# Ejercicios: Atributos y Métodos

1. Crear una clase Coche. Para cada coche se almacenará su marca, modelo y matrícula. También se desea una variable que contabilice el número total de coches construidos.
   a) Hacerlo en Java. ¿Has tenido que usar la pseudovariable this en el constructor? Explicar por qué.
   b) Hacerlo en Ruby considerando una **variable de clase** (@@) para el contador de coches
   c) Hacerlo en Ruby considerando un **atributo de instancia de la clase** (@) para el contador de coches.
2. En la clase del ejercicio anterior añadir un método que devuelva un String con el estado del objeto receptor del mensaje. Decidir si el método debe ser de instancia o de clase.
   a) Hacerlo en Java
   b) Hacerlo en Ruby
3. 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.
   a) Hacerlo en Java
   b) Hacerlo en Ruby en las 2 versiones que se han hecho en el ejercicio 1. ¿Has tenido que usar la pseudovarible self en este ejercicio? Explicar por qué.
4. 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.
   a) Hacerlo en Java
   b) Hacerlo en Ruby considerando una **variable de clase** para la constante.
   c) Hacerlo en Ruby considerando un **atributo de instancia de la clase** para la constante.
5. Los siguiente ejemplo en Java es correcto. Fijarse en el uso que se hace de atributos y métodos privados.

```
 1: public class CloudDrive {
 2:   private int capacity;
 3:
 4:   public CloudDrive (int capacity) {
 5:     this.capacity = capacity;
 6:   }
 7:
 8:   private int getCapacity() {
 9:     return capacity;
10:   }
11:
12:   public void setFromOther (CloudDrive other) {
13:     capacity = other.capacity;
14:   }
15:
16:   public void setFromOther_v2 (CloudDrive other) {
17:     capacity = other.getCapacity();
18:   }
```

   Se ha intentado reproducir dicho código en Ruby con una traducción literal como la mostrada en el fragmento siguiente pero lamentablemente presenta errores. Localizar los errores y proponer soluciones.

```
 1: class Cloud_drive
 2:   def initialize (capacity)
 3:     @capacity = capacity
 4:   end
 5:
 6:   private
 7:
 8:   def get_capacity
 9:     return @capacity
10:   end
11:
12:   public
13:
14:   def set_from_other (other)
15:     @capacity = other.@capacity
16:   end
17:
18:   def set_from_other_v2 (other)
19:     @capacity = other.get_capacity
20:   end
```

6. En Java, ¿se puede acceder desde un método de instancia a una variable de clase de la misma clase? ¿En qué circunstancias?
7. En Ruby, ¿se puede acceder desde un método de instancia a una variable de clase (@@) de la misma clase? ¿En qué circunstancias?
8. 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?

1a

```ruby
class Coche
    @num_coches = 0

    attr_accessor :marca, :modelo, :matricula

    def initialize(marca, modelo, matricula)
        @marca = marca
        @modelo = modelo
        @matricula = matricula
        self.class.incrementar_num_coches
    end

    def self.incrementar_num_coches
        @num_coches += 1
    end
    def self.num_coches
        @num_coches
    end

    def to_s
        "Marca: #{@marca}, Modelo: #{@modelo}, Matricula: #{@matricula}"
    end
end

COCHE1 = Coche.new("Ford", "Focus", "1234ABC")
puts "Coche 1: \n"
puts COCHE1.to_s
puts "Coche 1 es una instancia de: #{COCHE1.class}"
puts Coche.num_coches

COCHE2 = Coche.new("Chevrolet", "Corvette", "5678DEF")
puts "Coche 2: \n"
puts COCHE2.to_s
puts "Coche 2 es una instancia de: #{COCHE2.class}"
puts Coche.num_coches
```

16.1c

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

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

    public static int getNumCoches() {
        return numCoches;
    }

    public String getMarca() {
        return marca;
    }

    public String getModelo() {
        return modelo;
    }

    public String getMatricula() {
        return matricula;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }
}
```

2a

