



***“UNIVERSIDAD DE LAS FUERZAS ARMADAS”***

***ESPE***

***SOFTWARE ENGINEERING***

***OBJECT-ORIENTED PROGRAMMING***

***TEAM 6 : EncryptMPP***

***Moreira Vinuesa Erick Patricio***

***Pabon Gonzalez Elkin Andres***

***Ponce Arguello Diego Armando***

***TOPIC:***

---

***Other Programming Language - Ruby***

***INSTRUCTOR:***

***PHD. EDISON LASCANO***

***NRC:***

***4680***

***DATE:***

***July 7, 2022***

***SEDE MATRIZ SANGOLQUÍ***

***QUITO-ECUADOR***

***2022***

## 1.What is Ruby?

Ruby is an **interpreter**, high-level, general-purpose programming language which supports multiple programming paradigms.

In this case we are going to use it with an object-oriented paradigm.

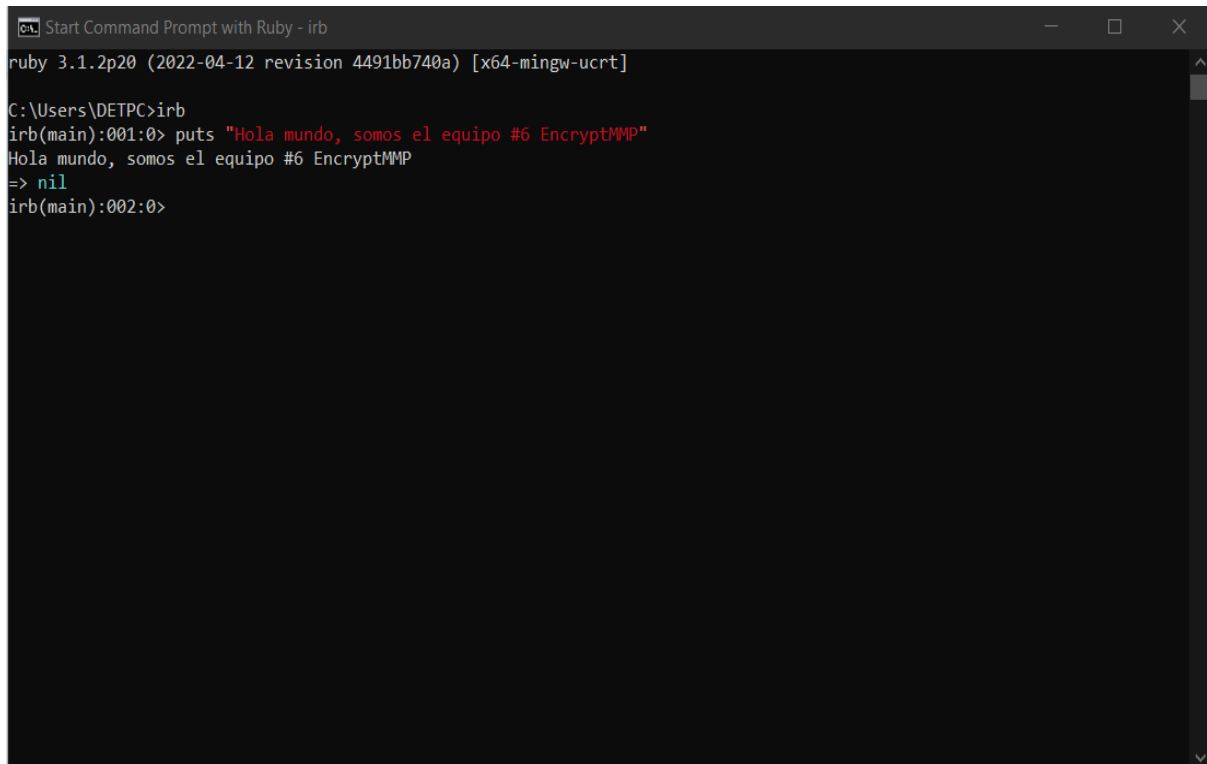
Developer → Ruby was created in the mid-1990s by Yukihiro "Matz" Matsumoto in Japan.

It has there qualities:

- Terse. Short, but still easy to understand.
- Dynamic. Easy to change, anytime and anywhere.
- Duck typing. If you think you understand it, you probably understand it.

Duck typing → Duck typing is a concept related to dynamic typing, where the type or the class of an object is less important than the methods it defines. When you use duck typing, you do not check types at all. Instead, you check for the presence of a given method or attribute.

