Ruby Objetos

Introducción a clases y objetos

En una clase anterior creamos un método para verificar elementos si un elemento estaba en un array

```
def check_if_exist(array, object)
    array.each do |element|
        if element == object
            return true
        end
    end
    false
end
```

Para ocuparlo: `check_if_exist(array, objeto)`

Pero el método include? lo llamamos así: `array.include?(object)`

Motivación

Los objetos nos permiten:

- Reutilizar y compartir código de forma fácil
- Abstraernos de los datos para pensar en alto nivel
- Necesitamos saber de objetos para poder crear aplicaciones con Rails

Nuevos conceptos

Algunos conceptos que vamos a aprender hoy:

- Clase
- Objeto
- Instancia
- Atributo
- Método de clase y de instancia
- Variables locales, de clase y de instancia
- Getters y Setters
- Constructor
- Self

¿Qué es un objeto?

En ruby salvo pocas excepciones todo es un objeto, los literales como los Fixnum, Strings, son objetos, los Array, Hashes, y tanto True como False son objetos.

Idea de concepto simple:

Agrupemos el código en torno a entidades

Para eso definimos una clase

Los tres conceptos más importantes

Clase

“El secreto para comprender bien la programación orientada a objetos es entender a las clases como una fábrica o molde de objetos “

Clase v/s objeto

La clase es el molde, el objeto es el producto, y un objeto específico creado de esa clase se le llama **Instancia.**

“Por ejemplo un molde de una pieza de lego (la clase) y los objetos como los productos de este molde, o sea los legos “

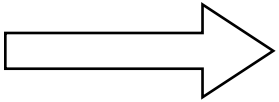
Clase en Ruby

En ruby para definir una clase:

```
class Lego
end
```

Para crear un objeto a partir de ese molde hay que **instanciarlo**, para **instanciar** basta con utilizar el nombre de la clase con el método **new**:

```
lego1 = Lego.new
```



Objeto en la terminal:

```
#<Lego:0x007ff7e0836b90>
```

Resumen

- Las clases son un molde que uno programa.
- Los objetos son los productos del molde.
- A un objeto específico de una clase se le llama instancia
- A través del método new creas instancias

Ejercicio

Crea la clase lego e instancia 10 objetos legos guardandolos en un array

Casi todos los objetos se instancian con .new

Pero algunos que ya conocemos tienen otra forma de instanciarse

```
a = Array.new()
```

=>

```
a = []
```

```
b = Hash.new()
```

=>

```
b = {}
```

Para saber de qué clase es un objeto podemos utilizar el método .class

```
a = []
a.class
```

```
b = {}
b.class
```

Los objetos tienen:

1) Identidad
2) Comportamiento
3) Atributos

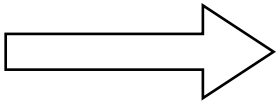
“Entonces tenemos la clase que es un molde de legos, que es capaz de imprimir objetos lego, los cuales tienen un identificador (por ejemplo un número de serie), atributos como el color y tamaño; y comportamientos como acoplarse y causar dolor infinito si uno los pisa. “

Identidad

En ruby todos los objetos tienen un identificador, el lego que creamos previamente tiene uno, este se puede capturar utilizando el método object_id.

Id en la terminal: (ejemplo)

```
lego1.object_id
```



```
#=>70351300158900
```

Comportamiento

El comportamiento define qué puede hacer un objeto

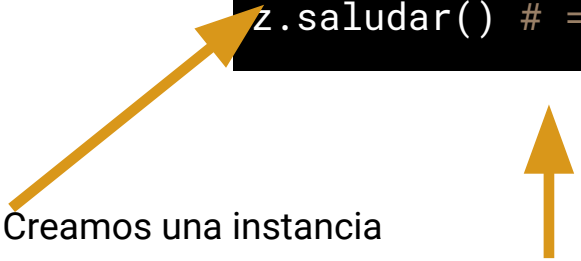
Podemos definir comportamientos agregando método dentro de una clase

```
class Zombie
  def saludar
    puts "braaaains"
  end
end
```

