

Unidad 7

7.1) Programación orientada en objeto

Temas: Antecedentes de la programación orientada al objeto. Características de la programación orientada al objeto, reusable, mantenible.

Unidad 7

7.1) Programación orientada en objeto

Temas: Antecedentes de la programación orientada al objeto. Características de la programación orientada al objeto, reusable, mantenible.

Antecedentes

La programación orientada al objeto es un paradigma de programación, así como existe también la programación estructurada o procedural.

En los inicios de la programación se programó tarjetas perforadas y luego código assembler, pero era demasiado difícil, aparece el concepto de alto nivel y bajo nivel, en el caso de assembler es de bajo nivel, más cercano a la máquina que al usuario.

A principios de los años 70 en el PARC, Palo Alto Research Center de XEROX un investigador llamado **Alan Kay** desarrolló el lenguaje **Smalltalk**, el primer lenguaje orientado al objeto, el código era más parecido al inglés y por lo tanto de más alto nivel, lo que podría eventualmente hacerlo más fácil de comprender y por tanto, extender su uso.

Permite además ser reutilizado al contener clases que pueden ser públicas y de acceso a otros documentos, facilitando su utilización y permitiendo una mantención del código al tener una estructura definida.

Unidad 7

7.2) El principio de abstracción

Temas: Atributos y métodos. Ejemplos de la vida cotidiana. todos. Ejemplos de la vida cotidiana.

Modelamiento de objetos.

Abstracción

Modelamiento de objetos.

La POO supone familiarizarnos con un concepto que es la Clase u objeto, todo es un objeto: un vehículo, una taza, incluso una persona, en el contexto de programación evidentemente.

Un objeto tiene atributos, en el caso de una persona la cantidad de manos, dedos, color de ojos, color de pelo. En el caso de un vehículo la cantidad de ruedas, el color, cantidad de puertas, eso es un atributo, es decir, una característica del objeto.

Por otro lado nos encontramos con los métodos, son las acciones que puede realizar, una persona puede mirar, comer, estudiar, dormir. Un vehículo puede avanzar, retroceder, doblar y detenerse.

Entonces una clase u objeto, de manera básica tiene atributos y métodos.


¿Cómo se define una clase en Ruby







¿Cómo funciona una clase en Ruby?

Podríamos pensar que una clase es como un molde, donde vaciamos un contenido y siempre nos entrega el mismo objeto.

Pensemos en una radio, podríamos querer saber

Una clase inicializa sus variables en un constructor, esas variables serán globales, puedes agregar todos los atributos, que luego pueden ser usados por los métodos.

▼  robjetos

-  1class.rb
-  1radio.rb
-  2claseyobjeto.rb
-  3constructormetodo.rb
-  5gettersetter.rb
-  6gettersetter.rb
-  7numgettersetter.rb
-  8numgettersetter.rb
-  9privado.rb
-  10herencia.rb

```
1  class Radio
2
3
4  end
5
```

▼ robjetos

- 1class.rb
- 1radio.rb
- 2claseyobjeto.rb
- 2radio.rb**
- 3constructormetodo.rb
- 5gettersetter.rb
- 6gettersetter.rb
- 7numgettersetter.rb
- 8numgettersetter.rb
- 9privado.rb
- 10herencia.rb

```
1 class Radio
2   # constructor method
3   def initialize(nombre,dial, transmitiendo)
4
5     #@nombre, @dial = nombre, dial
6
7     @nombre = nombre
8     @dial = dial
9     @transmitiendo = transmitiendo
10
11  end
12  # accessor se accederá a estos métodos
13
14  def printNombre
15    @nombre
16  end
17
18  def printDial
19    @dial
20  end
21
22  def printTransmitiendo
23    @transmitiendo
24  end
25
26  end
27  # creación de un objeto a partir de la clase
28  radio1 = Radio.new("Romántica", "101.5", "Sí")
29  n = radio1.printNombre()
30  d = radio1.printDial()
31  o = radio1.printTransmitiendo()
32
33  puts "El nombre de radio1 es #{n}"
34  puts "La dial del radio1s es #{d}"
35  puts "¿Está la radio transmitiendo? #{o}"
36
```


Inicialización o constructor

```
class Radio

  def initialize(nombre,dial, transmitiendo)

    @nombre = nombre
    @dial = dial
    @transmitiendo = transmitiendo

  end
```

Métodos

```
def printNombre  
  @nombre  
end
```

```
def printDial  
  @dial  
end
```

```
def printTransmitiendo  
  @transmitiendo  
end
```

Objeto

Una clase siempre produce un objeto

```
radio1 = Radio.new("Romántica", "101.5", "Sí")
```

Atributos y métodos

Los atributos siempre serán variables y los métodos siempre serán las acciones.

▼ robjetos

- 1class.rb
- 1radio.rb
- 2claseyobjeto.rb
- 2radio.rb**
- 3constructormetodo.rb
- 5gettersetter.rb
- 6gettersetter.rb
- 7numgettersetter.rb
- 8numgettersetter.rb
- 9privado.rb
- 10herencia.rb

```
1 class Radio
2   # constructor method
3   def initialize(nombre,dial, transmitiendo)
4
5     #@nombre, @dial = nombre, dial
6
7     @nombre = nombre
8     @dial = dial
9     @transmitiendo = transmitiendo
10
11  end
12  # accessor se accederá a estos métodos
13
14  def printNombre
15    @nombre
16  end
17
18  def printDial
19    @dial
20  end
21
22  def printTransmitiendo
23    @transmitiendo
24  end
25
26  end
27  # creación de un objeto a partir de la clase
28  radio1 = Radio.new("Romántica", "101.5", "Sí")
29  n = radio1.printNombre()
30  d = radio1.printDial()
31  o = radio1.printTransmitiendo()
32
33  puts "El nombre de radio1 es #{n}"
34  puts "La dial del radio1s es #{d}"
35  puts "¿Está la radio transmitiendo? #{o}"
36
```

Unidad 7

7.3) El principio de encapsulación

Temas: Getters y setters.