## 2.Installation of development environment

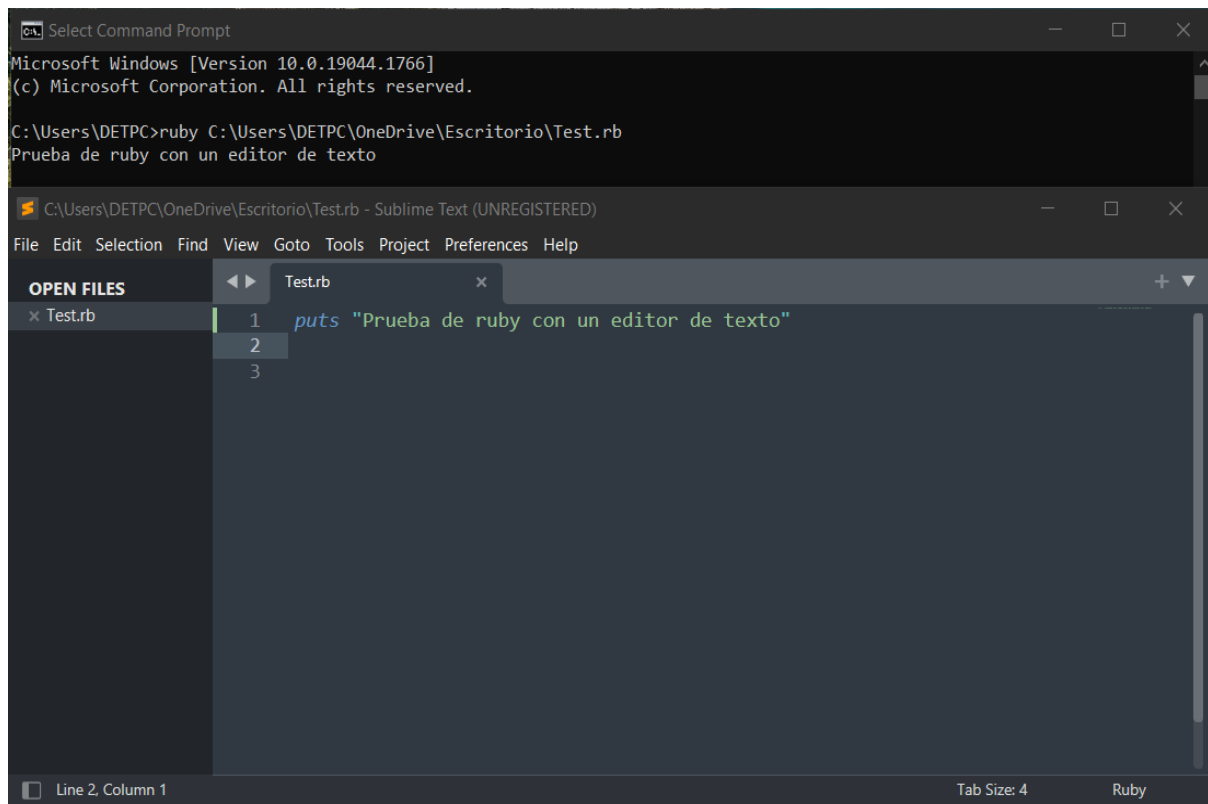


```
Start Command Prompt with Ruby - irb
ruby 3.1.2p20 (2022-04-12 revision 4491bb740a) [x64-mingw-ucrt]

C:\Users\DETPC>irb
irb(main):001:0> puts "Hola mundo, somos el equipo #6 EncryptMMP"
Hola mundo, somos el equipo #6 EncryptMMP
=> nil
irb(main):002:0>
```

As we can see the instruction “**Hola mundo, somos el equipo #6 EncryptMPP**” don't need “;” to print the string.

Also **nil** means that the program has finished.



```
Microsoft Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DETPC>ruby C:\Users\DETPC\OneDrive\Escritorio\Test.rb
Prueba de ruby con un editor de texto
```

C:\Users\DETPC\OneDrive\Escritorio\Test.rb - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

OPEN FILES

Test.rb

```
1 puts "Prueba de ruby con un editor de texto"
2
3
```

Line 2, Column 1 Tab Size: 4 Ruby

Obviously we can program in a text editor for code, for example Sublime text but we have to save the file with the format .rb, then we can run the program in the command prompt as we can see in the above example.

### 3. Data Types

Common Data types in Ruby are:

- Boolean
- Integer
- Floating point
- String
- Array

The screenshot shows a Sublime Text editor window with the title bar 'C:\Users\DETPC\OneDrive\Escritorio\Data Types.rb - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The 'OPEN FILES' sidebar on the left lists four files: Test.rb, AccessModifiers.rb, Classes and Objects implementation.rb, and Data Types.rb. The main editor area displays the following Ruby code:

```
1 #Boolean
2 isEnabled = true
3 isEnabled = false
4
5 #Integers
6 addend1 = 20
7 addend2 = 30
8
9
10 #Floating point
11 value_1 = 17.23
12 value_2 = 13.77
13
14
15 #String
16 movie_1 = "Alice in Wonderland! "
17 movie_2 = "Down the rabbit hole"
18
19 # Array of Integers
20 elements = [-3, -2, -1, 0, 1, 2, 3]
21
22 # Array of Strings
23 elements = ["Alice", "In", "Winterland"]
24
```

The status bar at the bottom indicates 'Line 24, Column 1', 'Tab Size: 4', and 'Ruby'.