Este tipo de método recibe el nombre de **métodos de instancia.**

Ocupando un método de instancia

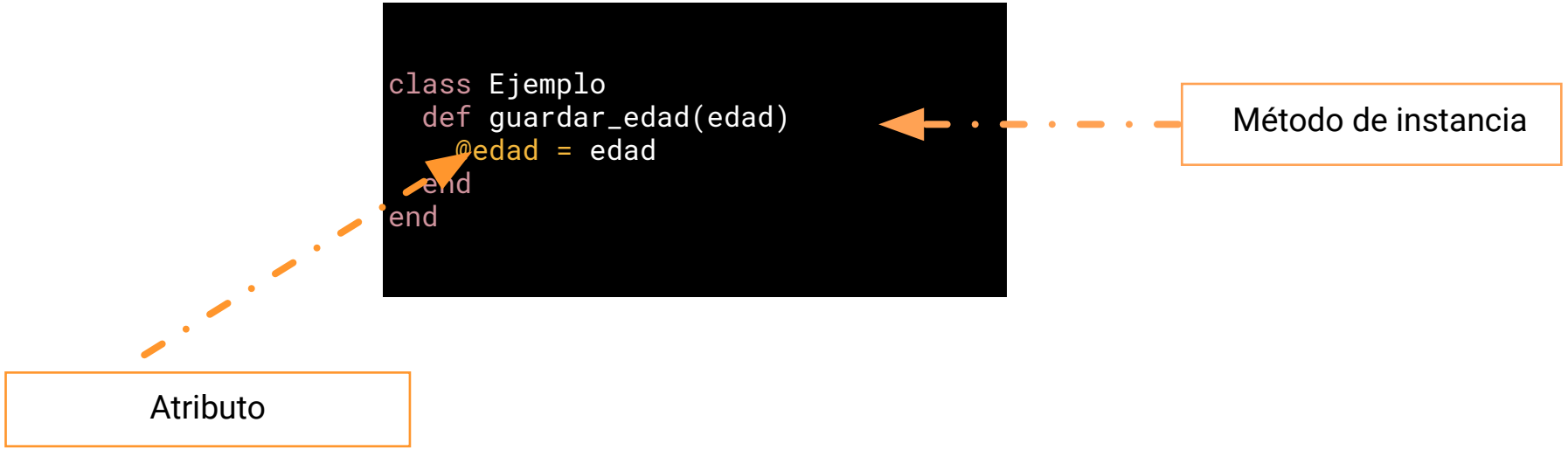
```
class Zombie
  def saludar()
    puts "Braaaains"
  end
end

z = Zombie.new
z.saludar() # => Braaaains
```



Creamos una instancia

Llamamos al método de instancia

Atributos



Los atributos son las variables de instancias, estas se identifican porque empiezan con @

Las variables de instancia dependen de la instancia

```ruby
class Persona
  def bautizar(nuevo_nombre)
    @nombre = nuevo_nombre
  end

  def saludar()
    puts "#{@nombre} dice hola"
  end
end

p1 = Persona.new
p1.bautizar("Jaime")
p2 = Persona.new
p2.bautizar("Francisca")
p2.saludar()
```

¿Qué mensaje se mostrará en pantalla?

Variables de instancia son distintas a las variables locales

```
class Foo
  def bar
    dato = 10
    @otro_dato = 20
  end
end
```

Para entender la diferencias necesitamos definir un nuevo concepto

Ámbito

(Scope)

Es el contexto que tiene un nombre dentro de un programa.

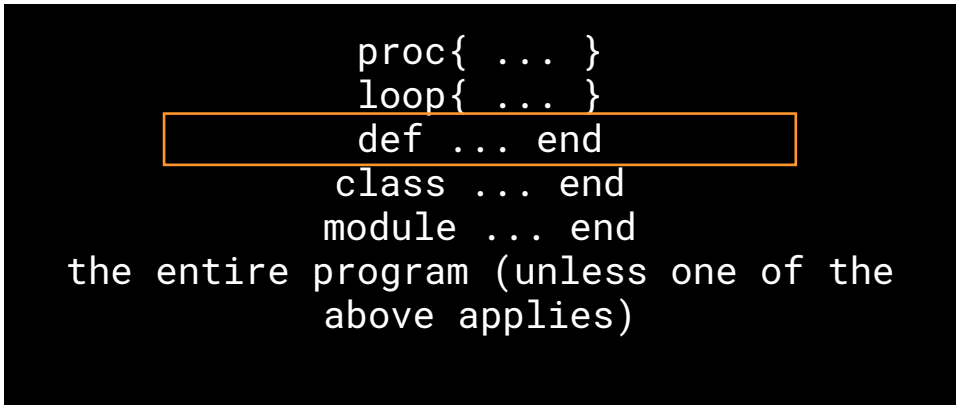
El ámbito determina en qué partes del programa una entidad puede ser usada.

