Atributos y Métodos - Relación de Ejercicios

Ismael Sallami Moreno

November 2024

1 Ejercicio 1

1.1 Parte a: Hacerlo en Java

Vamos a crear una clase `Coche` en Java. La clase tendrá atributos para la marca, el modelo y la matrícula del coche, y una variable de clase para contar el número total de coches construidos.

```
public class Coche {
    private String marca;
    private String modelo;
    private String matricula;
    private static int contadorCoches = 0;

    public Coche(String marca, String modelo, String matricula) {
        this.marca = marca;
        this.modelo = modelo;
        this.matricula = matricula;
        contadorCoches++;
    }

    public static int getContadorCoches() {
        return contadorCoches;
    }

    public String getEstado() {
        return "Marca: " + marca + ", Modelo: " + modelo + ", Matrícula: " + matricula;
    }

    public static void main(String[] args) {
        Coche coche1 = new Coche("Toyota", "Corolla", "ABC-123");
        Coche coche2 = new Coche("Honda", "Civic", "XYZ-456");
        System.out.println("Número de coches construidos: " + Coche.getContadorCoches());
        System.out.println("Estado del coche 1: " + coche1.getEstado());
        System.out.println("Estado del coche 2: " + coche2.getEstado());
```

```
    }
}
```

¿Has tenido que usar la pseudovariable `this` en el constructor? Sí, es necesario usar `this` para diferenciar entre las variables de instancia y los parámetros del constructor que tienen el mismo nombre.

1.2 Parte b: Hacerlo en Ruby (Variable de Clase)

Vamos a crear una clase `Coche` en Ruby utilizando una variable de clase (`@@`) para el contador de coches.

```
class Coche
  @@contador_coches = 0

  def initialize(marca, modelo, matricula)
    @marca = marca
    @modelo = modelo
    @matricula = matricula
    @@contador_coches += 1
  end

  def self.contador_coches
    @@contador_coches
  end

  def estado
    "Marca: #{@marca}, Modelo: #{@modelo}, Matrícula: #{@matricula}"
  end
end

coche1 = Coche.new("Toyota", "Corolla", "ABC-123")
coche2 = Coche.new("Honda", "Civic", "XYZ-456")
puts "Número de coches construidos: #{Coche.contador_coches}"
puts "Estado del coche 1: #{coche1.estado}"
puts "Estado del coche 2: #{coche2.estado}"
```

1.3 Parte c: Hacerlo en Ruby (Atributo de Instancia)

Vamos a crear una clase `Coche` en Ruby utilizando un atributo de instancia (`@`) para el contador de coches.

```
class Coche
  @contador_coches = 0

  class << self
    attr_accessor :contador_coches
```

```
  end

  def initialize(marca, modelo, matricula)
    @marca = marca
    @modelo = modelo
    @matricula = matricula
    self.class.contador_coches += 1
  end

  def self.contador_coches
    @contador_coches
  end

  def estado
    "Marca: #{@marca}, Modelo: #{@modelo}, Matrícula: #{@matricula}"
  end
end

coche1 = Coche.new("Toyota", "Corolla", "ABC-123")
coche2 = Coche.new("Honda", "Civic", "XYZ-456")
puts "Número de coches construidos: #{Coche.contador_coches}"
puts "Estado del coche 1: #{coche1.estado}"
puts "Estado del coche 2: #{coche2.estado}"
```

2 Ejercicio 3

2.1 Pregunta

En la clase de los ejercicios anteriores, añadir un método que devuelva el número de coches que se han instanciado. Decidir si el método debe ser de instancia o de clase.

2.2 Parte a: Hacerlo en Java

Vamos a crear un método de clase que devuelva el número de instancias de la clase `Car`:

```
public class Car {
    private static int instanceCount = 0;
    private String model;

    public Car(String model) {
        this.model = model;
        instanceCount++;
    }
```

```
    public static int getInstanceCount() {
        return instanceCount;
    }

    public static void main(String[] args) {
        new Car("Toyota");
        new Car("Honda");
        System.out.println("Número de coches instanciados: " + Car.getInstanceCount());
    }
}
```

En este caso, el método `getInstanceCount()` es un método de clase, ya que devuelve el número total de instancias creadas.

