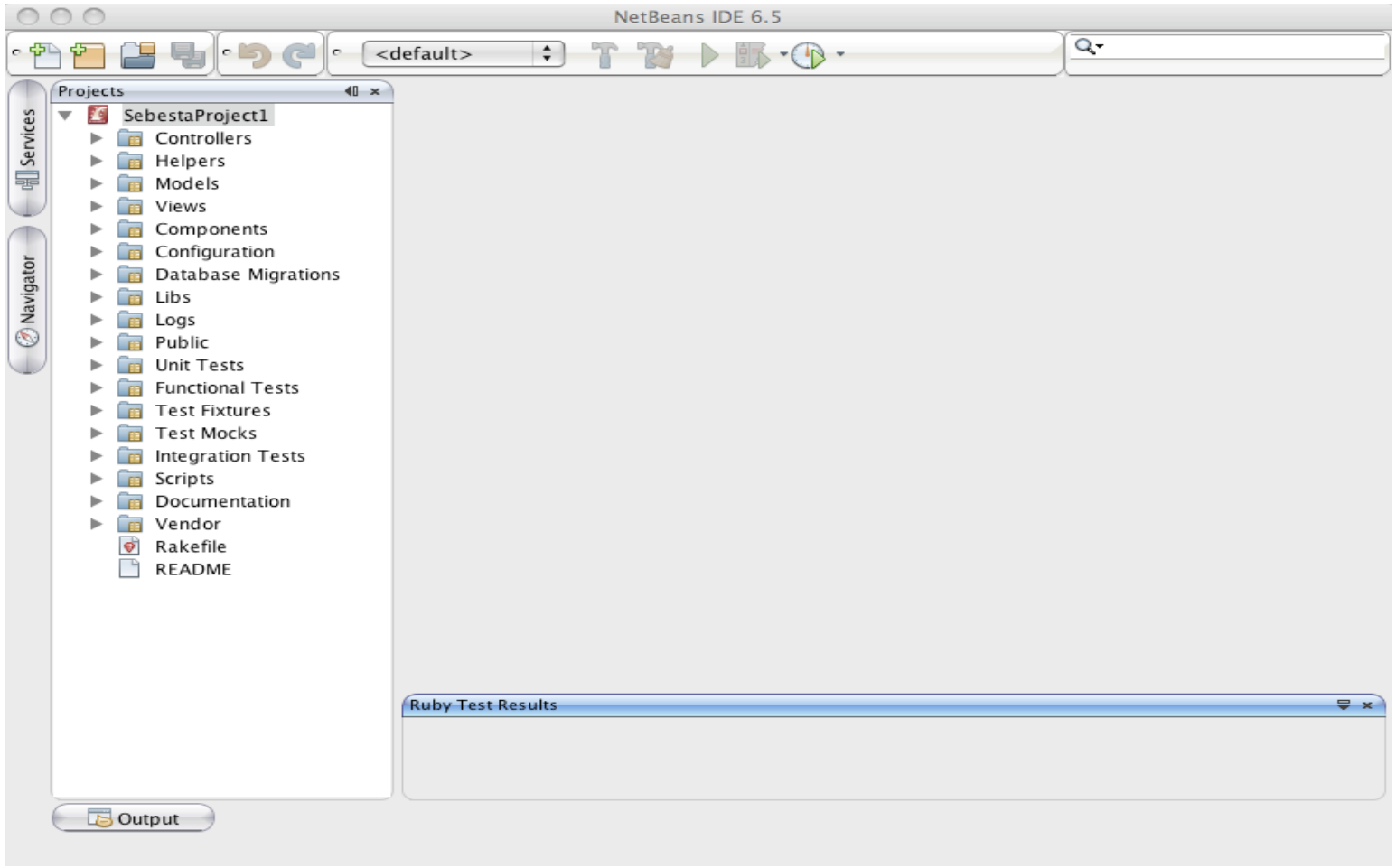
Create an RoR Project



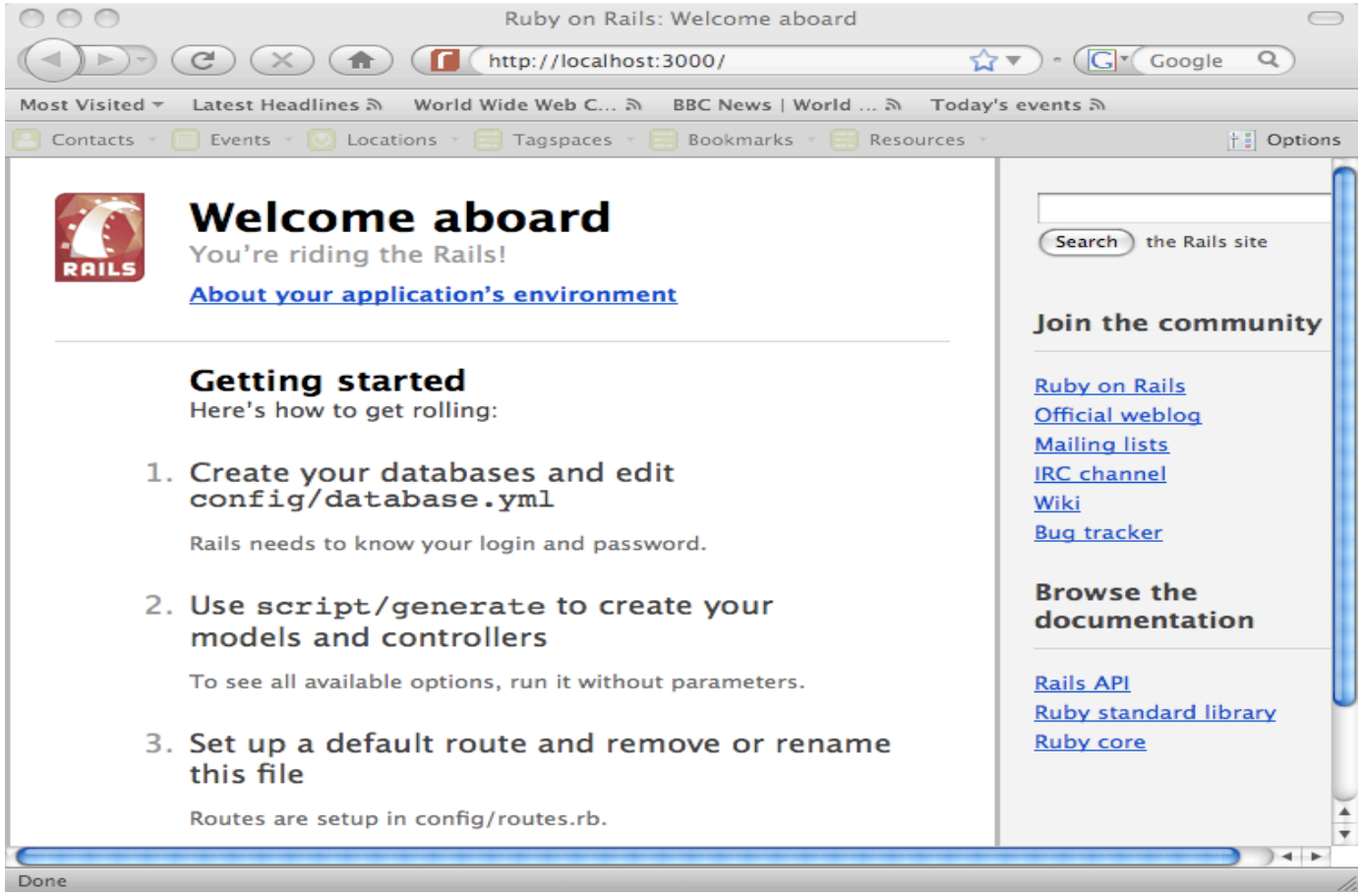
Select MySQL



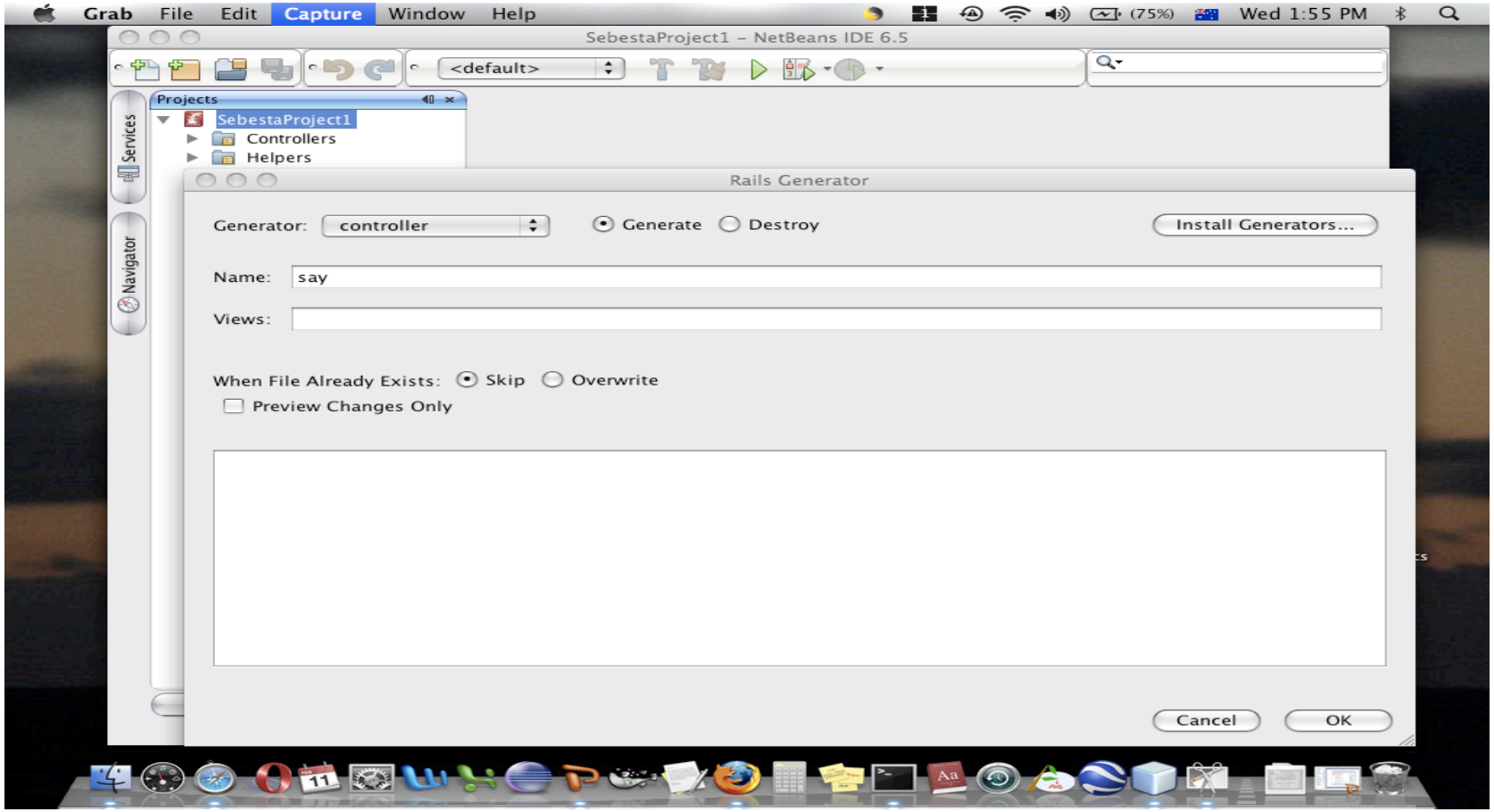
Models Views and Controllers



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
```

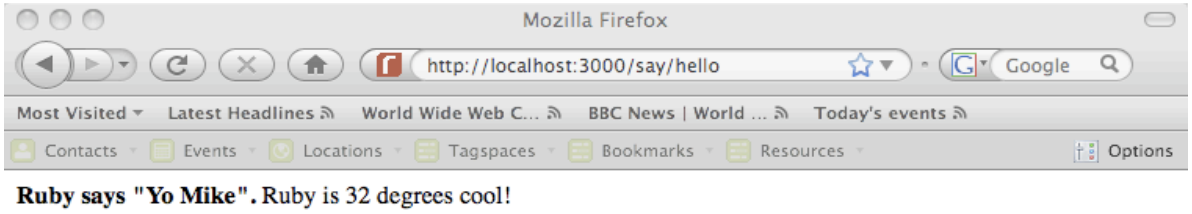
"hello" becomes part of the URL and tells the controller about the view.

Enter The View

1. Select SebestaProject1/Views/Say
2. Right Click
3. New HTML file
4. File name 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>
```

