Introducción a Ruby on Rails

Dr. Diego Lz. de Ipiña Gz. de Artaza
http://paginaspersonales.deusto.es/dipina
Cursos de Julio 2006, Universidad de Deusto,
11 y 12 Julio, 10:00 a 12:00
dipina@eside.deusto.es

Contenidos

1. Introducción a la programación de scripting con Ruby. (1h y 10')
   - Instalación de Ruby
   - Modo interactivo y de scripting
   - Tipos de datos en Ruby
   - Primitivos
   - Estructuras de Datos Complejas: Arrays, Iteradores y Mapas
   - Programación Orientada a Objetos en Ruby
   - Gestión de errores y de módulos
2. Programación avanzada en Ruby. (40 ')
   - Programación de redes con sockets
   - Multithreading en Ruby
   - Acceso a Bases de Datos desde Ruby

Descanso (10')

Contenidos

3. Introducción a Ruby on Rails (30')
   - Filosofía DRY-COC
   - Instalación de Rails
   - Desarrollo de una aplicación CRUD en 6 pasos
4. Entendiendo el modelo MVC de Rails (40')
   - Concepto de Scaffolding
   - Controladores en Rails: clase `ApplicationController`
   - Modelo en Rails: clase `ActiveRecord`
   - Vistas en Rails: ficheros `.rhtml`
5. Conceptos avanzados en Rails (40')
   - Caching, validación y AJAX
   - Integración con Apache
   - Distribuyendo aplicaciones Rails
6. Casos de Éxito de Rails y Conclusión (10').

Ruby

- Un lenguaje de scripting orientado a objetos "puro"
- Desarrollado por Yukihiro "Matz" Matsumoto
- Ruby combina:
  - Elegancia conceptual de Smalltalk
  - La facilidad de aprendizaje y uso de Python
  - El pragmatismo de Perl
- Liberado en 1993
- Más famoso que Python en Japón
- Website: http://www.ruby-lang.org/en/

Ventajas de Ruby

- Simple: fácil de aprender y mantener
- Poderoso
  - "Language stays out of your way"
- Equipado con excelentes librerías
- Desarrollo rápido
- Código abierto
- "Divertido"

Desventajas de Ruby

- Rendimiento comparable a Perl o Python, pero lejos de C o C++
  - Podemos extender Ruby con estos lenguajes
- No existen muchas frameworks desarrolladas en Ruby
  - Ruby on Rails (http://www.rubyonrails.com/) es la excepción
- No existe una framework de GUI multi-plataforma ampliamente aceptada
- RAA – Ruby Application Archive (http://raa.ruby-lang.org/)
  - No tan grande como CPAN – Comprehensive Perl Archive Network (http://www.cpan.org/)
- Peros:
  - Documentación en progreso
  - No es un lenguaje demasiado conocido en países occidentales
  - Más pensado para Linux que Windows
  - No tiene un buen soporte de Unicode todavía

Instalación Ruby

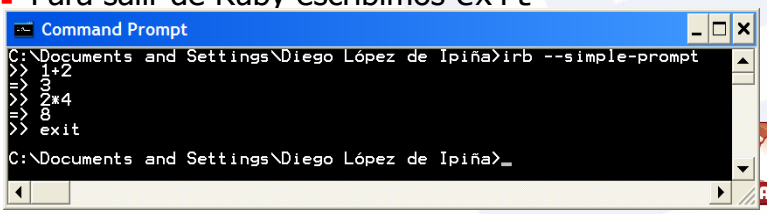
- Para Windows se puede bajar `.exe` de: http://rubyinstaller.rubyforge.org/wiki/wiki.pl?RubyInstaller
  - Utilizaremos la versión 1.8.2-14 de:
    - http://rubyforge.org/frs/download.php/2407/ruby182-14.exe
- Instalación en Linux:
  - Podemos bajarnos las fuentes y encontrar instrucciones sobre cómo instalarlo en:
    - http://www.ruby-lang.org/en/20020102.html

Características de Ruby

- Diseñado siguiendo la filosofía POLS (Principle of Least Surprise)
  - Las cosas funcionan del modo que esperas, "sin sorpresas"
- Dynamically typed/late binding, las decisiones en tiempo de ejecución ¡Peligro!
- Todo es un objeto o un método
- Herencia simple con módulos *mixin*
- Bloques y cierres
- Rangos
- Metaclases
- Introspección
- Manejo de excepciones
- Integración con C
- Convención de nombres: `@instanceVar`, `@@classVar`, `$global`, `CONSTANT`, `everythingElse`
- Portable, multiplataforma, open source
- Seguridad
- Internacionalización en progreso (1.9)
- Muchas librerías

Nuestros primeros pasos en Ruby

- Modo interactivo
  - Para trabajar en modo interactivo usamos `irb` (Interactive Ruby Shell) escribiendo en consola: `irb --simple-prompt`
  - Escribir operaciones aritméticas para que IRB se comporte como una calculadora
    - Soporta los operadores típicos +, -, *, /, % y **
    - 2**3 = 8
  - Para salir de Ruby escribimos `exit`



Tipos de Datos Primitivos

- Numéricos:
  - Integers, una división entre enteros devolverá un entero
  - Floats, es un número con decimales
  - Ruby puede utilizar números muy largos
    - 192,349,562,563,447 se puede expresar como 192_349_562_563_447
    - Números aún más largos se pueden expresar como:
      - 1.7e13 significa 1.7×10^{13}

Universidad de Deusto
Deustuko Unibertsitatea

Tipos de Datos Primitivos

- Strings:
  - Grupos de caracteres
  - Algunos ejemplos:
    - "Hello."
    - "Ruby rocks."
    - "5 is my favorite number... what's yours?"
    - 'Two plus two is #{2+2}' # iiino interpola!!!
  - Operaciones:
    - `"hi " * 3` → `"hi hi hi "`
    - `"hi".capitalize` → `Hi`
    - `"hi".reverse` → `ih`
    - `"hi".upcase` → `HI`
    - Probar: `downcase` y `swapcase`
    - `chomp` devuelve un string quitando los saltos de línea al final de la palabra (`"hola\n".chomp` → `"hola"`)
- Booleanos: `true` y `false`.
- `nil` es el valor nulo en Ruby
- Observar que los nombres de clases empiezan por mayúscula

Universidad de Deusto
Deustuko Unibertsitatea

En Ruby Todo son Objetos

Tipo de datos	Clase Ruby
integer	`Integer`
decimals	`Float`
text	`String`

Clase Ruby	Algunos Métodos
`Integer`	`+ - / * % **`
`Float`	`+ - / * % **`
`String`	`capitalize, reverse,length, upcase`

Método	Conversores	
	De	A
`String#to_i`	String	Integer
`String#to_f`	String	Float
`Float#to_i`	Float	Integer
`Float#to_s`	Float	String
`Integer#to_f`	Integer	Float
`Integer#to_s`	Integer	String

Fechas en Ruby

```
# ejemploFechas.rb
require 'date'
d = Date.today
puts d << 2
t = Time.new
t.year
t.strftime("%d-%b-%y")
```

Constructores literales para clases built-in

- String → "new string" o 'new string'
- Symbol → :symbol o :"symbol with spaces"
- Array → [1,2,3]
- Hash → {"New York" => "NY", "Oregon" => "OR"}
- Range → 0…10 o 0..9
- Regexp → /([a-z]+)/

Operaciones sobre strings

- `capitalize, upcase, downcase, swapcase, strip, lstrip, rstrip, chop, chomp, reverse, replace, "abc"[2], "abc"[2].chr, "abc"[2,1], equal?`
- Ejemplo:

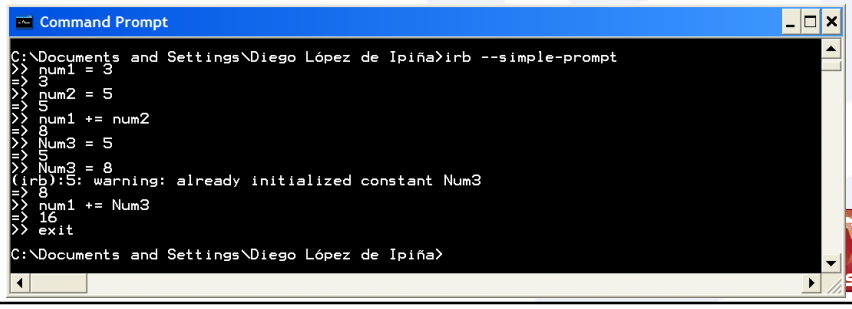
```
>> str="Hello"
=> "Hello"
>> str.reverse
=> "olleH"
>> str
=> "Hello"
>> str.reverse! # ¡¡¡método bang!!! Cambian campo
=> "olleH"
>> str
=> "olleH"
```

Símbolos

- Símbolos y strings representan unidades de texto
  - Un objeto símbolo por cada texto
    - `:a.equal?(:a) # true`
    - `"a".equal?("a") # false`
  - Los símbolos son inmutables
- Ejemplo:
  - `:a`
  - `“a".to_sym`
  - `:a.to_s`

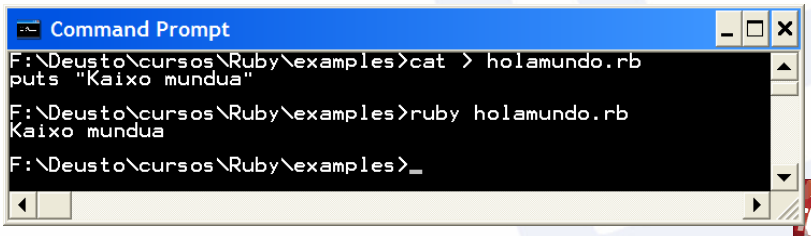
Variables y Constantes

- Las variables deben ser declaradas en minúsculas
- Las constantes empiezan por una letra mayúscula y son seguidas por n caracteres en minúscula
- Si intentas cambiar el valor de una constante Ruby dará un warning pero no lo impedirá (-:



Nuestro Primer Programa

- Modo script:
  - Crea el fichero `holamundo.rb`
  - Inserta la línea `puts "Kaixo mundua"`
    - `print` hace lo mismo sin imprimir una nueva línea
    - `printf "Number: %5.2f, String: %s", 1.23, "hello"`
  - Ejecuta el programa como: `ruby holamundo.rb`



Universidad de Deusto
Deustuko Unibertsitatea

Nuestro Primer Programa

- Para leer datos en Ruby utilizamos `gets`:

```
puts "¿Cómo te llamas?"
name = gets
puts "Hola " + name + ". ¿Qué tal
  estás?"
```

- Unicode en Ruby:
  - Funciona en irb
  - Pero no desde el intérprete
  - I18n ¡verde!