```ruby
#forma 1 usando @@ para el contador de los coches
class Coche
    @@num_coches = 0

    attr_accessor :marca, :modelo, :matricula

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

#forma 2 usando @ para el contador de los coches
class Coche
    @num_coches = 0

    attr_accessor :marca, :modelo, :matricula

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

2b

```java
public String toString() {
        return "Marca: " + marca + ", Modelo: " + modelo + ", Matricula: " + matricula;
    }
}
```

# 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.

# Ejercicios: Constructores, consultores y modificadores

1. Crear una clase Coche. Para cada coche se almacenará su marca, modelo, año de construcción y matrícula. Se desea poder instanciar coches de 2 maneras distintas:
   a) Pasando como parámetros una marca, un modelo, un año de construcción y una matrícula.
   b) Pasando como parámetros una marca, un modelo y un año de construcción (para coches que aún no están matriculados).
   c) Hacerlo en Java
   d) Si no lo has hecho así, hacerlo en Java de manera que desde uno de los constructores se reutilice el otro
   e) Hacerlo en Ruby definiendo 2 métodos de clase para ello
   f) Hacerlo en Ruby definiendo parámetros con valores por defecto
   g) Hacerlo en Ruby de una manera distinta a las anteriores
   h) ¿Qué valor has elegido para indicar que un coche no tiene matrícula aún?
2. Añade a la clase anterior un constructor de copia superficial
   a) En Java
   b) En Ruby
3. Crea una clase denominada Flota que tenga un único atributo que sea un vector de Coches. Solo en Java:
   a) Añadirle un constructor sin parámetros
   b) Añadirle un método que permita añadir un Coche a la flota
   c) Añadirle un constructor de copia profunda. ¿Cómo lo harías? ¿Coméntalo y consensúalo con los compañeros?
4. En la clase Flota, añadir un método que reciba un entero (i) y devuelva el Coche i-ésimo de la flota (considerar que el primero es el 0). ¡Ojo!, puedes recibir un entero (i) que esté fuera del rango del vector de Coches, realizar una implementación que no dé ningún error de ejecución si el entero (i) está fuera de rango. ¿Qué vas a hacer si eso ocurre?
   a) Hacer una versión del consultor que devuelva una referencia al Coche
   b) Hacer otra versión del consultor que devuelva una copia del Coche
5. Con respecto a la clase Coche del ejercicio 1:
   a) En JAVA
      - Añadir un consultor que devuelva un String con el estado del objeto consultado
      - Añadir un modificador para actualizar el año de construcción del coche, solo debe permitir poner años mayores a 2000 e inferiores a 2025. Decidir qué hacer si se intenta poner un año fuera de ese rango.
      - Modificar los constructores del ejercicio 1 para no permitir años de construcción fuera del rango indicado en este ejercicio.
   b) En RUBY
      - Repetir en este lenguaje los 3 puntos que has implementado para Java en este ejercicio.
      - Añadir un consultor implícito para la marca
      - Añadir un modificador implícito para el modelo
      - Añadir un consultor/modificador implícito para la matrícula

# Ejercicio 1

Crear una clase `Coche`. Para cada coche se almacenará su marca, modelo, año de construcción y matrícula. Se desea poder instanciar coches de 2 maneras distintas:

a) Pasando como parámetros una marca, un modelo, un año de construcción y una matrícula.

b) Pasando como parámetros una marca, un modelo y un año de construcción (para coches que aún no están matriculados).

c) Hacerlo en Java.

```java
public class Coche {
    private String marca;
    private String modelo;
    private int anio;
    private String matricula;

    // Constructor con matrícula
    public Coche(String marca, String modelo, int anio,
    String matricula) {
        this.marca = marca;
        this.modelo = modelo;
        this.anio = anio;
        this.matricula = matricula;
    }

    // Constructor sin matrícula
    public Coche(String marca, String modelo, int anio) {
        this(marca, modelo, anio, "SIN_MATRICULA");
    }
}
```