Hasta el momento hemos trabajado exclusivamente con variables locales

El ámbito de las variables locales es siempre una de las siguientes

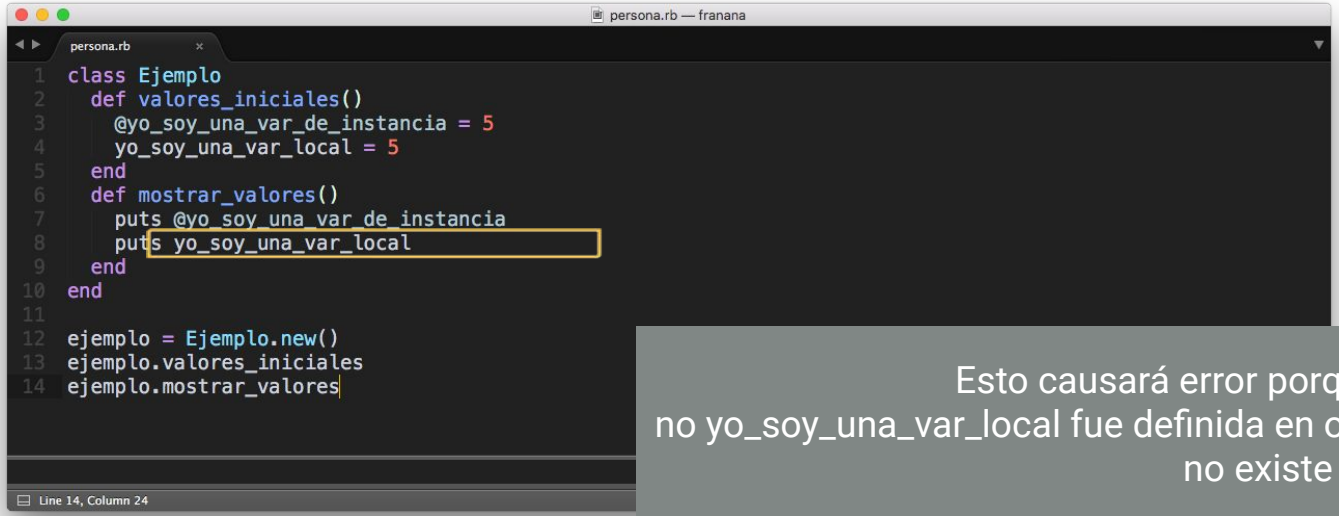
```
proc{ ... }
loop{ ... }
def ... end
class ... end
module ... end
the entire program (unless one of the
above applies)
```

Hasta el momento hemos trabajado exclusivamente con variables locales



O sea que si definimos una variable local dentro de un método, después del método esta no existirá.

Variables locales vs de instancia



Esto causará error porque la variable
no yo_soy_una_var_local fue definida en otro método, en mostrar_valores
no existe

```
NameError: undefined local variable or method `yo_soy_una_var_local' for
#<Ejemplo:0x007fb4591080C8 @yo_soy_una_var_de_instancia=5>
Did you mean?  @yo_soy_una_var_de_instancia
        from (irb):30:in `mostrar_valores'
        from (irb):36
        from /Users/gonzalosanchez/.rvm/rubies/ruby-2.3.0/bin/irb:11:in `<main>'
2.3.0 :037 >
```

Veamos un ejemplo

```ruby
class PersonaX
    def empezar
        @edad = 1
    end

    def envejecer
        @edad += 1
    end

  def
mostrar_edad()
    puts @edad
  end
end

p1 = Persona.new
p1.empezar
p1.envejecer
p1.envejecer
p1.envejecer
p1.mostrar_edad
```

El método constructor

Initialize

```
class Persona
    def initialize()
        @edad = 1
    end
    def envejecer()
        @edad =
@edad + 1
    end
  def mostrar_edad()
    puts @edad
  end
end

p1 = Persona.new
p1.envejecer
p1.envejecer
p1.envejecer
p1.mostrar_edad
```