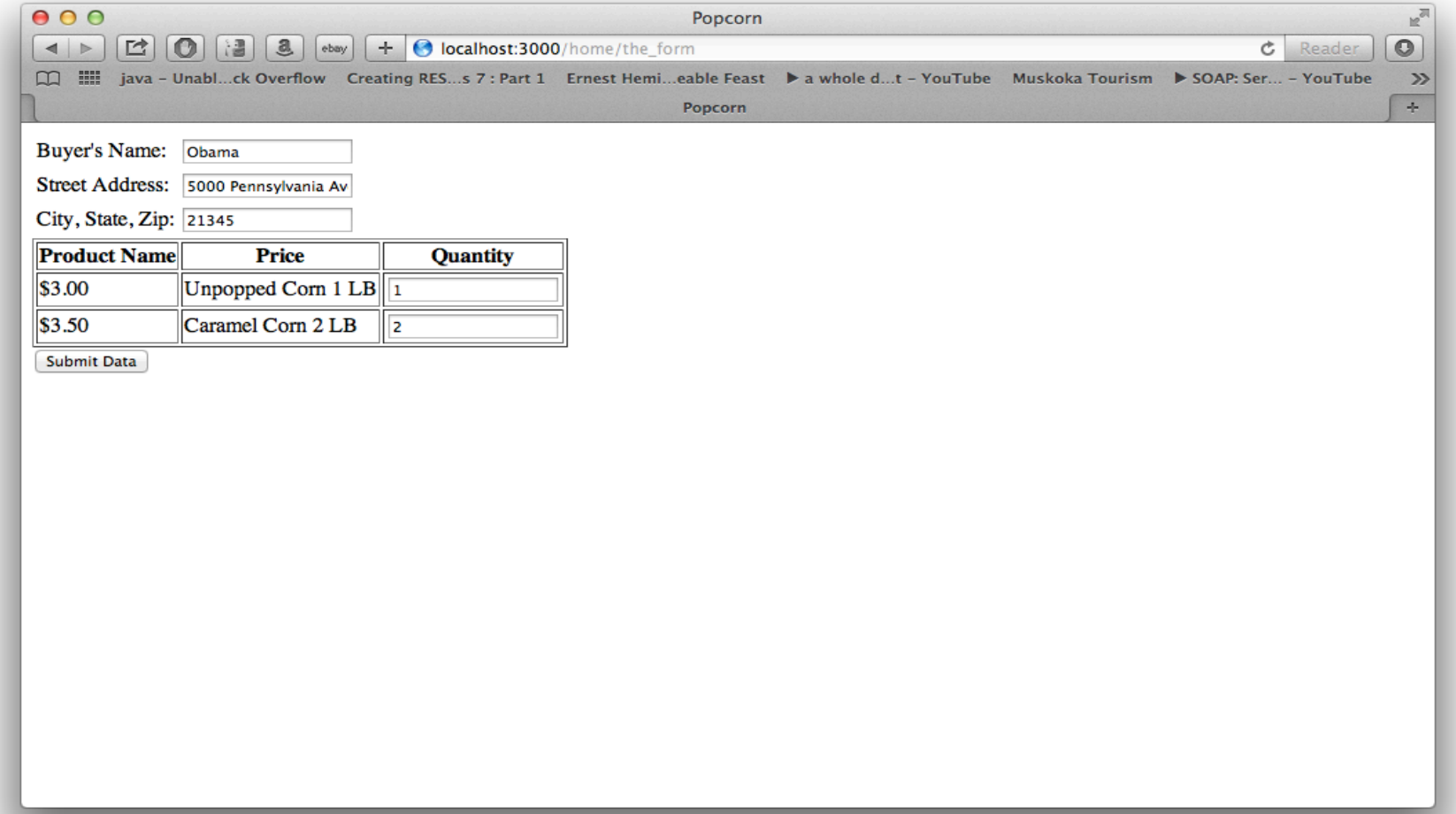
Run And Visit The Application



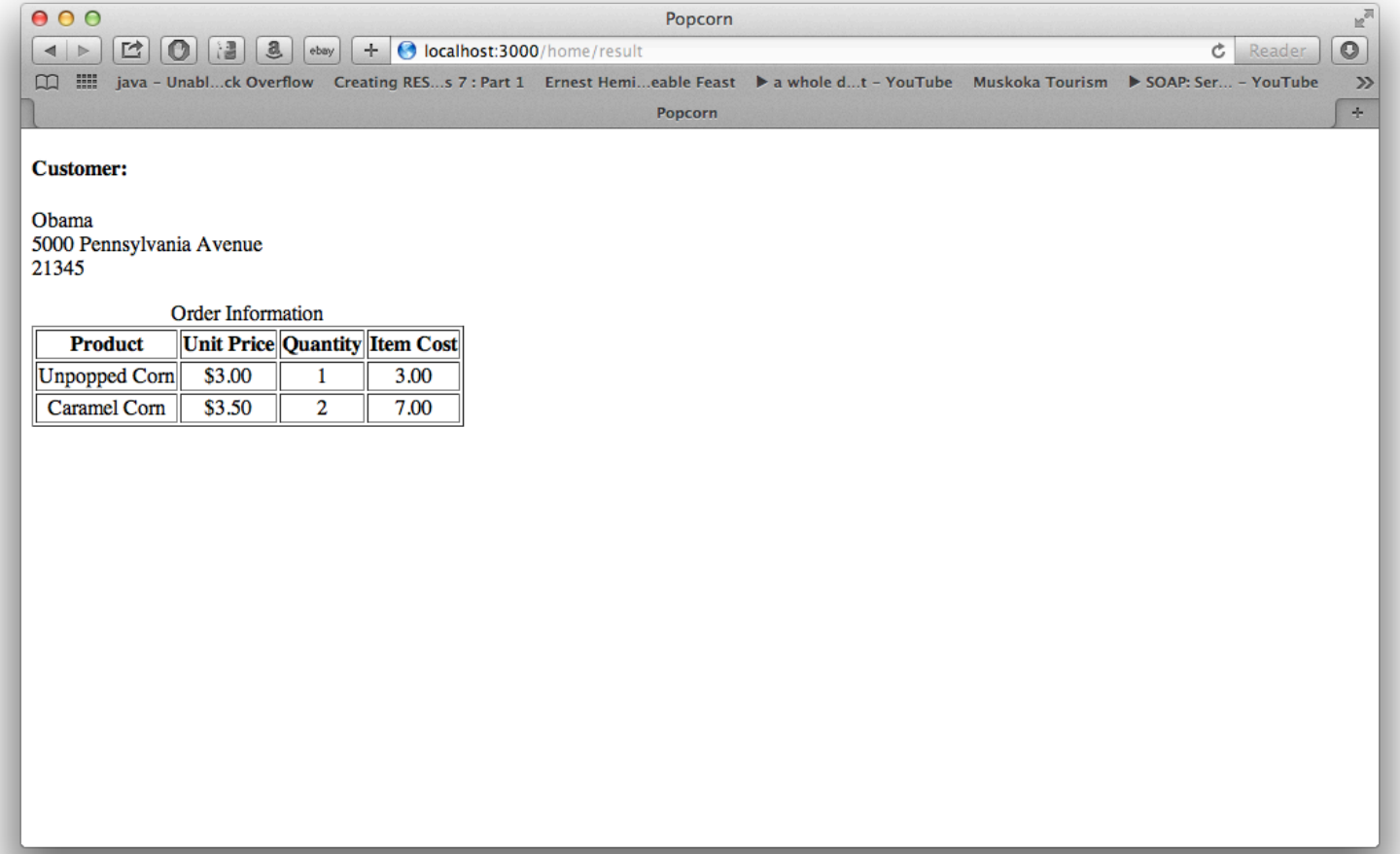
As an exercise, include the helper call
<%= link_to "Cool", :action => "hello" %>
in the html.

