Pauta I1-2015-1.md

Interrogación 1

> 13 de Abril del 2015

Tabla de contenidos

Pregunta 1

Solución:

Todas las lineas que incluyan un método para imprimir como `puts` o `print`, son tan válidas como simplemente retornar el *string*.

a) (2.0pts) Clase `Person`

```ruby
class Person
  attr_accessor :name, :age  # Necesitamos esto para poder acceder a los atributos públicamente

  def initialize(name, age)
    @name = name
    @age = age
  end

  def intro
    puts "hello, my name is #{@name} and I am #{@age} years old"
  end
end

person1 = Person.new('Pepe', 30)
person1.intro
# -> hello, my name is Pepe and I am 30 years old
```

Puntaje:

- (0.5 ptos) Por utilizar correcta sintaxis de Ruby.
- (0.5 ptos) Por definir bien las propiedades `name` y `age`.
- (0.5 ptos) Por utilizar `attr_accessor`.
- (0.5 ptos) Por correcta implementación del método `intro`.

b) (2.0pts) Clase `Team`

```ruby
class Team
  def initialize(name)
    @name = name
    @members = []
  end

  def add_member(name, age)
    @members << Person.new(name, age)
```

```ruby
  end

  def print_team
    puts @name
    puts '========='
    @members.each { |member| puts "#{member.name}, #{member.age} yeras old" }
  end
end

team = Team.new('The Killers')
team.add_member('John', 20)
team.add_member('Bob', 29)
team.print_team
# -> The Killers
#    =========
#    John, 20 yeras old
#    Bob, 29 yeras old
```

Puntaje:

- (0.5 ptos) Por utilizar correcta sintaxis de Ruby.
- (0.5 ptos) Por definir bien las propiedades `name` y `members`.
- (0.5 ptos) Por correcta implementación del método `add_member`.
- (0.5 ptos) Por correcta implementación del método `print_team` y buen uso del iterador.

c) (2.0pts) Bloques

Forma válida

```ruby
def describe(name, age)
  yield(name, age)
end

describe('Bob', 60) do |name, age|
  puts "#{name} is a #{age > 50 ? 'mature' : 'young'} #{age} years old person"  # bloque 1
end
# -> Bob is a mature 60 years old person

describe('Fred', 51) do |name, age|
  puts "At #{age} #{name} is considered a #{age > 50 ? 'mature' : 'young'} person"  # bloque 2
end
# -> At 51 Fred is considered a mature person
```

Forma más elegante y formal

```ruby
def describe(name, age, &block)
  person = Person.new(name, age)
  madurity = if (age > 50) then "mature" else "young" end
  block.call(person, madurity)
end

bloque1 = proc do |person, madurity|
  puts "#{person.name} is a #{madurity} #{person.age} years old person"
end

bloque2 = proc do |person, madurity|
  puts "At #{person.age} #{person.name} is considered a #{madurity} person"
end

describe('Bob', 60, &bloque1)
# -> Bob is a mature 60 years old person
describe('Fred', 51, &bloque1)
# -> Fred is a mature 51 years old person

describe('Bob', 60, &bloque2)
# -> At 60 Bob is considered a mature person
```

```
describe('Fred', 51, &bloque2)
# -> At 51 Fred is considered a mature person
```

Puntaje: (ambas formas son válidas y sin descuento).

- (0.5 ptos) Por utilizar correcta sintaxis de Ruby.
- (0.5 ptos) Por definir bien el método `describe` y que este llame correctamente al bloque.
- (0.5 ptos) Por adjuntar correctamente un bloque al método `describe`.
- (0.5 ptos) Por correcta implementación del *bloque1* y *bloque2*.