Un ejemplo con argumentos

```
class Persona
    def initialize(edad)
        @edad = edad
    end
    def envejecer()
        @edad = @edad +
1
    end
  def mostrar_edad()
    puts @edad
  end
end

p1 = Persona.new 5
p1.envejecer
p1.mostrar_edad
```

En ruby las variables de instancia no son accesibles desde fuera del objeto (por ejemplo en main)

```ruby
class Foo
    attr_accessor :a
    def initialize
        @a = 5
    end
end

Foo.new.a
# Este es un error
# No se puede llamar a una
variable de un objeto en ruby
```

Sin getters y setters no podemos acceder a @a

A este comportamiento se le llama encapsulamiento

¿Cúal es el error?

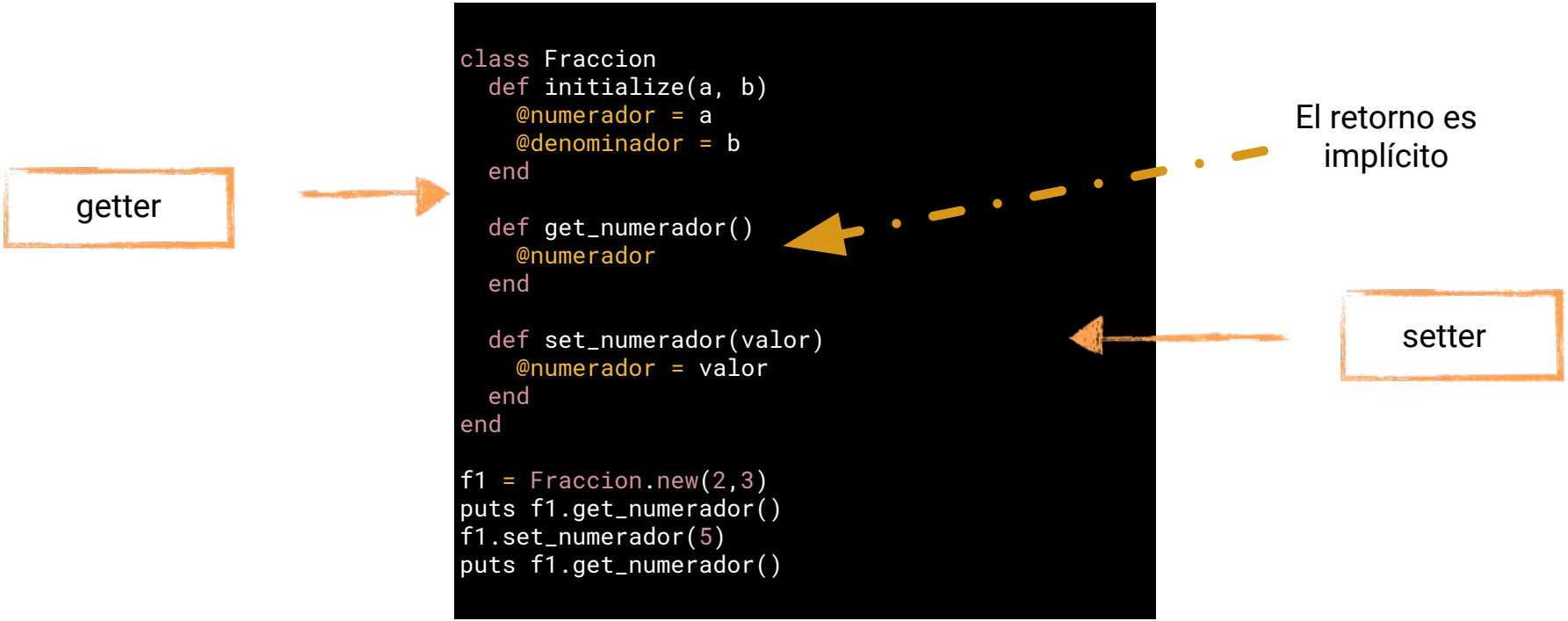
```
class Perro
    def initialize(edad)
        @edad = edad
    end
end

perro1 = Perro.new 10
puts perro1.edad
perro1.edad = 8
```