So far, no model.

Processing Forms



Result


Popcorn
localhost:3000/home/result
Reader
java – Unabl...ck Overflow
Creating RES...s 7 : Part 1
Ernest Hemi...eable Feast
a whole d...t – YouTube
Muskoka Tourism
SOAP: Ser... – YouTube
Popcorn
Customer:
Obama
5000 Pennsylvania Avenue
21345
Order Information
Product
Unit Price
Quantity
Item Cost
Unpopped Corn
$3.00
1
3.00
Caramel Corn
$3.50
2
7.00

routes.rb

```
get '/home/the_form', to: 'home#the_form'
post '/home/result', to: 'home#result'
```

Quiz: How could these routes be written with :to rather than to: ?

The Home controller(1)

```
class HomeController < ApplicationController

  def the_form
  end
```

The Home controller(2)

```
 def result
    @name = params[:name]
    @street = params[:street]
    @city = params[:city]
    @unpop = params[:unpop].to_i
    @unpop_cost = 3.0 * @unpop
    @caramel = params[:caramel].to_i
    @caramel_cost = @caramel * 3.5
    @unpop_cost = sprintf("%5.2f",@unpop_cost)
    @caramel_cost = sprintf("%5.2f",@caramel_cost)
  end
end
```

The Form View(1)

```
<%= form_tag("/home/result", method: "post") do %>
<table>
  <tr>
    <td><%= label_tag(:name, "Buyer's Name:") %></td>
    <td><%= text_field_tag(:name) %></td>
  </tr>
  <tr>
    <td><%= label_tag(:street, "Street Address:") %></td>
    <td><%= text_field_tag(:street) %></td>
  </tr>
```

The Form View(2)

```
  <tr>
    <td><%= label_tag(:city, "City, State, Zip:") %></td>
    <td><%= text_field_tag(:city) %></td>
  </tr>
</table>
<table border="border">
<tr>
  <th>Product Name</th>
  <th>Price</th>
  <th>Quantity</th>
</tr>
```

The Form View the_form.html.erb(3)

```
    <tr>
      <td>$3.00</td>
      <td><%= label_tag(:unpop, "Unpopped Corn 1 LB") %></td>
      <td><%= text_field_tag(:unpop) %></td>
    </tr>
    <tr>
      <td>$3.50</td>
      <td><%= label_tag(:caramel, "Caramel Corn 2 LB") %></td>
      <td><%= text_field_tag(:caramel) %></td>
    </tr>
    </table>
    <%= submit_tag("Submit Data") %>
<% end %>
```

Results View (result.html.erb) (1)

```
<h4>Customer:</h4>
<%= @name %> <br/>
<%= @street %> <br/>
<%= @city %>
<p/><p/>
```

Results View (result.html.erb) (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>
 </tr>
```

Results View (result.html.erb) (3)

```
<tr align ="center">
    <td>Caramel Corn</td>
    <td>$3.50</td>
    <td><%= @caramel %> </td>
    <td><%= @caramel_cost %> </td>
  </tr>
</table>
```

Routing Using routes.rb (1)

URL's must be mapped to actions in the controller.

Suppose, in routes.rb, we have

```
get '/jobs/:id', to:  'jobs#show'
```

Then, an HTTP

```
GET /jobs/3
```

results in execution of the jobs controller's show action with { :id => 3 } in params. Thus params[:id] is 3.

Routing Using routes.rb (2)

Suppose we have a line in routes.rb that reads:

```
resources :jobs
```

Then, we have created seven different routes to various actions in the jobs controller.

GET /jobs maps to the index action
GET /jobs/:id maps to the show action
GET /jobs/new maps to the new action
GET /jobs/:id/edit maps to the edit action
POST /jobs maps to the create action
PUT and DELETE are mapped as well…

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.
- 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