**HOMEWORK 5**

1. **¿Cómo funcionan los modificadores privados, públicos y protegidos en Ruby?**

**R//** Los modificadores de acceso en Ruby te permiten controlar la visibilidad de los métodos en una clase, lo cual es útil para implementar la encapsulación.

* Privado (private): Los métodos marcados como privados solo pueden ser llamados desde dentro de la clase que los define. No se pueden llamar desde instancias de la clase ni desde ninguna otra clase. Estos métodos son utilizados principalmente para encapsular la implementación interna de una clase y ocultarla del mundo exterior. Para definirlo se utiliza la palabra clave private seguida de los nombres de los métodos, separados por comas o en múltiples líneas.

Ejemplo:

class MiClase

def publico

puts "Método público"

end

private

def privado

puts "Método privado"

end

end

objeto = MiClase.new

objeto.publico # Esto es válido

objeto.privado # Esto generaría un error

* Público (public): Los métodos públicos pueden ser llamados desde cualquier parte del programa, ya sea desde dentro de la clase que los define, desde instancias de la clase o desde otras clases.

Se declara utilizando la palabra clave public, seguida de los nombres de los métodos.

Ejemplo:

class MiClase

def publico

puts "Método público"

end

private

def privado

puts "Método privado"

end

public

def otro\_metodo\_publico

puts "Otro método público"

end

end

objeto = MiClase.new

objeto.publico # Esto es válido

objeto.otro\_metodo\_publico # Esto también es válido

* Protegido (protected): Los métodos protegidos en Ruby pueden ser llamados desde dentro de la clase que los define y también desde instancias de la misma clase o de sus subclases. Estos métodos permiten el acceso a métodos de una clase desde otras instancias de la misma clase o de sus subclases, pero no desde otras clases externas. Se definen utilizando la palabra clave protected seguida de los nombres de los métodos.

Ejemplo:

class MiClase

def publico

puts "Método público"

end

private

def privado

puts "Método privado"

end

protected

def protegido

puts "Método protegido"

end

end

objeto1 = MiClase.new

objeto2 = MiClase.new

objeto1.publico # Esto es válido

objeto1.protegido # Esto es válido

objeto2.publico # Esto es válido

objeto2.protegido # Esto también es válido

1. **¿Cómo puedo hacer para configurar el método de inicialización, para que funcione recibiendo opcional argumento?**

**R//** En Ruby, puedes configurar el método de inicialización (también conocido como constructor) para que acepte un argumento opcional utilizando parámetros con valores predeterminados.

Ejemplo:

class MiClase

def initialize(opcional = nil)

@opcional = opcional

end

end

En el ejemplo anterior, el método initialize de la clase MiClase tiene un parámetro llamado opcional con un valor predeterminado de nil. Esto significa que, si no se proporciona ningún argumento al crear una instancia de la clase, el valor de opcional será nil. Sin embargo, también puedes pasar un valor opcional al crear la instancia de la clase:

objeto1 = MiClase.new # Sin argumento opcional

objeto2 = MiClase.new("Hola") # Con argumento opcional

En el primer caso, @opcional dentro de la instancia objeto1 será nil, mientras que, en el segundo caso, @opcional dentro de objeto2 será igual a "Hola".

De esta manera, puedes configurar el método de inicialización para que acepte un argumento opcional y establecer valores predeterminados para los parámetros si no se proporcionan al crear una instancia de la clase.