Los getters y setters se pueden crear de forma automática con attr_accessor

Pero primero aprenderemos otra forma que nos enseñará mucho de ruby

getters / setters

Para acceder a los estados tenemos que crear métodos para obtener los estados (getters) y métodos para cambiarlos (setters)




```ruby
class Fraccion
  def initialize(a, b)
    @numerador = a
    @denominador = b
  end

  def get_numerador()
    @numerador
  end

  def set_numerador(valor)
    @numerador = valor
  end
end

f1 = Fraccion.new(2,3)
puts f1.get_numerador()
f1.set_numerador(5)
puts f1.get_numerador()
```

getter y setters versión 2.0

```ruby
class Perro
    def initialize(edad)
        @edad = edad
    end

    def edad
        @edad
    end

    def edad=(edad)
        @edad = edad
    end
end

perro1 = Perro.new 10
puts perro1.edad # Esto ahora si funcionará
perro1.edad = 8 # Esto también funcionará
```

attr_accessor

(hacer accesible el atributo)

Es posible definir de forma simultánea la variable, los getters y setters a través del **attr_accessor:**

```
class Perro
  attr_accessor :edad
  def initialize(edad)
      @edad = edad
  end
end

perro1 = Perro.new 10
puts perro1.edad
perro1.edad = 8
```

Quiz rápido

Dado el siguiente código

```
class Alumno
  def initialize()
      @notas = []
      nombre = "Humberto"
  end
end
```

¿Cuál es el error con?

```
Alumno.notas
Alumno.new.notas
Alumno.nombre
Alumno.new.nombre
```

Modificar el código para poder acceder al nombre y modificar las notas

```
class Alumno
  def initialize()
      @notas = []
      nombre = "Humberto"
  end
end
```

¿Cuál es el problema con el siguiente código?

```ruby
class Store
  attr_accessor :name
  def initialize(name)
     name = name
  end
end
```

```ruby
store = Store.new("S1")
puts store.name
```

¿Cuál es el problema con el siguiente código?

```ruby
class Product
    attr_reader :stock
    def initialize
        @stock = 0
    end
end
```

```ruby
p1 = Product.new
p1.stock = 5
```

Agregar un método de instancia a **student** para calcular el promedio del alumno

```
class Student
    def initialize(grade1, grade2, grade3)
        @grade1 = grade1
        @grade2 = grade2
        @grade3 = grade3
    end
end
```

```
student = Student.new(2,3,4)
student2 = Student.new(4,5,6)
```

Ejercicio

- Crear la clase carta con número y pinta.
- Instanciar 5 cartas al azar.

¿Cúal es el error?

```ruby
class Foo
    def method1
        puts "hola"
    end
end
```

```ruby
Foo.method1
```

Aquí hay 2 errores grandes. ¿Cuáles son?

```
class Foo(x,y)
    def initialize()
        puts "hola"
    end
end
```

```
Foo.new().method1(2,3)
```

Reflexionando sobre los métodos y el attr_accessor

No debemos confundir los métodos con las variables de instancias

```ruby
class Foo
    attr_accessor :a
    def initialize
        @a = 5
        puts @a # Yo soy una variable
        puts a # Yo soy una método
    end
end

Foo.new.a # Yo estoy llamando al método

# No se puede llamar a una variable de un
objeto en ruby
```

¿Cuáles son los valores?

No debemos confundir los métodos con las variables de instancias

```ruby
class Foo
    attr_accessor :a
    def initialize
        @a = 5
    end

    def change
        a = 10
    end
end

foo = Foo.new
puts foo.a
puts foo.change
puts foo.a
```

Objetos y arreglos

```ruby
class Product
    def initialize(name, stock = 10)
        @name = name
        @stock = 10
    end
end
```

```ruby
products = []
10.times do |i|
    Product.new "product#{i}"
end
```

Estrategia para resolver problemas

Antes de resolver un problema donde haya código en main y en un objeto,

preguntarse, ¿dónde debería ir el código?

Si es un script pequeño o una prueba podemos ponerlo en main

```ruby
products = []
10.times do |i|
    Product.new "product#{i}"
end
```

O podemos ponerlo en otro objeto, por ahora como ejercicio en clases lo pondremos en main

Ejercicio

- Crear la clase alumno, cada alumno tiene un arreglo de calificaciones (enteros) y un nombre.
- Crear un arreglo con al menos 4 alumnos, cada alumnos.

Se pide: Calcular el promedio de notas del curso

Encontrar al alumno que tiene le promedio de notas más alto y devolver su nombre.