Pregunta 2

Queremos que la aplicación responda a dos URLs:

- `https://localhost:3000/welcome/short` -> Despliega una página con un `h1: hello`
- `https://localhost:3000/welcome/long` -> Despliega una página con un `h1: hello, nice to see you`

Solución:

a) (4.0pts)

Primero definiremos las actions `short` y `long` en el controlador anteriormente creado:

```
# path: app/controllers/welcome_controller.rb

class WelcomeController < ApplicationController
  def short
    @message = "hello"
    render 'short'
  end
  def long
    @message = "hello, nice to see you"
    render 'long'
  end
end
```

Luego crearemos las vistas:

```
<!-- path: /views/welcome/short.html.erb -->
<!-- path: /views/welcome/long.html.erb -->

<html>
  <head></head>
  <body>
    <h1> @message </h1>
  </body>
</html>
```

Finalmente modificamos el archivo de rutas

```
# path: routes.rb
get 'welcome/short', to: 'welcome#short'
get 'welcome/long', to: 'welcome#long'
```

Puntaje:

- (1 pto) Por modificar el controlador correctamente (*path* del controller, actions `short` y `long` con mensaje).
- (1 pto) Por crear las vistas. Se aceptó sólo crear una vista y rutear ambas actions a esta.

- (1 pto) Por modificar correctamente las rutas.
- (1 pto) Por utilizar correcta sintaxis de Ruby.

b) (2.0pts)

Dependiendo de cómo se hizo la primera parte, varias respuestas fueron aceptadas.

Cambiaríamos lo escrito en el archivo de rutas

```
# path: routes.rb
get 'hola/simple', to 'welcome#short'
get 'hola/formal', to 'welcome#long'
```

Puntaje:

- (1 pto) Por cambiar correctamente el archivo de rutas.
- (1 pto) Por utilizar correcta sintaxis de Ruby.

Pregunta 3

Solución:

a) (2.0pts) `Entidades`

```
class Product < ActiveRecord::Base
  has_many :line_items
  has_many :orders, through: :line_items
end

class Order < ActiveRecord::Base
  has_many :line_items, dependent: :destroy
end

class LineItem < ActiveRecord::Base
  belongs_to :order
  belongs_to :product
  belongs_to :cart
end

class Cart < ActiveRecord::Base
  has_many :line_items, dependent: :destroy
end
```

Puntaje:

- (0.5 ptos) Por utilizar correcta sintaxis de Ruby.
- (0.5 ptos) Utiliza bien los `belongs_to` y `has_many`.
- (1.0 ptos) Establece correctamente las relaciones (0.25 cada una)

b) (2.0pts) Tabla order

```
<h1>Order</h1>

<table>
  <thead>
  <tr>
    <th>Order <%= o.number %></th>
  </tr>
    <tr>
      <th>Qty</th>
      <th>Product</th>
      <th>Price</th>
```

```
      <th>Total</th>
    </tr>
  </thead>
  <tbody>
    <% @o.line_items.each do |li| %>
      <tr>
        <td><%= li.quantity %></td>
        <td><%= li.product.name %></td>
        <td><%= li.product.price %></td>
        <td><%= li.total_price %></td>
      </tr>
    <% end %>
    <tr>
      <td colspan = "2"></td>
      <td>Total</td>
      <td><%= o.total %></td>
    </tr>
  </tbody>
</table>
```

Podían asumir que había un método total en order implementado, pero también se considera correcto si calcularon el total y luego lo imprimieron

Puntaje:

- (0.5 ptos) Por utilizar correcta sintaxis de *erb*.
- (0.5 ptos) Por estructurar bien la tabla
- (1.0 ptos) Por definir bien todos los atributos que se deben mostrar

c) (2.0pts) Creación objetos

```
c = Cart.new()
for i in 1..26
  p = Product.new()
  p.name = i.chr
  p.price = i
  li = LineItem.new()
  li.product = p
  li.qty = 2
  c.push(li)
end

c.line_items.each do |it|
  puts it.name + " " + it.price
end
```

Puntaje:

- (0.5 ptos) Por utilizar correcta sintaxis de Ruby.
- (1.0 ptos) Crear correctamente los objetos pedidos (iterando)
- (0.5 ptos) Entrega correctamente los valores pedidos