Universidad de Deusto
Deustuko Unibertsitatea

Nuestro Primer Programa

- Sólo podemos añadir un string a otro
  - Lo siguiente daría un error:
    `puts "The answer is " + num2 # --> Error`
  - Para corregirlo haríamos:
    `puts "The answer is " + num2.to_S`
- Una característica interesante de Ruby es que podemos concatenar las llamadas a funciones y **no es necesario usar paréntesis**:
  - `name = gets.chomp`
- En Linux podemos hacer que un programa sea ejecutable:
  - Cambiando los derechos del fichero `chmod +x prog.rb'`
  - Insertando como primera lína `#!/usr/bin/env ruby`
  - Creamos un fichero con extensión `.rb`
  - Lo ejecutamos desde consola con `./nombre-fichero.rb`

Estructuras de control: `if`

- Igual a la de otros lenguajes de programación con el nemónico `if`
- Los operadores lógicos son `and`, `&&`, `or`, `||`, `not` y `!`
- Python incorpora `elif` y Ruby `elsif`:

```
if age >= 60
    puts "Senior fare"
elsif age >= 14
    puts "Adult fare"
elsif age > 2
    puts "Child fare"
else
    puts "Free"
end
```

- La forma negativa del `if` es `unless`:

```
unless aSong.duration > 180 then
  cost = .25
else
  cost = .35
end
```

- También hay soporte para la expresión condicional en C:

```
cost = aSong.duration > 180 ? .35 : .25
```

Estructuras de control: case

- También existe la sentencia `case`
- En Ruby todo son expresiones, por lo tanto un `case` o un `if` devuelven el valor de la última expresión ejecutada

```
kind = case year
            when 1850..1889 then "Blues"
            when 1890..1909 then "Ragtime"
            when 1910..1929 then "New Orleans Jazz"
            when 1930..1939 then "Swing"
            when 1940..1950 then "Bebop"
            else                 "Jazz"
       end
```

Estructuras de control: while

- Sigue el formato:

```
while cond
   ...
end
```

- Calcular potencias de 2 menor que 10000 (potencias2.rb):

```
#!/usr/bin/ruby
# Calcular la potencia de 2 menor que 10000
number=1
while number < 10_000
   number *= 2
end
number /=2
puts number.to_s + " es la potencia de 2 mayor menor que
   10.000“
```

- La forma negativa del `while` es `until`:

```
until playList.duration > 60
   playList.add(songList.pop)
end
```

Estructuras de control: loops

- Sentencia `loop`:

```
n=1
loop do
   n = n+1
   next unless n == 10
   break
end
```

- El método `times` hace que iteremos N veces:

```
4.times do
   puts "hola"
end
```

- Contando:

```
count = 0
5.times do
    count +=1
    puts “count = “ + count.to_s
end
```

Arrays

- La clase `Array` se utiliza para representar colecciones de objetos:
- Ejemplo:

```
$ irb --simple-prompt
>> nums = ["cero", "uno", "dos", "tres", "cuatro"]
=> ["cero", "uno", "dos", "tres", "cuatro"]
>> nums.class
=> Array
>> exit
```

- Observar que `class` devuelve el tipo de un objeto
- Podemos acceder a un elemento mediante `nums[0]`
- Podemos añadir nuevos elementos con: `nums[5] = "cinco"` o `nums.unshift(5)` (al principio) o `nums.push(5)` (al final)
- Eliminar elementos con `shift()` y `pop()`
- Se pueden concatenar arrays con `[1,2,3].concat([4,5,6])`
- En un `Array` puedes colocar cualquier tipo de dato

Operaciones sobre Arrays

- Podemos ordenar elementos con el método `Array#sort`
- Invertir su orden con `Array#reverse`
- Calcular su longitud con `Array#length`
- Reducir dimensiones con `Array#flatten`
- Eliminar duplicados con `Array#uniq`
- Se pueden realizar operaciones sobre arrays con los operadores aritméticos +, - y *
- Imprimimos el contenido de arrays con `puts`
- Ejemplo: `ejemploArrays.rb`

Operaciones sobre Arrays

```
# ejemploArrays.rb
primos = [11, 5, 7, 2, 13, 3]
puts primos
puts primos.sort
puts primos.reverse
puts "Longitud: " + primos.length.to_s
primos = primos + [34]
puts "primos += 34: "

index = 0
primos.length.times do
   puts primos[index].to_s
   index += 1
end

primos = primos - [11, 5]
puts "primos - [11, 5]:"
index = 0
primos.length.times do
   puts primos[index]
   index += 1
end
```

Iterando sobre una secuencia de valores

```
celsius = [0, 10, 20]
puts “Celsius\Fahrenheit”
for c in celsius
   puts
 “c\t#{Temperature.c2f(c)}”
end
```

Universidad de Deusto
Deustuko Unibertsitatea

Modificadores de Sentencias

- Útiles si un `if` o `while` son seguidos por una única sentencia

```
if radiation > 3000
  puts "Danger, Will Robinson"
end
```

- Con un statement modifier sería:

```
puts "Danger, Will Robinson" if radiation > 3000
```

- Otro ejemplo:

```
while square < 1000
   square = square*square
end
```

- Con un statement modifier sería:

```
square = square*square  while square < 1000
square = square*square unless square >= 100
```

Universidad de Deusto
Deustuko Unibertsitatea

Bloques de Código y `yield`

- Un code block es un componente opcional de una llamada a un método
- Es un conjunto de sentencias entre { y } o entre el par do/end
- Un code block puede ser invocado desde el método llamado mediante el método `yield`
  - Es una llamada de código de vuelta
- Se pueden pasar argumentos a un code block entre ||
- La principal razón de crear bloques de código es refinar métodos existentes
  - Puedes reutilizar de diferentes maneras un mismo código
  - Se utiliza esta técnica internamente en la implementación del método each de una secuencia
- Un ejemplo de un bloque sería:

```
{ puts "Hello" } # esto es un bloque
```

  - O:

```
do                      #
  club.enroll(person)   # y esto otro
  person.socialize      #
end                     #
```

Universidad de Deusto
Deustuko Unibertsitatea

Bloques de Código y `yield`

```
# ejemploConversorCodeBlock.rb
def temp_chart(temps)
   for temp in temps
      converted = yield(temp)
      puts "#{temp}\t#{converted}"
   end
end
celsiuses =
  [0,10,20,30,40,50,60,70,80,90,100]
temp_chart(celsiuses) {|cel| cel * 9 / 5
  + 32 }
```

Universidad de Deusto
Deustuko Unibertsitatea

Concepto de Bloques

- Los métodos pueden invocar a un bloque de código usando la sentencia `yield`
- Por ejemplo:

```
class Array
  def each
   for i in 0..(size-1) do
        yield self[i]
   end
  end
end
[1, 2, 3].each { |x| print x*x, "\n" }
```

Universidad de Deusto
Deustuko Unibertsitatea

Más ejemplos Bloques

- Ejemplo:

```
array = [1, 2, 3]
x = 5
array.collect! { | elem | elem + x }
p array
```

- Otro ejemplo:

```
def names
    yield("Joe")
    yield("Sandy")
    yield("Melissa")
end

names do |name|
    puts "Hello " + name + ", how are you?"
end
```

Universidad de Deusto
Deustuko Unibertsitatea

Filtrado sobre Arrays

- Seleccionar un elemento con `find`

```
>> [1,2,3,4,5,6,7,8,9,10].find {|n|
  n>5 }
=> 6
```

- Seleccionar un conjunto de elementos:
  - `a = [1,2,3,4,5,6,7,8,9,10]`
  - `a.find_all {|item| item > 5 }`
- Otros métodos de interés: `Array#size`, `Array#empty?`, `Array#include?(item)`, `Array#any?{|item| test}`, `Array#all?{|item| test}`

Iteradores en Ruby

- Un iterador es un método que te permite acceder a elementos uno a uno
- Los Arrays ofrecen el iterador `Array#each` y `Array#map`

```
amigos = ["Ane", "Inge", "Jon", "Eneko"]
amigos.each do |amigo|
     puts "Tengo un amig@ llamad@ " + amigo
end
amigos.each_with_index do |amigo, i|
     puts "Tengo un amig@ #{i} llamad@ " + amigo
end
```

- `n.times do ....`
  - Es realmente un iterador.
- Imprimir todos los amigos:

```
amigos.length.times do |i|
   puts "Tengo un amig@ llamad@ " + amigos[i]
end
```

- Imprimir los amigos en orden alfabetico

```
amigos.sort.each do |amigo|
   puts "Tengo un amig@ llamad@ " + amigo
end
```

Iteradores en Ruby

```
0.upto(9) do |x|
  print x, " "
end
```

- produce:

```
0 1 2 3 4 5 6 7 8 9
```

```
0.step(12, 3) {|x| print x, " " }
```

- produce:

```
0 3 6 9 12
```

Arrays Asociativos o Hashes

- En Ruby un mapa asociativo se define como:

```
# amigo = Hash.new o # amigo = Hash ["nombre" => "Diego",“apellido1" =>
   "Lz. de Ipiña“]
amigo = {
   "nombre" => "Diego",
   "apellido1" => "Lz. de Ipiña",
   "apellido2" => "Gz. de Artaza",
   "direccion" => "Dato Kalea",
   "ciudad" => "Vitoria-Gasteiz",
   "provincia" => "Araba"
}
```