#### 4. Classes and Objects implementation

A **Class** is a combination of data (attributes) and functions (methods).

The data and functions of a class can be accessed after creating an instance of a class.

Instance of a class is known as an **Object**.

Let us see an example of a Class in Ruby:

The screenshot shows a Sublime Text editor with a file named 'Classes and Objects implementation.rb'. The code defines an `Employee` class with an `initialize` method that sets `@employeeId` and `@employeeName`. It also includes `setEmployeeId`, `setEmployeeName`, `getEmployeeId`, and `getEmployeeName` methods. In the main script, an `Employee` object is created, its ID is set to 1, and its name is set to 'Erick'. The output in the Command Prompt shows: 'El ID del trabajador es 1' and 'El nombre del trabajador es Erick'.

In this example the first step was to create the class “Employee”, and as we can see in the line #24 We can create objects of Employee class as we did up there. The getters and setters concepts that were applied here, we will see later. The relevant thing here is known about the definition of class and object.

## 5. Access Modifiers

In Ruby, we have the following access modifiers to be used on methods.

1. **Public:** In this all members are available to everyone to modify.
2. **Private:** In this the members can only be accessed by functions inside the class.
3. **Protected:** The members in protected class can only be accessed by functions inside subclass. We will see the subclass concept in Inheritance.

The screenshot shows a Sublime Text editor with a file named 'AccessModifiers.rb'. The code defines a `Vehicle` class with an `initialize` method that sets `@color` and `@brand`. It also includes `turnOn` and `turnOff` methods, which are marked as `public`. In the main script, a `Vehicle` object is created with color 'azul' and brand 'mercedes'. The output in the Command Prompt shows: 'El mercedes se ha encendido' and 'El mercedes se ha apagado'. A white arrow points from the `public :turnOn, :turnOff` line in the code to the output in the Command Prompt.

```
AccessModifiers.rb
7 end
8
9 #methods
10
11 def turnOn
12   puts "El #{@brand} se ha encendido"
13 end
14
15 def turnOff
16   puts "El #{@brand} se ha apagado"
17 end
18
19 private :turnOn, :turnOff
20
21 end
22
23 #Object #1
24
25 mercedes = Vehicle.new('azul', 'mercedes')
27 puts mercedes.turnOn
28 puts mercedes.turnOff
```

```
C:\Users\DETPC>ruby C:\Users\DETPC\OneDrive\Escritorio\AccessModifiers.rb
C:\Users\DETPC\OneDrive\Escritorio\AccessModifiers.rb:27:in '<main>': private method 'turnOn' called for #<Vehicle:0x0000018a9b0db3b8 @color="azul", @brand="mercedes"> (NoMethodError)

puts mercedes.turnOn\r
*****
C:\Users\DETPC>
```

## 6. Data Input and Output

An example program in Ruby that asks for your name and then says it:

