**Programación Orientada por Objetos en Ruby**

La Programación Orientada por Objetos, también conocida por las siglas **OOP (Object Oriented Programming)**, surgió a principios de los años 80’s como respuesta a la dificultad que estaban experimentando los programadores manteniendo aplicaciones de tamaño y complejidad considerables. Cualquier cambio desencadenaba una reacción en cadena de errores ocasionados por las dependencias que existían en la aplicación. Hasta ese entonces se usaba un paradigma llamado [Programación Procedimental](http://es.wikipedia.org/wiki/Programaci%C3%B3n_por_procedimientos).

## Clases y Objetos

Las clases y los objetos son los conceptos más importantes de la Programación Orientada por Objetos, y están fuertemente relacionados. **Los objetos se crean a partir de clases, y las clases definen los atributos y el comportamiento que tendrán los objetos**. A los objetos también se les llama instancias de clase.

Veamos cómo definir una clase y crear objetos en Ruby:

class Humano

end

juan = Humano.new

pedro = Humano.new

ana = Humano.new

En este ejemplo, hemos definido una clase llamada Humano y creamos tres objetos usando Humano.new, que almacenamos en las variables juan, pedro, y ana. La clase aún no tiene atributos ni comportamiento.

Ahora modifiquemos la clase **Humano** para definir algunos atributos (nombre, edad y género), y un comportamiento (saludar):

class Humano

attr\_accessor :name, :age, :genre

def greet

puts "Hola, me llamo #{@name}"

end

end

attr\_accessor determina que los atributos se van a poder leer y escribir desde cualquier parte de la aplicación. Otras opciones son attr\_reader (solo lectura) y attr\_writer (solo escritura). A esto se le llama **la visibilidad** del atributo y corresponde al concepto de **encapsulamiento** que veremos más adelante.

La clase define un comportamiento, greet (saludar), que no es más que una función dentro de la clase. **A las funciones que están definidias en clases se les llama métodos**. Los métodos pueden usar los atributos del objeto usando el prefijo @ como lo hace el método greet.

Para leer y escribir los atributos de un objeto, o llamar sus métodos, se usa la notación objeto.atributo u objeto.método como se muestra en el siguiente ejemplo:

juan = Humano.new

juan.name = "Juan Perez"

juan.age = 12

juan.genre = "M"

puts juan.name # imprime Juan Perez

puts juan.age # imprime 12

puts juan.genre # imprime M

puts juan.greet # imprime Hola, me llamo Juan Perez

## Inicializando un objeto

Al ejecutar la línea Humano.new, Ruby busca un método llamado initialize que, si existe, se invoca automáticamente. A este método se le llama **el constructor**.

class Humano

attr\_accessor :name, :age, :genre

def initialize

puts "Inicializando humano ..."

end

end

En este ejemplo, cada vez que invoquemos Humano.new, se imprimirá la cadena “Inicializando Humano”.

El método initialize puede recibir argumentos, que nos pueden servir para inicializar los atributos del objeto.

class Humano

attr\_accessor :name, :age, :genre

def initialize(name, age, genre)

@name = name

@age = age

@genre = genre

end

end

El constructor ahora recibe tres argumentos que se utilizan para inicializar los atributos del objeto:

juan = Humano.new("Juan Perez", 12, "M")

## En Ruby (casi) todo es un objeto

Por ejemplo, la clase String de la que se crean las cadenas (strings), tienen comportamiento (métodos) que podemos llamar con la notación objeto.método:

puts "hola".capitalize # HOLA

puts "hola".reverse # aloh

En este caso, la clase String define los métodos capitalize y reverse. Puedes encontrar todos los métodos que se pueden llamar sobre las cadenas (strings) en la [documentación de Ruby](http://ruby-doc.org/core-2.2.0/String.html).

Los enteros también tienen comportamiento:

puts 2.even? # true

puts 1.next # 2

Todos los objetos tienen un método llamado class que nos devuelve la clase de la cuál fue creada el objeto:

puts juan.class # Humano

puts "hola".class # String

puts 1.class # Fixnum

## Encapsulamiento

Uno de los mayores problemas de la Programación Procedimental (el paradigma previo a OOP) es que no existía una forma de ocultar información y procedimientos del resto de la aplicación. A medida que la aplicación cambiaba, se empezaban a crear dependencias a partes del código que podían ser muy inestables, y que no estaban diseñadas para que el resto de la aplicación las usara directamente. Esas dependencias hacían muy difícil mantener el código, cualquier cambio dispara varios errores en cadena.