- Los hashes asocian un valor a una clave
  - Claves (`keys`): nombre, apellido1, apellido2, etc.
  - Valores (`values`): Diego, Lz. de Ipiña, Gz. de Artaza, etc.
- `Hash#each` nos permite iterar sobre un mapa:

```
amigo.each do |key, value|
   puts key + " => " + value
end
```

- `Hash#each_key` nos permite iterar sobre todas las claves
- `Hash#each_value` sobre cada valor
- Otros métodos: `fetch`, `values_at`, `update`, `invert`, `has_key?`, `has_value?`
- Revisar ejemplo complejo `agenda.rb`

Ejemplo Hashes

```
#ejemplohash.rb
amigo = {
    "nombre" => "Diego",
    "apellido1" => "Lz. de Ipiña",
    "apellido2" => "Gz. de Artaza",
    "direccion" => "Dato Kalea",
    "ciudad" => "Vitoria-Gasteiz",
    "provincia" => "Araba"
}

puts "\n\nContenido del mapa:\n"
amigo.each do |key, value|
    puts key + " => " + value
end

# para recuperar todas las claves hacemos
puts "\n\nClaves del mapa:\n"
amigo.each_key do |key|
    puts key
end


# para recuperar todas los valores hacemos
puts "\n\nValores del mapa:\n"
amigo.each_value do |value|
    puts value
end
```

Ordenando Arrays Asociativos

- Consideremos la siguiente estructura de datos:

```
friends = [
    [ "Joe", "Smith" ],
    [ "Melissa", "Adams"],
    [ "Sandy", "Koh" ]
]
```

- Por defecto `sort` ordenaría por la primera entrada
- El operador <=> simplifica la devolución de valores:

```
>> 3 <=> 1  # 3 > 1 so <=> devuelve 1
=> 1
>> 3 <=> 3  # 3 == 3 so <=> devuelve 0
=> 0
>> 3 <=> 5  # 3 < 5 so <=> devuelve -1
=> -1
```

- Para ordenar por la segunda entrada haríamos:

```
friends = friends.sort do |a,b|
    a[1] <=> b[1]   # Ordenamos por la segunda entrada
end
```

- Revisar ejemplo `agenda.rb`.

Módulo Enumerable

- Para ver los métodos que ofrece hacer lo siguiente:

```
Enumerable.instance_methods(false).sort
```

- Ejemplo:

```
class Rainbow
   include Enumerable
   def each
      yield "red"
      yield "orange"
      yield "yellow"
      yield "green"
      yield "blue"
      yield "indigo"
      yield "violet"
   end
end
r = Rainbow.new
y_color = r.find {|color| color[2,1] == 'y' }
```

Universidad de Deusto
Deustuko Unibertsitatea

Ordenando Colecciones

- Para ordenar instancias de una clase hay que definir el método <=> en esa clase
- Colocar sus elementos en una colección:
  - Mix-in con Enumerable implementa:
    - `sort` → `["2", 1, 5, "3", 4, "6"].sort {|a, b| a.to_i <=> b.to_i }`
    - `sort_by` → `["2", 1, 5, "3", 4, "6"].sort_by {|a| a.to_i}`

Universidad de Deusto
Deustuko Unibertsitatea

Funciones

- Una función es un método no asociado a un objeto
- Ejemplo de función:

```
def say_hi
    puts "Hello, How are you?"
end
say_hi
```

- Podemos pasar parámetros a funciones:

```
def say_hi(name="Diego")
    puts "Hello " + name + ", How are you?"
end
say_hi("Daniel")
```

- El valor devuelto por Ruby es el de la última expresión evaluada, aunque se puede devolver un valor usando `return`
  - Parámetro precedido por `*`, indica array de parámetros
- Revisar `agendaConFunción.rb`.

Universidad de Deusto
Deustuko Unibertsitatea

Clases

- La definición de una clase sigue la siguiente sintaxis:

```
class Direccion
   # constructor
   def initialize(calle)
      # atributo de la clase
      @calle = calle
   end
   # método de la clase dirección
   def calle
      @calle
   end
   def calle=(nuevoValor)
      @calle = nuevoValor
   end
end
# invocación al constructor
dir = Direccion.new("Dato Kalea")
puts dir.calle
```

Universidad de Deusto
Deustuko Unibertsitatea

Más sobre clases

- Para hacer que un atributo sea leíble usamos `attr_reader`, escribible `attr_writer` y ambas cosas `attr_accessor`:

```
class Direccion
   attr_reader :calle
   attr_reader :ciudad
   def initialize(calle, ciudad)
      @calle = calle
      @ciudad = ciudad
   end
   # Sirve para modificar el contenido de un atributo
   def calle=(calle)
      @calle = calle
   end
end

dir = Direccion.new("Dato Kalea", "Vitoria-Gasteiz")
puts dir.calle + " " + dir.ciudad
dir.calle = "Calle Dato Kalea"
puts dir.calle
```

- Podemos hacer lo mismo que antes con `attr_writer`, sin necesidad de definir el método `calle=`
- Para imprimir un objeto sobreescribimos el método `to_s`
- Para inspeccionar una clase usamos `inspect`
- Revisar ejemplo `ejemploClaseMetodos.rb`

Ejemplo AddressBook

- Revisar `addressbook.rb`
- Cómo añadir elementos a la agenda de manera ordenada:

```
class AddressBook
    def add(person)
        @persons += [person]
        @persons = @persons.sort{|a,b| by_name(a,b)}
    end

    def by_name(a,b)
        if  a.first_name  == b.first_name
            a.last_name  <=> b.last_name
        else
            a.first_name <=> b.first_name
        end
    end
end
```

Ejemplo Song

```
class Song
   def initialize(name, artist, duration)
       @name     = name
       @artist   = artist
       @duration = duration
   end

   def to_s
       "Song: #{@name}--#{@artist} (#{@duration})"
   end
end

aSong = Song.new("Bicylops", "Fleck", 260)
puts aSong.to_s     #     "Song: Bicylops--Fleck
   (260)"
```

Herencia

```
#KaraokeSong.rb
require "Song"
class KaraokeSong < Song
  def initialize(name, artist, duration, lyrics)
      super(name, artist, duration)
      @lyrics = lyrics
  end
end

aSong = KaraokeSong.new("My Way", "Sinatra", 225,
  "And now, the...")
aSong.to_s  #     "Song: My Way--Sinatra (225)"
```

Operador ===

```
class Ticket
   attr_accessor :venue, :date
   def initialize(venue, date)
      self.venue = venue
      self.date = date
   end
   def ===(other_ticket)
      self.venue == other_ticket.venue
   end
end

ticket1 = Ticket.new("Town Hall", "07/08/06")
ticket2 = Ticket.new("Conference Center", "07/08/06")
ticket3 = Ticket.new("Town Hall", "08/09/06")

puts "ticket1 is for an event at: #{ticket1.venue}."
case ticket1
   when ticket2
      puts "Same location as ticket2!"
   when ticket3
      puts "Same location as ticket3!"
   else
      puts "No match"
end
```

Sobrecarga de Operadores

- Aritméticos
  - + | - | * | / | % → def +(x) → obj + x
- Get/set/append
  - [] → def [](x) → obj[x]
  - []= → def []=(x) → obj[x] = y
  - << → def <<(x) → obj << x
- Comparación
  - == | > | < | >= | <= → def ==(x) → obj == x
- Ejemplo:

```
# ejemploSobrecargaOperadores.rb
obj = Object.new
def obj.+(other_obj)
   "trying to add something"
end
puts obj + 100 # output: "trying to add something"
```

Comparaciones

```
class Bid
  include Comparable
  attr_accesor :contractor
  attr_accesor :estimate
  def <=>(other_bid)
    # self.estimate <=> other_bid.estimate
    if self.estimate < other_bid.estimate
      -1
    elsif self.estimate > other_bid.estimate
      1
    else
      0
    end
  end
end
```

Bloques `main`

- Para recuperar argumentos acceder al array global ARGV
- Siguen el siguiente formato en Ruby:

```
if __FILE__ == $0
  aSong = KaraokeSong.new("My Way",
  "Sinatra", 225, "And now, the...")
  puts aSong.to_s      #    "Song: My
  Way--Sinatra (225)"
end
```

Definiendo Iteradores

- Iteradores para la clase `AddressBook`

```
class AddressBook
    def each
        @persons.each { |p| yield p }
    end
end

class AddressBook
    def each_address
        @persons.each { |p| yield p.address }
    end
end
```

- Revisar ejemplo `testAddressBook.rb`

Ejemplos Iteradores

- Todas las clases tienen disponibles los siguientes métodos de interacción:
  - each, each_with_index, sort, collect, detect, reject, select, entries, find, grep, include?, map, max, reject y más
- Ejemplo:

```
array.each { | elem | puts elem }
hash.each { | key, val | puts "#{key} =>
  #{val}" }
file.each { | line | puts line }
array.each_with_index { | elem, i |
   puts "array[#{i}] = #{elem}"
}
array.include?(42)
```

Rangos

- Algunos ejemplos de rangos son:

```
(0 .. 9).each { | i | sum += i }
('a' .. 'z').each { | str | puts str }
(0 ... array.size).each { | i | puts array[i]
  }
```

- Ejemplo:

```
5.times {  print "*" }
3.upto(6) {|i|  print i }
('a'..'e').each {|char| print char }
```

- Produce:

```
*****3456abcde
```

Métodos públicos/privados

- Puedes definir el scope de métodos con las palabras clave `public` y `private`

```
class SomeClass
    def method1 # default to public
        ...
    end
  private   # subsequent methods are private.
    def method2 # private method
        ...
    end
    def method3 # private method
        ...
    end
  public    # Set back to public.
    def method4 # public method
        ...
    end
end
```

Reutilización de Código

- Podemos importar código ya creado con `require`:

```
require "addressbook"
#  Sandy
addr = Address.new
addr.street = "324 Campus Dr."
addr.city   = "College Park"
addr.state  = "MD"
addr.zip    = "23659"
puts addr
```