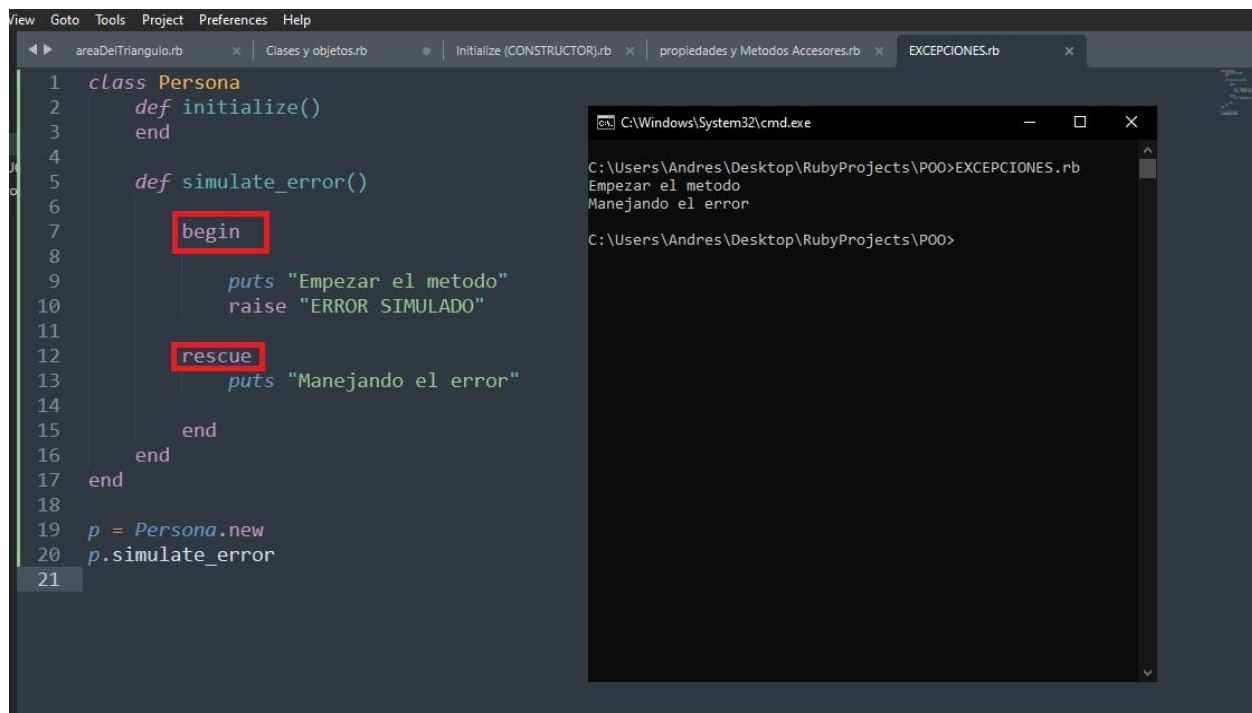
```
Data Input and Output.rb
1 puts "Cual es tu nombre?"
2 name = gets.chomp
3 puts "Hola mi estimado amigo, #{name}"
```

```
C:\Users\DETPC>ruby "C:\Users\DETPC\OneDrive\Escritorio\Data Input and Output.rb"
Cual es tu nombre?
Erick
Hola mi estimado amigo, Erick
C:\Users\DETPC>
```

## 7. Exceptions:

Exception handling in any programming language is very important, because if an error occurs in the code, an exception must be handled to prevent the program from working correctly.

To handle exceptions and avoid closing our program we use the **Begin** and **Rescue** keywords.



The screenshot shows a Ruby IDE with a file named `EXCEPCIONES.rb` open. The code defines a `Persona` class with two methods: `initialize()` and `simulate_error()`. The `simulate_error()` method uses a `begin` block to execute `puts "Empezar el metodo"` and `raise "ERROR SIMULADO"`, followed by a `rescue` block that executes `puts "Manejando el error"`. The code is then executed, and a command prompt window shows the output: `C:\Users\Andres\Desktop\RubyProjects\P00>EXCEPCIONES.rb`, `Empezar el metodo`, `Manejando el error`, and the prompt `C:\Users\Andres\Desktop\RubyProjects\P00>`.

```
1 class Persona
2   def initialize()
3   end
4
5   def simulate_error()
6
7     begin
8
9       puts "Empezar el metodo"
10      raise "ERROR SIMULADO"
11
12      rescue
13        puts "Manejando el error"
14      end
15    end
16  end
17 end
18
19 p = Persona.new
20 p.simulate_error
21
```

```
C:\Windows\System32\cmd.exe
C:\Users\Andres\Desktop\RubyProjects\P00>EXCEPCIONES.rb
Empezar el metodo
Manejando el error
C:\Users\Andres\Desktop\RubyProjects\P00>
```

To better understand the example, line #10 simulates an error that would terminate the execution of the program with the keyword **raise**, an exception which causes a *RuntimeError*.