Get y seter

Veamos otro ejemplo, con un estudiante, esta vez implementaremos los get y los sets, permiten setear o configurar métodos y acceder a ellos, de ahí get y sets.

Te hemos dejado el ejemplo en el manual del participante, los identificarás así

```
def getEdad
```

```
  @edad
```

```
end
```

```
def setEdad(valor)
```

```
  @edad= edad
```

```
end
```

Utilización

```
estudiante.setNombre("Carlos Zamorano")  
estudiante.setCarrera("Arquitectura")  
estudiante.setEdad(22)  
# usa los métodos del accesor, accedemos a los métodos  
n = estudiante.getNombre()  
c = estudiante.getCarrera()  
e = estudiante.getEdad()
```


Unidad 7

7.4) Estados de un objeto

Temas: Estado de un objeto

Estado de un objeto

Un objeto tiene estados(atributos) y comportamientos(métodos).

Todos los objetos que nos rodean tienen estados y comportamiento, una cocina tendrá estados: cantidad de quemadores, ruedas para graduar el calor del quemador, dimensiones y tendrá también comportamiento (encender, apagar, calentar, enfriarse).

Unidad 7

7.5) Polimorfismo y duck typing

Temas: Módulos y mixins. Herencia y polimorfismo.

Módulo

Un módulo es similar a una clase, con la diferencia que no se inicializan atributos, no tiene constructor, más bien cada método contiene sus propios atributos, sin embargo, si tienen instancias de los métodos

Pensemos en un blog, debería tener título, subtítulo y contenido.

▼ robjetos

- 1class.rb
- 1radio.rb
- 2claseyobjeto.rb
- 2radio.rb
- 3constructormetodo.rb
- 5gettersetter.rb
- 6gettersetter.rb
- 7numgettersetter.rb
- 8numgettersetter.rb
- 9privado.rb
- 10herencia.rb
- modulo.rb

```
1 module Curso
```

```
2
```

```
3
```

```
4
```

```
5   def Curso.titulo
```

```
6     puts "Bienvenid@s al curso de Ruby"
```

```
7   end
```

```
8
```

```
9
```

```
10  def Curso.subtitulo
```

```
11    puts "Llegó el momento de aprender!"
```

```
12  end
```

```
13
```

```
14
```

```
15  def Curso.contenido
```

```
16    puts "Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of  
17    (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of
```

```
17  end
```

```
18
```

```
19 end
```

```
20
```

```
21
```

```
22
```

```
23
```

```
24 Curso.titulo
```

```
25 Curso.subtitulo
```

```
26 Curso.contenido
```

```
27
```

Mixin

Los módulos se pueden combinar, al interior de una clase.

▼ robjetos

- 1class.rb
- 1radio.rb
- 2claseyobjeto.rb
- 2radio.rb
- 3constructormetodo.rb
- 5gettersetter.rb
- 6gettersetter.rb
- 7numgettersetter.rb
- 8numgettersetter.rb
- 9privado.rb
- 10herencia.rb
- mixin.rb
- modulo.rb

```
1 module Cafe
2   def cafe1
3   end
4   def cafe2
5   end
6 end
7 module Azucar
8   def a1
9   end
10  def a2
11  end
12 end
13
14 class Espresso
15 include Cafe
16 include Azucar
17   def e1
18   end
19 end
20
21 samp = Espresso.new
22 samp.cafe1
23 samp.cafe2
24 samp.a1
25 samp.a2
26 samp.11
27
```


Herencia y polimorfismo

En la POO el polimorfismo está relacionado con la herencia, es decir, un objeto puede heredar atributos y métodos independiente que no sea una copia exacta del objeto.

Pensemos en un electrodoméstico, una juguera, podría tener los atributos de tamaño, peso, color, y los métodos de encender y apagar. Un horno también podría tener esos atributos y métodos, y no necesariamente ser una copia del objeto juguera. Hemos dejado un ejemplo en el manual del estudiante.

Duck typing

Ruby a diferencia de otros lenguajes de programación es de tipado dinámico, esto quiere decir que las variables no deben ser inicializadas en su tipo, ej: string, float, in, array, etc.

Lo comentamos al inicio del curso, el tipo de variable se define por el contenido del código y en su ejecución.

Ejemplo

```
bienvenida = "hola"  
numero = 1
```

En otros lenguajes sería

```
String bienvenida = "Hola";  
int numero = 1;
```

Material complementario de la unidad

Link a video relacionado

1. <https://www.youtube.com/watch?v=r0KjrzbSRec>

Link a lectura complementaria

1. <https://www.irjet.net/archives/V7/i10/IRJET-V7I10247.pdf>

Link a investigación relacionada

1. https://www.researchgate.net/publication/221536843_Student_understanding_of_object-oriented_programming_as_expressed_in_concept_maps/link/02bfe5137acd2008ec000000/download