2.3 Parte b: Hacerlo en Ruby

2.3.1 Versión 1

Primero, vamos a hacer una versión simple utilizando una variable de clase:

```
class Car
  @@instance_count = 0

  def initialize(model)
    @model = model
    @@instance_count += 1
  end

  def self.instance_count
    @@instance_count
  end
end

car1 = Car.new("Toyota")
car2 = Car.new("Honda")
puts "Número de coches instanciados: #{Car.instance_count}"
```

2.3.2 Versión 2

Ahora, vamos a hacer una versión que no utiliza variables de clase y en su lugar emplea una pseudovariable `self` para contar las instancias:

```
class Car
  @instance_count = 0

  class << self
    attr_accessor :instance_count
```

```
  end

  def initialize(model)
    @model = model
    self.class.instance_count += 1
  end

  def self.instance_count
    @instance_count
  end
end

car1 = Car.new("Toyota")
car2 = Car.new("Honda")
puts "Número de coches instanciados: #{Car.instance_count}"
```

En ambas versiones, `instance_count` es un método de clase, ya que necesitamos contar todas las instancias de la clase `Car`.

3 Ejercicio 4

3.1 Pregunta

Pensar un ejemplo en el que sea necesario usar una variable de clase para almacenar una constante numérica. Implementar dicha clase y un ejemplo de uso donde se emplee dicha constante en una instrucción.

3.2 Parte a: Hacerlo en Java

Vamos a crear una clase en Java que utilice una variable de clase para almacenar una constante numérica:

```
public class ConstanteEjemplo {
    public static final double PI = 3.14159;

    public double calcularCircunferencia(double radio) {
        return 2 * PI * radio;
    }

    public static void main(String[] args) {
        ConstanteEjemplo ejemplo = new ConstanteEjemplo();
        double circunferencia = ejemplo.calcularCircunferencia(5);
        System.out.println("La circunferencia es: " + circunferencia);
    }
}
```

3.3 Parte b: Hacerlo en Ruby (Variable de Clase)

Vamos a hacerlo en Ruby utilizando una variable de clase:

```
class ConstanteEjemplo
  @@pi = 3.14159

  def calcular_circunferencia(radio)
    2 * @@pi * radio
  end
end

ejemplo = ConstanteEjemplo.new
circunferencia = ejemplo.calcular_circunferencia(5)
puts "La circunferencia es: #{circunferencia}"
```

3.4 Parte c: Hacerlo en Ruby (Atributo de Instancia)

Finalmente, vamos a hacerlo en Ruby considerando un atributo de instancia de la clase para la constante:

```
class ConstanteEjemplo
  def initialize
    @pi = 3.14159
  end

  def calcular_circunferencia(radio)
    2 * @pi * radio
  end
end

ejemplo = ConstanteEjemplo.new
circunferencia = ejemplo.calcular_circunferencia(5)
puts "La circunferencia es: #{circunferencia}"
```

4 Ejercicio 4

4.1 Parte a: Hacerlo en Java

Primero, vamos a crear una clase en Java que utilice una variable de clase para almacenar una constante numérica. Aquí tienes un ejemplo sencillo:

```
public class Ejemplo {
    public static final double CONST_NUMERICA = 3.14;

    public static void mostrarConstante() {
        System.out.println("La constante es: " + CONST_NUMERICA);
```

```
    }

    public static void main(String[] args) {
        Ejemplo.mostrarConstante();
    }
}
```

4.2 Parte b: Hacerlo en Ruby (Variable de Clase)

Ahora, vamos a ver cómo hacerlo en Ruby utilizando una variable de clase:

```
class Ejemplo
  @@const_numerica = 3.14

  def self.mostrar_constante
    puts "La constante es: #{@@const_numerica}"
  end
end

Ejemplo.mostrar_constante
```

4.3 Parte c: Hacerlo en Ruby (Atributo de Instancia)

Finalmente, vamos a hacerlo en Ruby considerando un atributo de instancia de la clase:

```
class Ejemplo
  def initialize
    @const_numerica = 3.14
  end

  def mostrar_constante
    puts "La constante es: #{@const_numerica}"
  end
end

ejemplo = Ejemplo.new
ejemplo.mostrar_constante
```

5 Ejercicio 5

Para lograr la solución sin usar `instance_variable_get`, hemos hecho un pequeño ajuste a la clase para permitir el acceso al atributo privado de una manera controlada. Vamos a agregar un método `public` para obtener la capacidad de otro objeto.