**Rescue** contains the action to take in case an exception occurs.

## 8. Data persistence

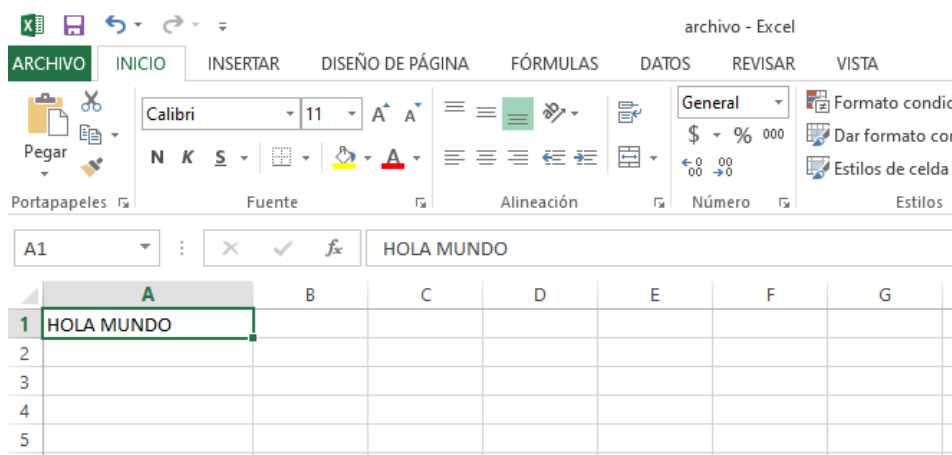
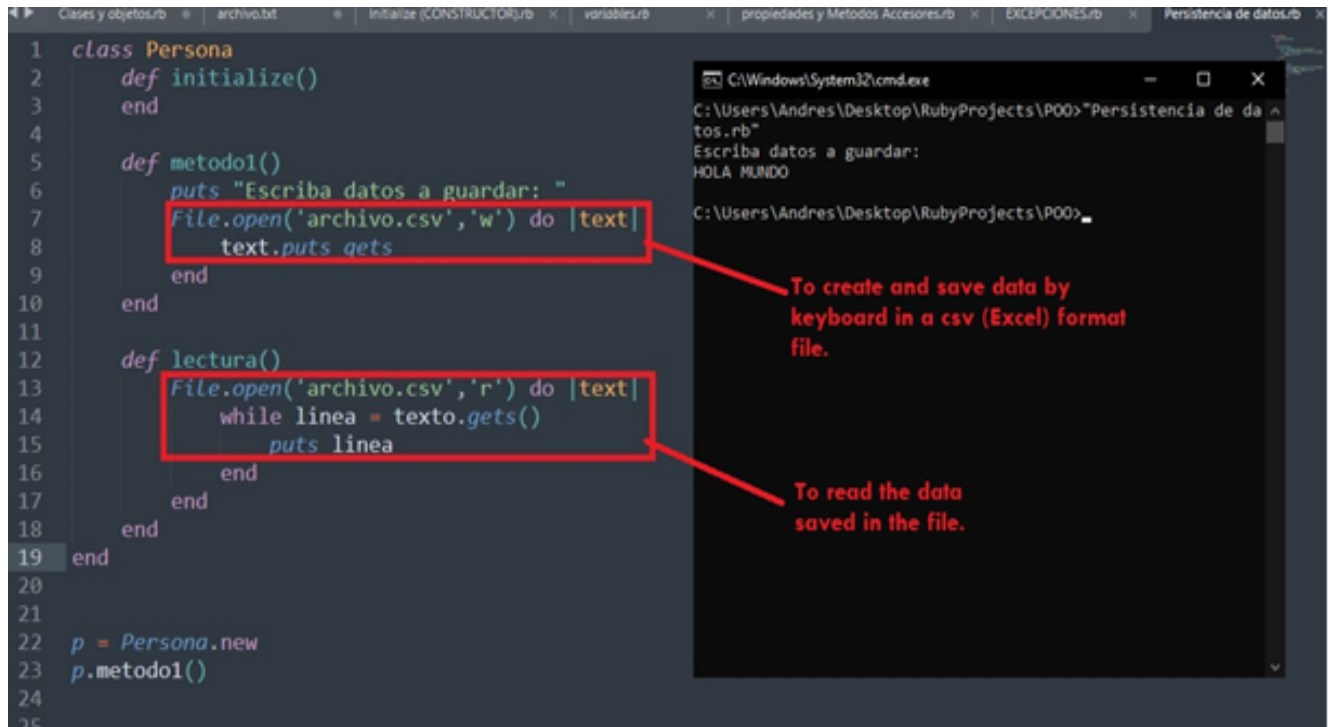
Persistence is the ability to save information from a program so that it can be used again at a later time. It is what users know as Save the file and then Open the file.

In order to create files in Ruby, we use the reserved word **File** and its **open** method with two parameters, one in which the file to be created is named along with the file type (.txt, .csv, etc) and in the second parameter a letter that represents what action can be performed with the file

w = write , r = read

```
File.open('archivo.csv', 'w') do |text|
```

text = variable which helps us to save the data we want to enter in a line.  
do = to give the order to do the defined action.