Excepciones

- Encerramos el código que puede lanzar una excepción en: `begin/end`
- `rescue` indica a Ruby que excepciones son soportadas
- Para lanzar una excepción usamos `raise`
- El bloque `finally` en Ruby es `ensure`
- Ejemplo:

```
f = File.open("testfile")
begin
  # .. process
rescue
  # .. handle error
else
  puts "Congratulations-- no errors!"
ensure
  f.close unless f.nil?
end
```

Excepciones

- Todas derivan de Exception

```
class MyNewException < Exception
end
raise MyNewException
```

- Algunas excepciones comunes son:
  - `RuntimeError`
  - `NoMethodError`
  - `NameError`
  - `IOError`
  - `Errno::error`
  - `TypeError`
  - `ArgumentError`
  - `ZeroDivisionError`

Ejemplo Excepciones

```
opFile = File.open(opName, "w")
begin
  # Exceptions raised by this code will
  # be caught by the following rescue clause
  while data = socket.read(512)
    opFile.write(data)
  end
rescue SystemCallError
  $stderr.print "IO failed: " + $!
  opFile.close
  File.delete(opName)
  raise
end
```

Expresiones Regulares

- Algunos ejemplos de expresiones regulares:
  - `/\d\d:\d\d:\d\d/` # a time such as 12:34:56
  - `/Perl.*Python/` # Perl, zero or more other chars, then Python
  - `/Perl\s+Python/` # Perl, one or more spaces, then Python
  - `/Ruby (Perl|Python)/` # Ruby, a space, and either Perl or Python
- El operador de evaluación de expresiones regulares es '=~"
  - Devuelve `nil` si no se encuentra
  - O la posición en la que se ha encontrado la expresión regular
- Ejemplo: determina si un mensaje contiene las palabras 'Perl' o 'Python'.

```
if line =~ /Perl|Python/
  puts "Scripting language mentioned: #{line}"
end
```

- Podemos reemplazar substrings encontrados mediante expresiones regulares:
  - `line.sub(/Perl/, 'Ruby')` # replace first 'Perl' with 'Ruby'
  - `line.gsub(/Python/, 'Ruby')` # replace every 'Python' with 'Ruby'

Convención de Identificadores

- Las variables locales, los parámetros de métodos y los nombres de métodos deben empezar por una letra en minúsculas o un "_"
- Las variables globales tienen como prefijo $
- Las variables de instancias compienzan con @.
- Las variables de clases empiezan por @@.
- Los nombres de clases, módulos y constantes deben empezar por letra mayúscula
- Métodos bang (modifican el receptor) acaban en !, e.j. `sort!`
- Métodos que acaban en ? devuelven `true` o `false`

Módulos

- Un módulo es un mecanismo de agrupar métodos, clases y constantes
  - Proveen un espacio de nombres
  - Implementan la facilidad mixin
- Creando un módulo:

```
module Trig
  PI = 3.141592654
  def Trig.sin(x)
   # ..
  end
  def Trig.cos(x)
   # ..
  end
end
```

- Usándolo:

```
require "trig"
y = Trig.sin(Trig::PI/4)
```

Módulos

- Permiten olvidarnos de la herencia múltiple e implementar el mecanismo de *mixin*
- Podemos incluir un módulo dentro de una clase
  - Los métodos del módulo pasan a formar parte de la clase → mixed in
- Los mixins son muy útiles para añadir funcionalidad a una clase
  - Por ejemplo, `Comparable`

Ejemplo mixin module

```
module Debug
  def whoAmI?
    "#{self.type.name} (\##{self.id}): #{self.to_s}"
  end
end
class Phonograph
  include Debug
  # ...
end
class EightTrack
  include Debug
  # ...
end
ph = Phonograph.new("West End Blues")
et = EightTrack.new("Surrealistic Pillow")
ph.whoAmI? #   "Phonograph (#537766170): West End Blues"
et.whoAmI? #   "EightTrack (#537765860): Surrealistic Pillow"
```

Input/Output Ruby

- Un objeto IO es un canal bidireccional entre Ruby y un programa externo
  - Subclases son `File` y `BasicSocket`
- `IO#print` para escribir datos
- Más info en: http://www.rubycentral.com/book/tut_io.html
- Ejemplo (`readFile.rb`):

```
aFile = File.new("testfile", "r")
aFile.each_byte {|ch| putc ch; putc ?. }
aFile.each_line {|line| puts "Got #{line.dump}" }
aFile.each_line("e") do |line|
  puts "Got #{ line.dump }"
end
IO.foreach("testfile") { |line| puts line }
arr = IO.readlines("testfile")
arr.length
puts arr[0]
aFile.close
```

Programación de redes

- Ejemplo uso sockets:

```
require 'socket'
client = TCPSocket.open('localhost', 'finger')
client.send("oracle\n", 0)    # 0 means standard
  packet
puts client.readlines
client.close
```

- Ejemplo uso HTTP:

```
require 'net/http'
h = Net::HTTP.new('www.pragmaticprogrammer.com',
  80)
resp, data = h.get('/index.html', nil)
if resp.message == "OK"
  data.scan(/<img src="(.*?)"/) { |x| puts x }
end
```

Hilos en Ruby

- La clase Thread (http://www.ruby-doc.org/core/)
- Para más información: http://www.rubycentral.com/book/tut_threads.html
- Ejemplo:

```
# ejemploSimpleHilo.rb
Thread.abort_on_exception = true
t1 = Thread.new do
   puts  "In new thread"
   raise "Exception from thread“
end
sleep(1)
puts "not reached"
```

Ejemplo Hilos

```
# ejemploHiloHTTP.rb
require 'net/http'

pages = %w( www.rubycentral.com
            www.rubyonrails.org
            www.pragmaticprogrammer.com
          )

threads = []

for page in pages
   threads << Thread.new(page) { |myPage|
        h = Net::HTTP.new(myPage, 80)
        puts "Fetching: #{myPage}"
        resp, data = h.get('/', nil )
        puts "Got #{myPage}:  #{resp.message}"
   }
end

threads.each { |aThread|  aThread.join }
```

Universidad de Deusto
Deustuko Unibertsitatea

Bases de Datos en Ruby

- Se puede trabajar:
  - Directamente con el módulo mysql
  - Con el módulo DBI → independencia de nuestro código a la BBDD
- Instrucciones sobre cómo instalarlo en Windows: http://www.joeygibson.com/blog/tech/ruby/RubyMySQL.html
- Documentación sobre el uso de Ruby y MySQL en: http://www.kitebird.com/articles/

Universidad de Deusto
Deustuko Unibertsitatea

Ejemplo Ruby y MySQL

```
# simpleMySQL.rb - simple MySQL script using Ruby MySQL module
#mysql> GRANT ALL ON test.* TO 'testuser'@'localhost' IDENTIFIED BY
    'testpass';
#mysql> CREATE DATABASE test;
require "mysql"
begin
    # connect to the MySQL server
    dbh = Mysql.real_connect("localhost", "testuser", "testpass",
    "test")
    # get server version string and display it
    puts "Server version: " + dbh.get_server_infodbh.query("DROP TABLE
    IF EXISTS animal")
    dbh.query("CREATE TABLE animal(name CHAR(40), category CHAR(40))")
    dbh.query("INSERT INTO animal (name, category) VALUES ('snake',
    'reptile'),('frog', 'amphibian'),('tuna', 'fish'),('racoon',
    'mammal')")
    printf "%d rows were inserted\n", dbh.affected_rows
rescue MysqlError => e
    print "Error code: ", e.errno, "\n"
     print "Error message: ", e.error, "\n"
ensure
    # disconnect from server
     dbh.close
end
```

MySQL y Ruby

- Bajarse mysql-ruby (mysql-ruby-2.7.1.tar.gz) de: http://tmtm.org/downloads/mysql/ruby/
- El módulo MySql define 4 clases:
  - `MySql` → clase principal, define métodos para conectar al servidor, enviarle consultas y tareas administrativas
  - `MySqlRes` → clase que guarda resultados
  - `MySqlField` → permite obtener metadatos de columnas
  - `MySqlError` → indica un error
- En todo programa importar modulo `mysql`:
  - `require "mysql"`
- Para conectarse a una base de datos usar método `real_connect` de clase MySql
  - `dbh = Mysql.real_connect("localhost", "testuser", "testpass", "test")`

Ruby y MySql: Consultas sin Resultados

- Usar método `query` para realizar estas consultas
  - Recuperar el número de filas afectadas con `affected_rows`
- Ejemplo:

```
dbh.query("INSERT INTO animal (name,
  category) VALUES ('snake',
  'reptile'), ('frog', 'amphibian'),
  ('tuna', 'fish'), ('racoon',
  'mammal')")
printf "%d rows were inserted\n",
  dbh.affected_rows
```

Ruby y MySql: Consultas con Resultados

- Los siguientes pasos serían necesarios:
  - Invocar a `query` y guardar los resultados en una instancia de `MysqlRes`
  - Usar los métodos provistos por `MysqlRes` para recuperar filas (`fetch_row`) o metadatos o simplemente el iterador `each`
  - Invocar `free` para liberar el result set
- Ejemplo:

```
res = dbh.query("SELECT name, category FROM animal")
res.each do |row|
   printf "%s, %s\n", row[0], row[1]
end
printf "%d rows were returned\n", res.num_rows
while row = res.fetch_hash do
   printf "%s, %s\n", row["name"], row["category"]
end
res.each_hash do |row|
   printf "%s, %s\n", row["name"], row["category"]
end
res.free
```

Ruby y MySql: Avanzado

- Utilizar `scape_string` para introducir strings con caracteres especiales:
  - category = dbh.escape_string("don't know")
- Ejemplo de extracción de metadatos:

```
# mysql/simple.rb
res = dbh.query(query)
if res.nil? then
   puts "Query has no result set"
   printf "Number of rows affected: %d\n", dbh.affected_rows
else
   puts "Query has a result set"
   printf "Number of rows: %d\n", res.num_rows
   printf "Number of columns: %d\n", res.num_fields
   res.fetch_fields.each_with_index do |info, i|
     printf "--- Column %d (%s) ---\n", i, info.name
     printf "table:      %s\n", info.table
     printf "def:        %s\n", info.def
     printf "type:       %s\n", info.type
     printf "length:     %s\n", info.length
     printf "max_length: %s\n", info.max_length
     printf "flags:      %s\n", info.flags
     printf "decimals:   %s\n", info.decimals
   end
   res.free
```