5.1 Solución en Ruby

```
class CloudDrive
    def initialize(capacity)
        @capacity = capacity
    end

    private

    def get_capacity
        @capacity
    end

    public

    def set_from_other(other)
        @capacity = other.capacity
    end

    def set_from_other_v2(other)
        @capacity = other.get_capacity
    end

    # Método público para acceder a la capacidad de otro objeto
    def capacity
        @capacity
    end
end

# Crear instancias y probar los métodos
cd1 = CloudDrive.new(100)
cd2 = CloudDrive.new(200)
cd1.set_from_other(cd2)
puts cd1.capacity # Salida esperada: 200
cd1.set_from_other_v2(cd2)
puts cd1.capacity # Salida esperada: 200
```

5.2 Explicación

En este código:

- He añadido un método `public` llamado `capacity` que devuelve el valor de `@capacity`.
- Esto permite que el método `set_from_other` acceda a la capacidad de otro objeto sin necesidad de usar `instance_variable_get`.

De esta manera, el código cumple con el encapsulamiento y sigue siendo claro y mantenible.

6 Ejercicio 6

6.1 Pregunta

En Java, ¿se puede acceder desde un método de instancia a una variable de clase de la misma clase? ¿En qué circunstancias?

6.2 Respuesta

Sí, en Java se puede acceder desde un método de instancia a una variable de clase de la misma clase. Esto es posible porque las variables de clase (también conocidas como variables estáticas) son compartidas por todas las instancias de la clase. Aquí hay un ejemplo ilustrativo:

```
public class Ejemplo {
    private static int variableDeClase = 10;

    public int obtenerVariableDeClase() {
        return variableDeClase;
    }

    public static void main(String[] args) {
        Ejemplo instancia = new Ejemplo();
        System.out.println("Variable de clase: " + instancia.obtenerVariableDeClase());
    }
}
```

En este ejemplo, el método de instancia `obtenerVariableDeClase()` puede acceder a la variable de clase `variableDeClase`.

7 Ejercicio 7

7.1 Pregunta

En Ruby, ¿se puede acceder desde un método de instancia a una variable de clase (@@) de la misma clase? ¿En qué circunstancias?

7.2 Respuesta

Sí, en Ruby se puede acceder desde un método de instancia a una variable de clase (`@@`) de la misma clase. Las variables de clase en Ruby son compartidas por todas las instancias de la clase. Aquí hay un ejemplo:

```
class Ejemplo
  @@variable_de_clase = 10

  def obtener_variable_de_clase
    @@variable_de_clase
  end
end

instancia = Ejemplo.new
puts "Variable de clase: #{instancia.obtener_variable_de_clase}"
```

En este ejemplo, el método de instancia `obtener_variable_de_clase` puede acceder a la variable de clase `@@variable_de_clase`.

8 Ejercicio 8

8.1 Pregunta

En Ruby, ¿se puede acceder desde un método de instancia a un atributo de instancia de la clase (@) de la misma clase? ¿En qué circunstancias? ¿Qué puede hacerse para conseguir el acceso?

8.2 Respuesta

Sí, en Ruby se puede acceder desde un método de instancia a un atributo de instancia de la misma clase. Los métodos de instancia tienen acceso a los atributos de instancia (variables prefijadas con `@`) de la misma instancia. Aquí hay un ejemplo:

```
class Ejemplo
  def initialize(valor)
    @atributo_de_instancia = valor
  end

  def obtener_atributo_de_instancia
    @atributo_de_instancia
  end
end

instancia = Ejemplo.new(10)
puts "Atributo de instancia: #{instancia.obtener_atributo_de_instancia}"
```

En este ejemplo, el método de instancia `obtener_atributo_de_instancia` puede acceder al atributo de instancia `@atributo_de_instancia`.

8.3 Explicación

Para acceder a un atributo de instancia desde otro método de instancia en la misma clase, simplemente se puede hacer referencia al atributo de instancia utilizando el prefijo `@`. No se necesitan métodos adicionales para este acceso básico. Sin embargo, si se requiere acceder a atributos de instancia desde fuera de la clase, se pueden definir métodos getter y setter públicos.