## 9. Encapsulation

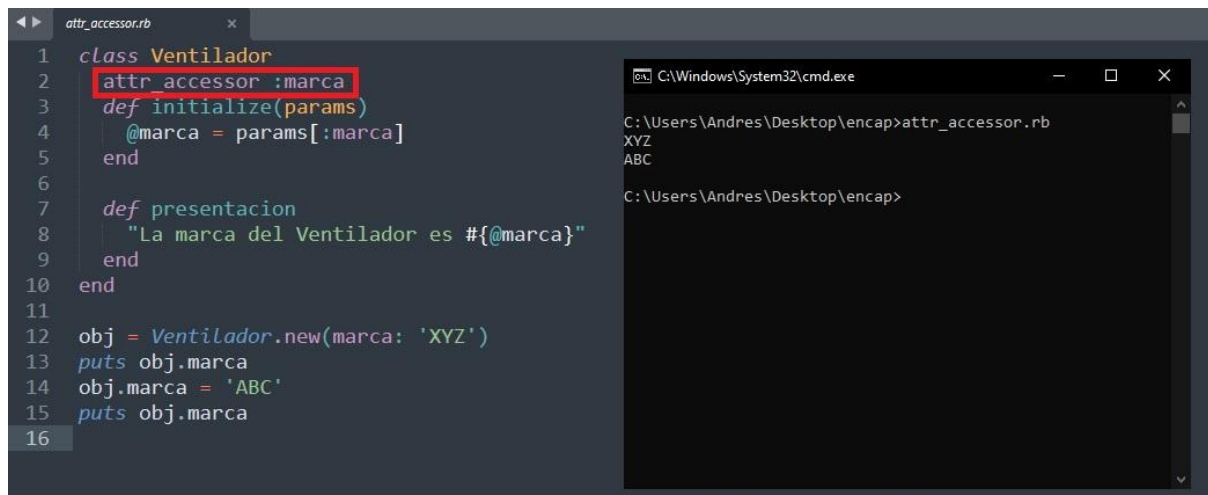
Object-oriented programming has certain characteristics, such as encapsulation. Here are some characteristics of encapsulation:

- Expose the external behavior of the class using methods.
- Protect internal data.
- Scopes in Ruby: *public*, *protected*, *private*.
- Accessories.



## Types of Accessories:

attr\_reader  
attr\_writer  
attr\_accessor

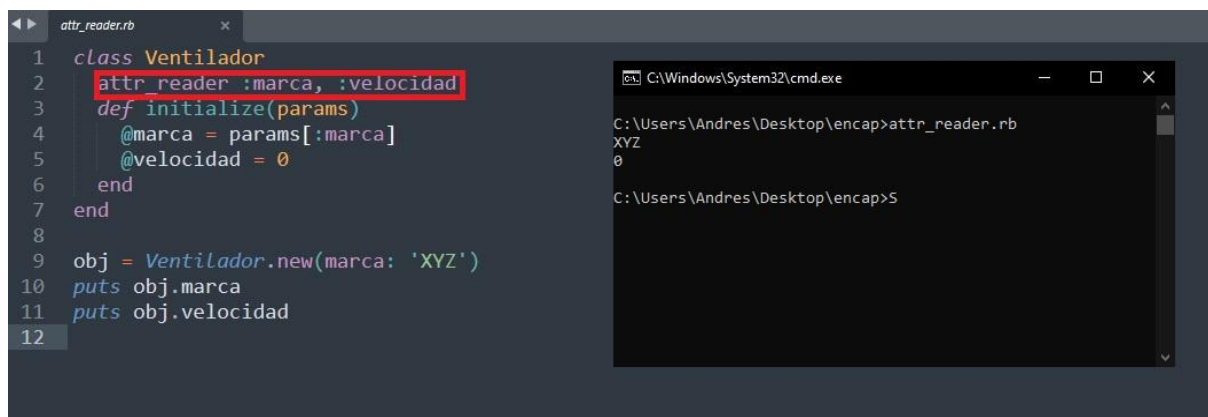


The screenshot shows a Ruby file named `attr_accessor.rb` with the following code:

```
1 class Ventilador
2   attr_accessor :marca
3   def initialize(params)
4     @marca = params[:marca]
5   end
6
7   def presentacion
8     "La marca del Ventilador es #{@marca}"
9   end
10 end
11
12 obj = Ventilador.new(marca: 'XYZ')
13 puts obj.marca
14 obj.marca = 'ABC'
15 puts obj.marca
16
```

The terminal output shows the execution of the script:

```
C:\Windows\System32\cmd.exe
C:\Users\Andres\Desktop\encap>attr_accessor.rb
XYZ
ABC
C:\Users\Andres\Desktop\encap>
```



The screenshot shows a Ruby file named `attr_reader.rb` with the following code:

```
1 class Ventilador
2   attr_reader :marca, :velocidad
3   def initialize(params)
4     @marca = params[:marca]
5     @velocidad = 0
6   end
7 end
8
9 obj = Ventilador.new(marca: 'XYZ')
10 puts obj.marca
11 puts obj.velocidad
12
```

The terminal output shows the execution of the script:

```
C:\Windows\System32\cmd.exe
C:\Users\Andres\Desktop\encap>attr_reader.rb
XYZ
0
C:\Users\Andres\Desktop\encap>S
```

Access modifiers are the access that we can give to different methods, these can be:

- **Public:** Default
- **Private:** They will only work in that class, they cannot be called from the object nor can they inherit a child class.
- **Protected:** They cannot be called once the object is created, but they will be able to inherit a child class.

By default, the attributes of an object in Ruby are private, that is, they can only be accessed by methods of the same class. To give them visibility from other parts of the application, `attr_accessor` (read and write), `attr_reader` (read), and `attr_writer` (write) are used.