Ruby DBI Module

- Ruby DBI es una interfaz independiente de la base de datos
  - Similar a Perl y Python DBI
- Está compuesta de dos capas:
  - The database interface layer (DBI)
  - The dababase driver layer (DBD)
- Necesitamos:
  - Ruby MySQL module
  - Ruby DBI module
- Instalación:
  - Download dbi-0.1.0.tar.gz de http://rubyforge.org/frs/?group_id=234&release_id=4323
  - Ejecuta:
    - `ruby setup.rb config --with=dbi,dbd_mysql`
    - `ruby setup.rb setup`
    - `ruby setup.rb install`

Ruby DBI Module

- `DBI.connect` → permite conectarte a la base de datos y devuelve un db handle
- `dbh.do(<sentencia-sql>)` → permite ejecutar sentencias que no devuelven resultados
- `dbh.prepare` → permite preparar queries que van a devolver resultados

```
sth = dbh.prepare(statement)
sth.execute
... fetch rows ...
sth.finish
```

Ejemplo Ruby DBI

```
dbh = DBI.connect("dbi:Mysql:test:localhost", "testuser", "testpass")
# get server version string and display it
row = dbh.select_one("SELECT VERSION()")
puts "Server version: " + row[0]

dbh.do("DROP TABLE IF EXISTS people")
dbh.do("CREATE TABLE people (
      id INT UNSIGNED NOT NULL AUTO_INCREMENT,
      PRIMARY KEY (id),
      name CHAR(20) NOT NULL,
      height FLOAT)")
rows = dbh.do("INSERT INTO people (name,height)
   VALUES('Wanda',62.5),('Robert',75),('Phillip',71.5),('Sarah',68)")
printf "%d rows were inserted\n", rows

puts "\nResults of SELECT * FROM people with fetch"
sth = dbh.execute("SELECT * FROM people")
while row = sth.fetch do
   printf "ID: %d, Name: %s, Height: %.1f\n", row[0], row[1], row[2]
end
sth.finish
```

Ruby vs. Python

- Ruby usa una estructura de sentencias conservadora con `'end'`
- `self` no es necesario como en Python, en Ruby las variables de instancia se acceden con @
- Python separa tipos y clases, mientras que para Ruby no hay datos primitivos, todo son objetos
- En Python los tipos son más limitados: no hay herencia, no se pueden añadir nuevos métodos a tipos
- Los atributos de objetos no son accesibles por defecto en Ruby (concepto de scopes)
- Ruby no tiene tuplas
- Ruby tiene un bloque `loop` más orientado a objetos:

```
10.times do
   # ...
end
```

- En Ruby podemos definir nuestros propios iteradores.

Ruby vs. Python

- No hay demasiada diferencia entre Python y Ruby
  - La sintaxis de Ruby es más clara ya que es un lenguaje orientado a objeto PURO
  - Viene equipado de serie con muchas más características que Python
    - Aunque probablemente nunca las usaremos
- Python es mucho más utilizado y hay muchas más librerías disponibles
  - Está mucho mejor documentado
- Quizás elige Ruby si no conoces ninguno de los dos o si eres un purista
- Si eres práctico y no vas a usar Ruby on Rails ¡quédate con Python!

LAMP = Linux + Apache + MySQL + [PHP|Perl|Python]

- LAMP es un término utilizado para definir cómo MySQL puede ser utilizado en conjunción con Linux, Apache y cualquiera de los lenguajes de scripting cuyo nombre empieza por 'P':
  - Perl, Python y PHP (o **Ruby**)
    - PHP se está constituyendo como el lenguaje dominante en desarrollo web
      - PHP 5.0 y php.MVC
    - Ruby, Perl y Python son más de propósito general
    - Rails se está consituyendo como la framework basada en LAMP más sofisticada

LAMP = Open Source Web Platform

- Permite el desarrollo y explotación de portales web de alto rendimiento
  - Sólida y robusta
    - Sólo hay que tomar Apache como referencia
  - Muy popular
    - Por medio de la herramienta Netcraft se puede comprobar que muchas de las webs más populares usan Apache sobre Linux y tienen `mod_perl` o `mod_php` instalados.
- En general hace referencia a cualquier combinación de herramientas open source para desarrollo web
  - Linux podría reemplazarse por OpenBSD
  - MySQL por PostGreSQL
  - La 'P' podría traducirse en PHP, Perl, Python o Ruby

Características LAMP

- LAMP es una alternativa open source de calidad a otras plataformas de desarrollo web
  - ASP.NET 2.0
  - Java EE 5.0
- Su popularidad y fácil aprendizaje son argumentos para considerarla
  - El pago de licencias no siempre está ligado a la calidad de las herramientas
  - LAMP es gratuito y muy indicado para portales de tamaño medio

Ruby on Rails

- Para muchos la *killer app* de Ruby
  - http://www.rubyonrails.com/
- Es una open source web framework
- La principal razón por la que es interesante aprender Ruby

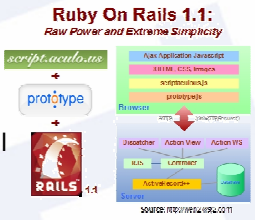
"the super productive new way to develop web applications"

Ruby On Rails

- Rails es una framework Ruby para el desarrollo de aplicaciones web que usan bases de datos
  - Creada por David Heinemeier Hansson
    - Version 1.0 en Diciembre 2005
    - Version 1.1 en Marzo 2006
- Sin duda, la Framework web más productiva
  - Hasta 10 veces más que otras frameworks, por:
    - Elegancia y simpleza de Ruby
      - Lenguaje de scripting orientado a objetos
    - El diseño de Ruby on Rails está guiado por los principios:
      - Menos software
      - Convenciones en vez de configuraciones en XML

Características Ruby on Rails

- Rails incluye soporte para:
  - AJAX (Ajax on Rails) -- http://blog.curthibbs.us/
  - Servicios Web (Action Web Service)
  - Patrón de diseño Model-View-Controller
  - Mapeo automático de objetos a modelo relacional
  - RJS (Ruby to JavaScript compiler)
- Sigue filosofía de desarrollo (DRY-COC):
  - **"Don´t Repeat Yourself"** → DRY
    - Patrón de diseño Active Record → la definición de clases no tiene que especificar los nombres de columnas
  - **"Convention Over Configuration"** → COC
    - La clase `User` en Ruby corresponde con la tabla `users`
- Otras características avanzadas: caching, validación y callbacks, transacciones, testing, generadores, seguridad
  - Su propio servidor web WEBrick o se acopla con Apache (`mod_ruby`)



Características de Ruby on Rails

- Rails puede ser visto como:
  - Un lenguaje de programación de dominio específico (Programación Web)
    - Ruby ideal para alojar DSL (Domain Specific Languages)
      - `has_many :editions` ⇔ `has_may(:editions)`
  - Una framework en la que sólo hay que escribir ficheros de configuración
    - Da la impresión de que escribes ficheros de configuración en vez de código
      - `belongs_to :previous, :class_name => "Message", :foreign_key => "previous_id"`
    - Curiosamente el fichero de configuración principal de Rails `config/database.yml` es código Ruby aunque no lo parezca

Universidad de Deusto
Deustuko Unibertsitatea

Ruby on Rails y MVC

- Rails es una framework que incluye todo lo necesario para crear aplicaciones web soportadas en BBDD de acuerdo al patrón de diseño MVC:
  - La vista o capa de presentación incluye plantillas responsables de insertar datos entre etiquetas HTML
  - El modelo contiene objetos inteligentes de dominio que encapsulan la lógica de negocio y persistencia
  - El controlador gestiona las peticiones entrantes manipulando el modelo y redirigiendo a la vista
    - "Dirige el tráfico"

Universidad de Deusto
Deustuko Unibertsitatea

ActiveRecord

- El modelo en Rails está representado por una capa de mapeo objeto-relacional denominada Active Record
  - Permite representar filas de una base de datos como objetos
    - Asociar a estos objetos métodos de negocio

Universidad de Deusto
Deustuko Unibertsitatea

ActionPack

- Las capas de control y vistas son gestionadas por Action Pack, compuesto de:
  - Action View
  - Action Controller
- Empaquetadas como un solo paquete por sus interdependencias
- El propósito de una acción en un controlador es asignar datos a variables Ruby que el código Erb (http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc/) en el fichero de vista puede interpretar y visualizar

```
@work = Work.find(params[:id])
```

Universidad de Deusto
Deustuko Unibertsitatea

Instalación Ruby on Rails

- Requirements:
  - Ruby
    - Windows Installer se puede obtener de:
      - http://rubyforge.org/frs/download.php/4174/ruby182-15.exe)
  - RubyGems, gestor de paquetes estándar de Ruby
    - http://rubyforge.org/frs/download.php/5208/rubygems-0.8.11.zip
      - Instalar con comando: `ruby setup.rb`
- Una vez instalado RubyGems, se puede instalar Rails con:
  - `gem install rails --include-dependencies`
  - `gem list --local # listas las gemas instaladas`
- Se pueden bajar instalables incluyendo todas las dependencias:
  - InstantRails (incluye Ruby, Rails, Apache y MySQL)
    - http://instantrails.rubyforge.org/wiki/wiki.pl
- Se puede extender Rails a través de numerosos plugins:
  - http://wiki.rubyonrails.org/rails/pages/Plugins

Otras Herramientas Importantes

- MySQL BBDD
  - http://dev.mysql.com/get/Downloads/MySQL-5.0/mysql-noinstall-5.0.22-win32.zip/from/pick#mirrors
- Apache2 Web Server
  - http://httpd.apache.org/download.cgi
    - http://apache.rediris.es/httpd/binaries/win32/apache_2.2.2-win32-x86-no_ssl.msi

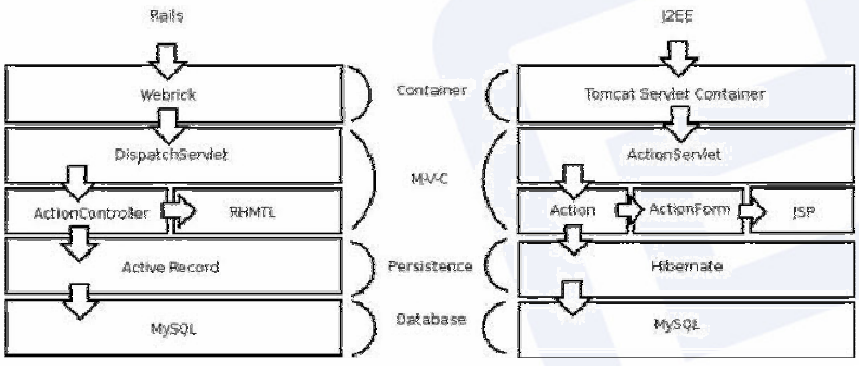
Pasos en el Desarrollo de una Aplicación "MCV" con Rails

1. Describir y modelar el dominio de la aplicación
   - Entidades y relaciones entre ellas
2. Especificar qué ocurre en ese dominio
   - Escenarios o acciones en los que participan los elementos del dominio
3. Elegir y diseñar las vistas de ese modelo
   - ¿Cómo presentar la funcionalidad al usuario?

Clases que implementa MVC en Rails

Capa MVC	Librería Rails	Propósito
Modelo	`ActiveRecord`	Establece una correspondencia entre clases Ruby y tablas en BBDD
Vista	`ActionView`	Un sistema basado en Embedded Ruby (Erb) para definir plantillas de presentación. `ActionView` y `ActionController` empaquetados juntos en `ActionPack`
Controlador	`ActionController`	Actúa de broker de datos entre `ActiveRecord` (interfaz de BBDD) y `ActionView` (motor de presentación)

Ruby on Rails vs. J2EE



Rails-friendly SQL

- El código SQL de tu BBDD debe cumplir lo siguiente:
  - Cada entidad (`PERSONA`) recibe una tabla nombrada en plural (`personas`)
  - Cada tabla representando a una entidad contiene un campo entero denominado `id`
  - Si la entidad `y` pertenece a `x`, entonces la tabla `y` contiene un campo `x_id`
  - Los campos de una tabla guardan las propiedades de una entidad

Labor de un desarrollador en Rails

- Como desarrollador Rails deberás:
  - Añadir funcionalidad a tu aplicación siguiendo unas reglas prefijadas
    - Se añadirán métodos a los controladores por cada acción soportada
  - Definir funcionalidad común en los ficheros `helper`
    - Un fichero `helper` es creado por cada controlador

```
def link_to_order(order)
   link_to(order.capitalize, :controller => “rcr”, :action
  => “all”, :params => { “order” => order })
end
```

  - Añadir funcionalidad a los modelos
    - Añadiendo métodos de `callback`

```
def before_create
  self.description = “standard” unless description
end
```

    - Añadiendo métodos de ayuda al modelo
      - Ej. Método que concatena dos campos (`nombre` y `apellido1`)

Ejemplo Método Controlador

```
def all
  @order = params[:order] || “number”
  sort_proc = case @order
    when “author” then lambda {|r|
  [r.user.name.downcase, r.number]}
    when “status”, “title” then lambda {|r|
  [r.send(@order).downcase, r.number]}
    when “number” then lambda {|r| -r.number}
  end
  @rcrs = Rcr.find(:all).sort_by &sort_proc
end
```

Desarrollando la aplicación

- Rails dicta dónde colocar el código de control, vista y modelo, dentro de `%INTALL_DIR%\<nombre-app>\apps`
  - `controllers` → contiene los controladores
  - `views` → contiene plantillas de visualización
  - `models` → clases que modelan tablas BBDD
  - `helpers` → clases de ayuda
- Para crear un controller:
  - `ruby script\generate controller MyTest`
    - Crea fichero `my_test_controller.rb` con clase `MyTestController`
    - Para ver su contenido tenemos que añadir método `index`:

```
def index
    render :text => “Hello World”
end
```

    - Podemos acceder al contenido como http://127.0.0.1:3000/My_Test/index

Generadores en Rails

- Para crear un modelo y controlador para tabla `recipes`:

```
ruby script\generate model Recipe
ruby script\generate controller Recipe
ruby script\generate controller main
  welcome
```

- Para crear el código de scaffolding con las vistas y luego poderlo modificar:

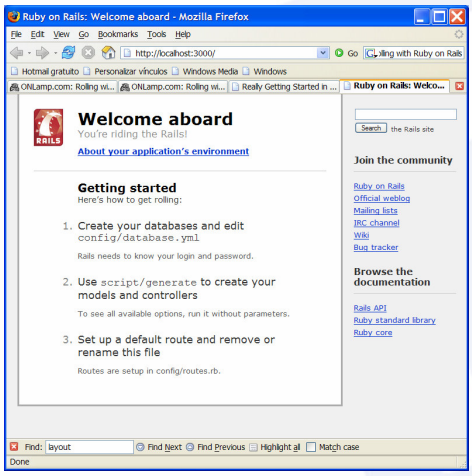
```
ruby script\generate scaffold Recipe
```

Utilidades Ruby on Rails

- El `irb` de Rails es `script/console`
- `rails` → crea el directorio de la aplicación
- Ruby Gems
  - `gem list -local` → lista módulos instalados
  - `gem list -remote` → lista módulos disponibles en repositorio de gemas
  - `gem install <program>` → instala un módulo
- `Rake`
  - `rake stats` → muestra detalles de modificaciones efectuadas al código fuente

Universidad de Deusto
Deustuko Unibertsitatea

Ruby on Rails Example



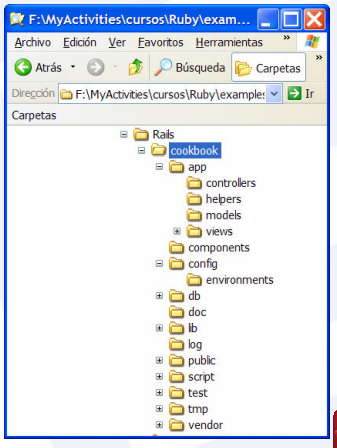
Universidad de Deusto
Deustuko Unibertsitatea

Ruby on Rails Example

- Gestor de Recetas de Cocina
  - Acciones a definir en controlador:
    - Muestra listado recetas
    - Crea y edita recetas
    - Asigna recetas a categorías
- Crea el árbol de directorios de la aplicación ejecutando en localización deseada:
  `rails cookbook`
- Prueba la aplicación:
  - `ruby script\server` (arranca WEBrick)
  - http://127.0.0.1:3000/

Estructura de Directorios de Rails

- El subdirectorio `controllers` es donde Rails ubica las clases que gestionan las peticiones de los usuarios
  - La URL de una petición se mapea a una clase controladora y a métodos dentro de ella
- El subdirectorio `views` contiene las plantillas a rellenar con datos de la aplicación y generar HTML
- El subdirectorio `models` mantiene las clases que modelan y encapsulan los datos guardados en la BBDD



Desarrollando una aplicación en 6 pasos

1. Crear estructura de directorios de la aplicación: `rails cookbook`
2. Crear una BBDD MySQL: `mysql -uroot -p < cookbook.sql`
3. Modificar el fichero `%APP_DIR%\cookbook\config\database.yml,,` indicando nombre base de datos, username y password

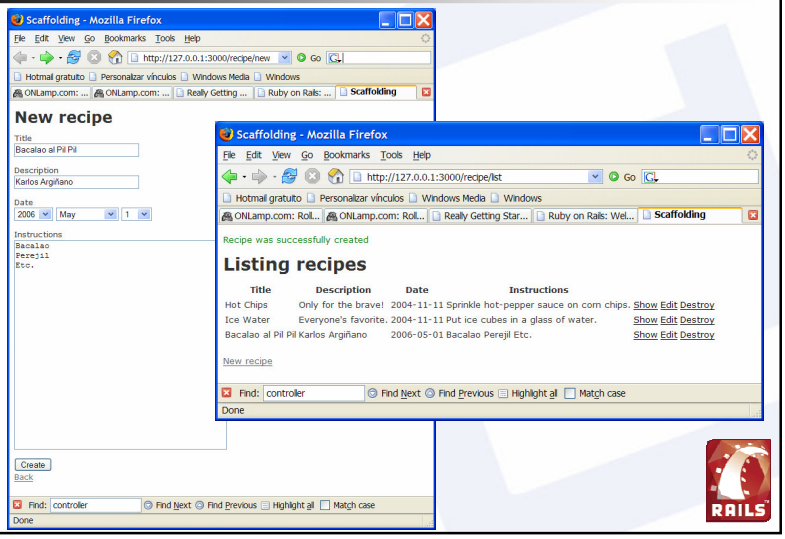
Desarrollando una aplicación en 6 pasos

3. Crear modelo `Recipe` asociado a tabla `recipes`: `ruby script\generate model Recipe`
   - Genera fichero `recipe.rb`, contiene métodos para actualizar DDBB y atributos:

     ```
     class Recipe < ActiveRecord::Base
     end
     ```

4. Crear controlador para manipular recetas con operaciones CRUD (Create, Read, Update, Delete): `ruby script\generate controller Recipe`
   - Crea fichero `recipe_controller.rb` con clase `RecipeController`
     - Añade línea `scaffold :recipe`,
       - Define acciones list, show, edit y delete
       - Vistas para cada una de las acciones
5. Arranca servidor: `ruby script\server`
   - Vete a: http://127.0.0.1:3000/recipe/new