**Private:**

```

1  class Food
2    def public_method
3
4    end
5    private
6
7    def bacon
8    end
9
10   def orange
11   end
12
13   def coconut
14   end
15
16 end
17

```

## Protected:

```

private.rb  x  protected.rb  x  attr_writer.rb
1  class Food
2    def initialize(name)
3      @name = name
4    end
5    def ==(other)
6      name == other.name
7    end
8    protected
9    attr_reader :name
10 end
11 food = Food.new("chocolate")
12 puts food == food

```

## 10. Constructor

It has the characteristic that it is a method that is executed when we create an object of a class and is usually used to initialize values that the object will have.

In Ruby it is called as **initialize**.

```
1 class Video
2
3   attr_accessor :minutes, :tittle
4
5   def initialize(tittle)
6     self.tittle = tittle
7     puts "I am the constructor"
8   end
9
10  def play
11  end
12
13  def pause
14  end
15
16  def stop
17  end
18
19 end
20
21
22 video_P00 = Video.new("Class and Object")
23
24
```

Terminal output:

```
C:\Users\Andres\Desktop\RubyProjects\P00>"Initialize (CONSTRUCTOR).rb"
I am the constructor
C:\Users\Andres\Desktop\RubyProjects\P00>
```

## 11. Getter, setter methods:

They are important methods in OOP since they allow us to manipulate the attributes or characteristics that our classes present.

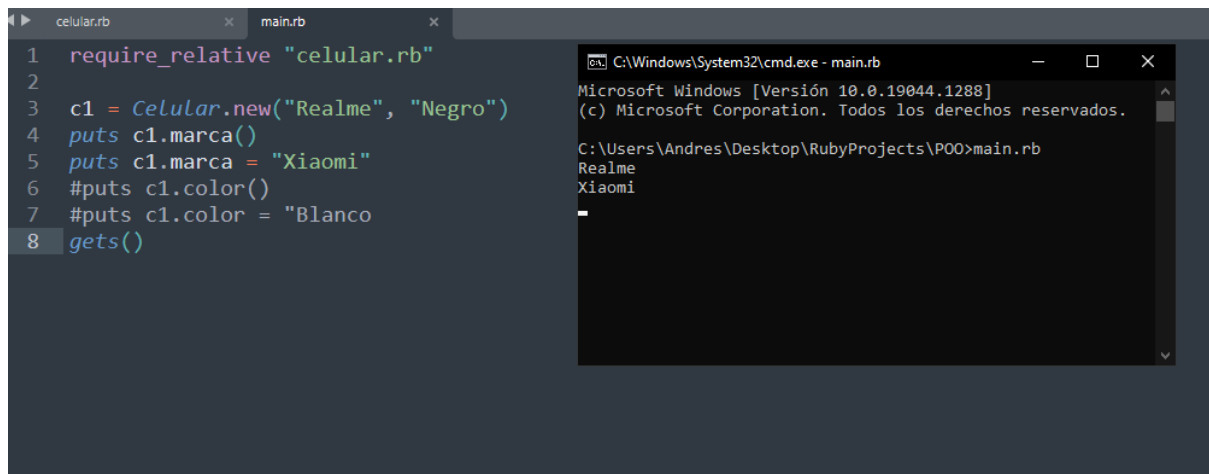
A "setter" method is used to "load a value" (assign a value to a variable).

A "getter" method is used to "return the value" (only return the attribute information to the requester).

```
1 class Celular
2
3   def initialize(marca,color)
4     @marca = marca
5     @color = color
6   end
7
8   def llamar()
9     puts @marca + "-" + @color + " esta llamando ..."
10  end
11
12  def marca
13    return @marca
14  end
15
16  def marca=(marca)
17    @marca = marca
18  end
19 end
20
21
```

Annotations in the code:

- A red box highlights the `def marca` method (lines 12-14), with a red arrow pointing to it and the label **get**.
- A red box highlights the `def marca=(marca)` method (lines 16-18), with a red arrow pointing to it and the label **set**.



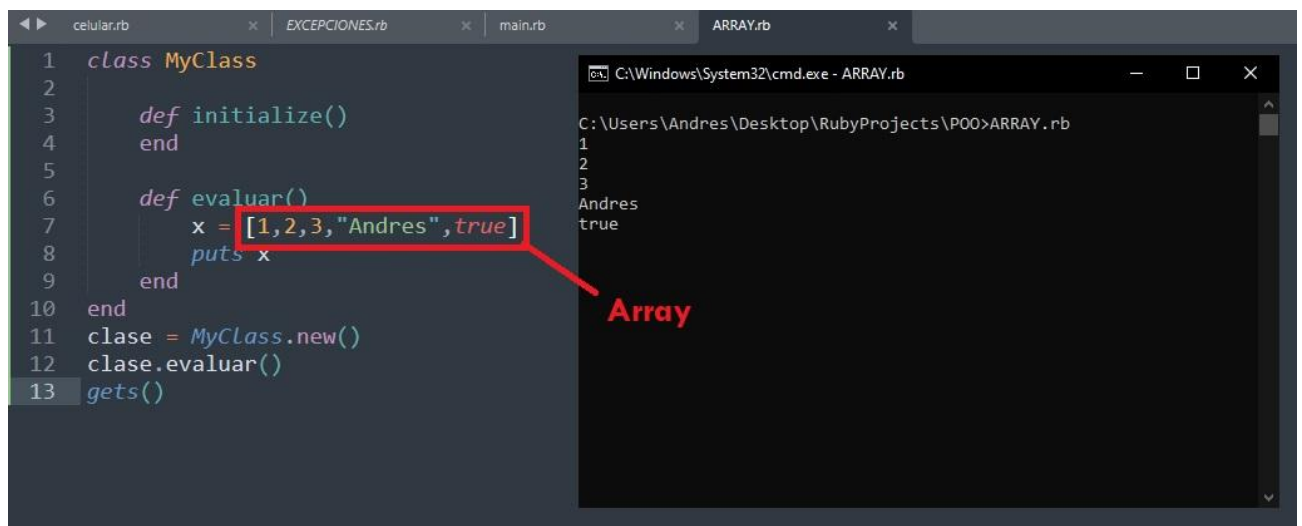
```
1 require_relative "celular.rb"
2
3 c1 = Celular.new("Realme", "Negro")
4 puts c1.marca()
5 puts c1.marca = "Xiaomi"
6 #puts c1.color()
7 #puts c1.color = "Blanco"
8 gets()
```

```
C:\Windows\System32\cmd.exe - main.rb
Microsoft Windows [Versión 10.0.19044.1288]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Andres\Desktop\RubyProjects\P00>main.rb
Realme
Xiaomi
```

## 12. Arrays and collections:

In Ruby there is the Array data structure that allows you to store a collection of data and then access any of its components by means of a subscript. To define an array, the nomenclature of square brackets [ ] is used, it can also contain different data types such as string, floating and boolean data types.



```
1 class MyClass
2
3   def initialize()
4     end
5
6   def evaluar()
7     x = [1,2,3,"Andres",true]
8     puts x
9   end
10 end
11 clase = MyClass.new()
12 clase.evaluar()
13 gets()
```

```
C:\Windows\System32\cmd.exe - ARRAY.rb
C:\Users\Andres\Desktop\RubyProjects\P00>ARRAY.rb
1
2
3
Andres
true
```

Array

## Polymorphism

In Ruby we can achieve polymorphism using method overriding. In polymorphism, Ruby invokes the actual method during running of the program. One way of achieving polymorphism is with inheritance.

## Data Abstraction in Modules

modules are defined as a set of methods, classes, and constants together. For example, consider the `sqrt()` method present in the `math` module. Whenever we need to calculate the square root of a non-negative number, we simply call the `sqrt()` method present in the `math` module and send the number as a parameter without understanding the actual algorithm that actually calculates the square root of the numbers.

## Data abstraction in classes

We can use classes to perform data abstraction in ruby. The class allows us to group information and methods using access specifiers (`private`, `protected`, `public`). The Class will determine what information must be visible and what must not.

### Data abstraction using access control:

There are three levels of access control in Ruby (`private`, `protected`, `public`).

```
# Ruby program to demonstrate Data Abstraction
class Geeks
  # defining publicMethod
  public

  def publicMethod
    puts "In Public!"
    # calling privateMethod inside publicMethod
    privateMethod
  end

  # defining privateMethod
  private

  def privateMethod
    puts "In Private!"
  end
end

# creating an object of class Geeks
obj = Geeks.new

# calling the public method of class Geeks
obj.publicMethod
```

### Ruby interface with raise

This way would be using a module with empty methods whose body is to raise an exception when sent to objects.

```

1  module Animal
2    def eat
3      raise 'Not implemented'
4    end
5
6    def travel
7      raise 'Not implemented'
8    end
9  end

```

```

1  class Cat
2    include Animal
3
4    def eat
5      puts 'The cat is eating'
6    end
7  end

```

### Ruby interface with tests

An interface can be imitated using unit tests. The way is that the tests would have to pass so that the code has an adequate level of validity, that is, the contract between the class and the "interface" would be fulfilled.

```

1  describe Cat do
2
3    before { @cat = described_class.new }
4
5    describe '#meow' do
6      it 'meows' do
7        expect(@cat.meow).to be_a(String)
8      end
9    end
10 end

```