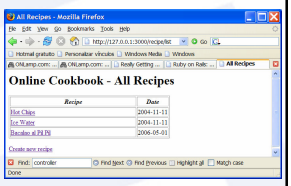
Resultado del Desarrollo



Personalizando la Aplicación

- Sobreescribir la acción `list` en `RecipeController`:

```
def index
   redirect_to :action => 'list'
end
def list
   @recipes = Recipe.find_all
end
```



- Crear `list.rhtml` en `app\views\recipe`:

```
<html><head><title>All Recipes</title></head>
<body>
<h1>Online Cookbook - All Recipes</h1>
<table border="1">
 <tr>
  <td width="80%"><p align="center"><i><b>Recipe</b></i></td>
  <td width="20%"><p align="center"><i><b>Date</b></i></td>
 </tr>
 <% @recipes.each do |recipe| %>
  <tr>
   <td><%= link_to recipe.title, :action => "show", :id => recipe.id %></td>
   <td><%= recipe.date %></td>
  </tr>
 <% end %>
</table>
<p><%= link_to "Create new recipe", :action => "new" %></p>
</body>
</html>
```

Modificando la Aplicación

- Ligando recetas a categorías:
  - `ruby script\generate controller Category`
  - `ruby script\generate model Category`
- Añadir scaffolding al controlador categoría: `scaffold :category`
- Ligando categoría y receta:
  - En clase `Recipe`: **`belongs_to :category`**
  - En clase `Categoría`: **`has_many :recipes`**
- Modificar método `edit` de recetas:

```
def edit
     @recipe = Recipe.find(@params["id"])
     @categories = Category.find_all
end
```

- Crear `edit.rhtml` para recetas

```
<select name="recipe[category_id]">
   <% @categories.each do |category| %>
      <option value="<%= category.id %>"
         <%= ' selected' if category.id == @recipe.category_id %>
         <%= category.name %>
       </option>
   <% end %>
</select>
```

- Modificar `list.rhtml` para mostrar categorías:

```
<td><%= recipe.category.name %></td>
```

Creando y Borrando Recetas

- En `recipe_controller.rb` incluir:

```
def create
   @recipe = Recipe.new(@params['recipe'])
   @recipe.date = Date.today
   if @recipe.save
      redirect_to :action => 'list'
   else
      render_action 'new'
   end
end
```

- Para borrar en `list.rhtml`:

```
<%= link_to "(delete)",
                    {:action => "delete", :id => recipe.id},
                    :confirm => "Really delete #{recipe.title}?" %>
```

- En `recipe_controller.rb` incluir método `delete`:

```
def delete
   Recipe.find(@params['id']).destroy
   redirect_to :action => 'list'
end
```

Layouts en Rails

- Layout es una metaplantilla de páginas
- Para crear layouts:
  - Introduce en el controlador de recetas y categorías: `layout "standard-layout"`
    - Indica que usen `standard-layout.rhtml` para todas las páginas a las que redirija el flujo
      - Alternativamente en `app/controllers/application.rb` colocar la sentencia `layout "nombre-fichero-plantilla.rhtml"` y coloca `"nombre-fichero-plantilla".rhtml` en `app/views/layouts`
  - Elimina las etiquetas redundantes en las vistas
  - Utiliza `@content_for_layout` en el punto donde se inserta una vista particular

Layouts en Rails

```
<html>
 <head>
   <title>Online Cookbook</title>
 </head>
 <body>
   <h1>Online Cookbook</h1>
   <%= @content_for_layout %>
   <p>
     <%= link_to "Create new recipe",
                 :controller => "recipe",
                 :action => "new" %>

   <%= link_to "Show all recipes",
               :controller => "recipe",
               :action => "list" %>

   <%= link_to "Show all categories",
               :controller => "category",
               :action => "list" %>
   </p>
 </body>
</html>
```

Configurando WEBrick

- WEBrick es un servidor web distribuido con Rails
  - A partir de la URL de entrada averigua el controlador, acción a ejecutar e `id` de la entidad
  - Las variables CGI disponibles a través del método `params` o atributo `@params`
  - Se puede guardar estado entre peticiones con `session['user'] = user.id`
- Para traducir una URL vacía en una llamada a la acción `welcome` de un controlador `main`:
  - Modificar el fichero `config/routes.rb`
  - Añadir la línea:

    ```
    map.connect '', :controller => "main", :action =>
      welcome
    ```
  - Borrar o renombrar la página por defecto: `public/index.html`
- `tail -f log/development.log`, te permite monitorizar qué está ocurriendo con el servidor en todo momento

Más sobre el Modelo

- Una clase modelo en Rails tiene capacidades provenientes de:
  - Herencia de la clase `ActiveRecord::Base`
  - Creación automática de *accesors* y otros métodos en base a campos de tablas
  - Creación semi-automática a partir de asociaciones
  - Añadidos de instancias de métodos
- Métodos importantes `ActiveRecord`:
  - Métodos de instancia:
    - `save`
    - `update`
    - `destroy`
  - Métodos de clase:
    - `new`
    - `create` (`new` + `save`)
    - `find`
    - `delete` (`find` + `destroy`)
    - find
    - find_by_*fieldname*

Más sobre el Modelo

- Tipos de asociaciones:
  - `has_many :works`
  - `belongs_to :composer`
  - `has_one :works`
  - `has_and_belongs_to_many :instruments`
- Mejoras:
  - Soft → devuelven instancias de colecciones en base a datos existentes

```
def editions
  Edition.find_by_sql("SELECT edition_id from edition_works“)
end
def open_orders
   orders.find(:all, :conditions => "status = 'open'")
end
```

  - Hard → producen nuevas representaciones en nuevas estructuras de datos

```
def nice_opus
   if /^\d/.match(opus)
      "op. #{opus}"
   else
      opus
   end
end
```

Más sobre ActionPack

- Un partial es un fragmento reusable de plantilla colocado en un fichero de nombre `_<nombre_partial>.rhtml`:

```
<%= render :partial => "editions" %>
```

- Aspectos comunes en muchas aplicaciones son la redirección de peticiones y el almacenamiento de datos en sesión:

```
redirect_to :action => "show", :id => 5
session['customer'] = c.id
```

- Para hacer que un controller use métodos helper, no declarados en `application_helper.rb` o en su propio helper, usar:
  - `helper :composer`
- Algunos métodos helper interesantes son:
  - `form_tag, text_field, password_field`

Filtros y Páginas de Error

- Filtros definidos en clase `ApplicationController` de fichero `application.rb`, son aplicables a cualquier acción:

```
before_filter :get_customer
def get_customer
   @c = Customer.find(session['customer'])
end
```

- Otro ejemplo que redirecciona a página de error:

```
before_filter :authorize, :except => ["signup", "login"]
def authorize
   return true if @c
   report_error("Unauthorized access; password required")
end
```

- Definiríamos `report_error` en `ApplicationController` en el fichero `application.rb`:

```
def report_error(message)
   @message = message
   render("main/error")
   return false
end
```

- Donde `app/views/main/error.rhtml` contendría:

```
<% @page_title = "Error" %>
<% @message %>
```

Ejercicio

- Hacer que solamente se muestren las recetas en una categoría
  - Recuperar parámetro categoría de url:

    ```
    @category = @params['category']
    ```
  - Modificar `list.rhtml` con:
    - `<% if (@category == nil) || (@category == recipe.category.name)%>`
    - `<%= link_to recipe.category.name, :action => "list", :category => "#{recipe.category.name}" %>`

Convenciones en Rails

- Un nombre de clase singular (`Recipe`) mapea a un plural de bases de datos (`recipes`)
  - Requiere que la clave se denomine `id` y sea auto-incrementable
  - La clave extranjera tiene el formato `*singularOfForeignTableName*_id`
- Toda acción en un controlador (`list`) tiene asociada una plantilla rhtml (`list.rhtml`)
- Si hay un modelo denominado `persona` tiene que haber un `persona_controller`
  - Es posible tener un controlador, sin que tenga asociado un modelo
- La línea de código `scaffold :recipe` nos permite trabajar con nuestro modelo de datos, creando las acciones `list`, `show`, `edit` y `delete` y sus vistas correspondientes
  - Puedes ver el código generado con: `ruby script\generate scaffold Recipe`

Universidad de Deusto
Deustuko Unibertsitatea

Ajax

- AJAX (Asynchronous Javascript and XML), técnica de desarrollo que genera aplicaciones web más interactivas combinando:
  - XHTML y CSS para la presentación de información
  - Document Object Model (DOM) para visualizar dinámicamente e interactuar con la información presentada
  - XML, XSLT para intercambiar y manipular datos
    - JSON y JSON-RPC pueden ser alternativas a XML/XSLT
  - `XMLHttpRequest` para recuperar datos asíncronamente
  - Javascript como nexo de unión de todas estas tecnologías
- Características:
  - Aplicaciones son más interactivas al estilo desktop
  - Reduce tamaño de la información intercambiada
  - Libera de procesamiento a la parte servidora???
  - Actualiza porciones de la página en vez de la página completa
  - Necesario asegurar aplicación AJAX funciona en todo navegador

Universidad de Deusto
Deustuko Unibertsitatea

Ajax on Rails

- Rails hace muy sencillo el uso de Ajax
  - No es necesario ni escribir código JavaScript que haga uso del objeto `XMLHttpRequest`
- Operación `link_to_remote` → actualiza un elemento de la página con el resultado de una invocación a `XMLHttpRequest`
- Uso:
  - Modificar un fichero `.rhtml` en el que se incluyan las sentencias:
    - `<%= javascript_include_tag "prototype" %>`
    - `<%= link_to_remote( “<label>", :update => “<dom-element-id>", :url =>{ :action => :<server-method> }) %>`
  - Añadir `server-method` en el controlador

Ejemplo link_to_remote

- En /views/demo/index.rhtml:

```
<html>
  <head>
    <title>Ajax Demo</title>
    <%= javascript_include_tag "prototype" %>
  </head>
  <body>
    <h1>What time is it?</h1>
    <div id="time_div">
      I don't have the time, but
      <%= link_to_remote( "click here",
                          :update => "time_div",
                          :url => { :action => :say_when },
                          :position => "after" ) %>
      and I will look it up.
    </div>
  </body>
</html>
```

- En /controllers/demo_controller.rb:

```
class DemoController < ApplicationController
   def index
   end
   def say_when
      render_text "<p>The time is <b>" + DateTime.now.to_s + "</b></p>"
   end
end
```

Ejemplo form_remote_tag

- Funciona de manera similar pero permite enviar datos de un formulario
- Uso:
  - Modificar el .rhtml:

```
<html>
  <head>
    <title>Ajax List Demo</title>
    <%= javascript_include_tag "prototype" %>
  </head>
  <body>
    <h3>Add to list using Ajax</h3>
    <%= form_remote_tag(:update => "my_list",
                        :url => { :action => :add_item },
                        :position => "top" ) %>
      New item text:
      <%= text_field_tag :newitem %>
      <%= submit_tag "Add item with Ajax" %>
    <%= end_form_tag %>
    <ul id="my_list">
      <li>Original item... please add more!</li>
    </ul>
  </body>
</html>
```

  - Modificar el controlador:

```
class ListdemoController < ApplicationController
   def index
   end
   def add_item
         render_text "<li>" + params[:newitem] + "</li>"
   end
end
```

Ejemplo observe_field

- También podemos escribir aplicaciones muy dinámicas que monitoricen el cambio en un campo de un formulario (observer)
- Ejemplo:

```
<label for="searchtext">Live Search:</label>
<%= text_field_tag :searchtext %>
<%= observe_field(:searchtext,
                  :frequency => 0.25,
                  :update => :search_hits,
                  :url => { :action =>
  :live_search }) %>
<p>Search Results:</p>
<div id="search_hits"></div>
```

Desarrollo de un Weblog

- Desarrollo de un simple blog en Rails que permita añadir comentarios y una interfaz de administración con métodos CRUD
- Tablas de partida:

```
mysql> desc comments;
+---------+---------+------+-----+---------+----------------+
| Field   | Type    | Null | Key | Default | Extra          |
+---------+---------+------+-----+---------+----------------+
| id      | int(11) | NO   | PRI | NULL    | auto_increment |
| body    | text    | YES  |     | NULL    |                |
| post_id | int(11) | NO   |     | 0       |                |
+---------+---------+------+-----+---------+----------------+
mysql> desc posts;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| title      | varchar(255) | YES  |     | NULL    |                |
| created_at | date         | YES  |     | NULL    |                |
| body       | text         | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

Desarrollo de un Weblog

1. Generar estructura de la aplicación: `rails myweblog`
2. Arrancar el servidor: `ruby script/server`
3. Comprobar que la aplicación está instalada: http://localhost:3000
4. Mirar la estructura de directorios creadas
   - `controllers/application.rb`
   - `helpers/application_helper.rb`
5. Generar el controlador de la aplicación: `ruby script/generate controller Blog`
   - Comprobar `controllers/blog_controller.rb`
6. Ir a http://localhost:3000/blog, como no funciona, definir un controlador:

```
def index
   render :text => "Kaixo mundua"
End
```

7. Crear `views\blog\index.rhtml` con contenido `"hola desde la plantilla"`
8. Reconfigurar la base de datos, denominada weblog en `database.yml`
   - Poner nombre BBDD, nombre de usuario y contraseña a `weblog`
   - Poner `production: development`
9. Crear el modelo: `ruby script/generate/ model Post`
   - Comprobar fichero generado `model/post.rb`

Desarrollo de un Weblog

10. Escribir en `BlogController`, la línea `scaffold:post`
11. Añadir a `post.rb`, la línea `validates_presence_of :title`
12. Ejecutar `ruby script/generate scaffold Post Blog`
    - Ver todo lo que se ha generado
13. Modificar `views\blog\list.rhtml` a:

```
<h1>My wonderful weblog</h1>
<% for post in @posts %>
   <div>
      <h2><%= link_to post.title, :action => 'show', :id => post %>
      <p><%= post.body %></p>    <p>
      <small>
         <%= post.created_at.to_s(:long) %>
         (<%= link_to 'Edit', :action => 'edit', :id => post %>)
      </small></p>
   </div><% end %>  <%= link_to 'New post', :action => 'new' %>
```

    - Aplicar método `reverse` a la lista
14. Modificar `blog_controller.rb` para colocar en list `@posts = Post.find(:all)`
15. Tomar como base `list.rhtml` y crear `_post.rhtml` con el contenido:
    - `<h1>My wonderful weblog</h1> <% for post in @posts.reverse %><% end %>  <%= link_to 'New post', :action => 'new' %>`
16. Añadir `<%= render :partial => "post", :collection => @posts %>` y `<%= render :partial => "post", :collection => [@post] %>` en `show.rhtml` y `list.rhtml`

Desarrollo de un Weblog

17. Generar el modelo `ruby script/generate model Comment`
18. Añadir `belongs_to :post` a clase modelo `Comment`
19. `has_many :comments` a `Post`
20. Modificar `show.rhtml` para que visualice comentarios:

```
<%= render :partial => "post", :object => @post %>
<%= link_to 'Edit', :action => 'edit', :id => @post %> |
<%= link_to 'Back', :action => 'list' %>
<h2>Comments</h2>
<% for comment in @post.comments %>
   <%= comment.body %>   <hr/>
<% end %>
<%= form_tag :action => "comment" %>
   <%= text_area "comment", "body" %><br/>
   <%= submit_tag "Comment!" %>
</form>
```

21. Añadir a `BlogController` el método `comment`:

```
def comment
   Post.find(params[:id]).comments.create(params[:comment])
   flash[:notice] = "Added your comment"
   redirect_to :action => "show", :id => params[:id]
end
```

Desarrollo de un Weblog

21. ¿Cuánto código hemos escrito?
    - `rake stats`
22. ¿Qué comandos SQL se están generando?
    - `tail –f log/development.log`
23. Aplicar tests de unidad a la aplicación:
    - `rake test_units`
      - Directorio `test/unit/*.rb`
24. Interactuar con la aplicación y depurar:
    - `ruby /script/console`

```
p=Post.find :first
p.title = Vitoria
p.save
p.comment.create :body => “Provincia de Álava“
p.destroy
```

Integración Ruby y Apache

- Instalación FastCGI+Apache en Linux:
  - http://wiki.rubyonrails.com/rails/pages/FastCGI
- Instalación FastCGI+Apache en Windows:
  - Instalar Apache2
  - Instalar MySQL
  - Instalar la última versión de Ruby
  - Instalar Ruby for Apache (http://rubyforge.org/projects/rubyforapache/)
  - Comprobar:
    - `mod_fastcgi.so` está en `Apache2/modules`
    - `msyql.so` y `fcgi.so` en `ruby\lib\ruby\site_ruby\1.8\i386-msvcrt`
  - Añadir a `Apache2/conf/httpd.conf`:

```
LoadModule fastcgi_module modules/mod_fastcgi.so
<IfModule mod_fastcgi.c>
  AddHandler fastcgi-script .fcgi
</IfModule>
```

  - Más información en: http://dema.ruby.com.br/articles/2005/08/23/taming-fastcgi-apache2-on-windows
- Distribución de aplicaciones Rails con RubyScript2Exe: http://www.erikveen.dds.nl/distributingrubyapplications/rails.html

Características Avanzadas Ruby

- Caching
  - http://ap.rubyonrails.com/classes/ActionController/Caching.html
- Validación
  - Ejemplo:

```
class Entry < ActiveRecord::Base
   # Relationships
   belongs_to :bliki
   # Validation
   validates_length_of :name, :within => 6..100
   validates_uniqueness_of :name
   validates_format_of :name, :with => /^\w+$/,
   :message => "cannot contain whitespace"
   validates_length_of :content, :minimum => 10
   validates_associated :bliki
end
```

  - http://wiki.rubyonrails.com/rails/pages/HowtoValidate
- ActiveRecord callbacks
  - http://api.rubyonrails.org/classes/ActiveRecord/Callbacks.html
- ActiveRecord transaction:

```
transaction do
  david.withdrawal(100)
  mary.deposit(100)
end
```

- Otras características: Testeo, Seguridad y Generadores

Historias de Éxito de Ruby on Rails

- A pesar de haber nacido a mediados de 2004:
  - BaseCamp
  - 43 Things
  - Instiki – software wiki
  - Typo – software para weblog
  - 37 signals (http://www.37signals.com/)

Conclusión

- Ruby on Rails permite la construcción de sofisticadas aplicaciones web CRUD en un tiempo record
  - Incorpora además funcionalidad que permite la realización de aplicaciones profesionales:
    - Cacheo
    - Transacciones
    - AJAX
- Considerada por algunos como "la plataforma" de desarrollo de Web 2.0:
  - http://web2.wsj2.com/ruby_on_rails_11_web_20_on_rocket_fuel.htm

Referencias

- Ruby
  - Programming Ruby
    - http://www.rubycentral.com/book/
  - Learning Ruby
    - http://www.math.umd.edu/~dcarrera/ruby/0.3/index.html
  - Ruby, an Introduction
    - http://www.io.com/~jimm/writing/rubytalk/talk.html
  - Ruby Application Archive
    - http://raa.ruby-lang.org/
  - ruby-doc.org – Ruby Documentation Project
    - http://www.ruby-doc.org/
  - Ruby Central – The source for Ruby
    - http://www.rubycentral.com
  - Ruby Online Reference
    - http://www.rubycentral.com/book/builtins.html
  - Ruby Standard Library Reference
    - http://www.ruby-doc.org/stdlib/

Referencias

- Tutorial Ruby on Rails:
  - Rolling with Ruby on Rails
    - http://www.onlamp.com/lpt/a/5546
  - Rolling with Ruby on Rails, Part 2
    - http://www.onlamp.com/lpt/a/5641
- Ajax on Rails
  - http://www.onlamp.com/pub/a/onlamp/2005/06/09/rails_ajax.html
- Ruby on Rails API
  - http://www.rubyonrails.org/api/