d) Si no lo has hecho así, hacerlo en Java de manera que desde uno de los constructores se reutilice el otro.

```java
public Coche(String marca, String modelo, int anio) {
    this(marca, modelo, anio, "SIN_MATRICULA");
}
```

e) Hacerlo en Ruby definiendo 2 métodos de clase para ello.

```ruby
class Coche
    attr_accessor :marca, :modelo, :anio, :matricula

    def initialize(marca, modelo, anio, matricula = "SIN_MATRICULA")
        @marca = marca
        @modelo = modelo
```

```
        @anio = anio
        @matricula = matricula
    end

    def self.con_matricula(marca, modelo, anio, matricula)
        new(marca, modelo, anio, matricula)
    end

    def self.sin_matricula(marca, modelo, anio)
        new(marca, modelo, anio)
    end
end
```

f) Hacerlo en Ruby definiendo parámetros con valores por defecto.

```
def initialize(marca="marca", modelo="modelo", anio="anio",
matricula = "SIN_MATRICULA")
    @marca = marca
    @modelo = modelo
    @anio = anio
    @matricula = matricula
end
```

g) Hacerlo en Ruby de una manera distinta a las anteriores.

```
class Coche
    attr_accessor :marca, :modelo, :anio, :matricula

    def initialize(marca, modelo, anio, matricula = nil)
        @marca = marca
        @modelo = modelo
        @anio = anio
        @matricula = matricula || "SIN_MATRICULA"
    end
end
```

```
    class Coche
        def initialize(*args)
            @marca = args[0]
            @modelo = args[1]
            @anio = args[2]
            @matricula = args[3]
        end
    end
```

h) ¿Qué valor has elegido para indicar que un coche no tiene matrícula aún?

Para indicar que un coche no tiene matrícula aún, he elegido el valor `"SIN_MATRICULA"`.

## Ejercicio 2

Añade a la clase anterior un constructor de copia superficial.

a) En Java.

```java
public Coche(Coche otro) {
    this(otro.marca, otro.modelo, otro.anio, otro.matricula);
}
```

b) En Ruby.

```ruby
class Coche
    def initialize(otro)
        @marca = otro.marca
        @modelo = otro.modelo
        @anio = otro.anio
        @matricula = otro.matricula
    end
end
```

## Ejercicio 3

Crea una clase denominada `Flota` que tenga un único atributo que sea un vector de `Coches`. Solo en Java:

a) Añadirle un constructor sin parámetros.

b) Añadirle un método que permita añadir un `Coche` a la flota.

c) Añadirle un constructor de copia profunda. ¿Cómo lo harías? Coméntalo y consénsualo con los compañeros.

## Solución

Listing 1: Clase Flota con clonación manual

```java
import java.util.Vector;

class Coche {
    private String modelo;
    private int anio;

    // Constructor parametrizado
    public Coche(String modelo, int anio) {
        this.modelo = modelo;
        this.anio = anio;
    }
```

```
12
13      // Método de clonación manual
14      public Coche clonar() {
15          return new Coche(this.modelo, this.anio);
16      }
17
18      // Métodos getters y setters si son necesarios
19  }
20
21  public class Flota {
22      private Vector<Coche> coches;
23
24      // Constructor sin parámetros
25      public Flota() {
26          this.coches = new Vector<Coche>();
27      }
28
29      // Método para añadir un Coche a la flota
30      public void addCoche(Coche coche) {
31          this.coches.add(coche);
32      }
33
34      // Constructor de copia profunda
35      public Flota(Flota otraFlota) {
36          this.coches = new Vector<Coche>();
37          for (Coche coche : otraFlota.coches) {
38              this.coches.add(coche.clonar());
39          }
40      }
41  }
```

## Ejercicio 4

En la clase `Flota`, añadir un método que reciba un entero $i$ y devuelva el Coche $i$-ésimo de la flota. Implementar esto en dos versiones: una que devuelva una referencia al Coche y otra que devuelva una copia del Coche.

### Solución en Java

```
1   public class Flota {
2       private List<Coche> coches;
3
4       public Flota() {
5           coches = new ArrayList<>();
6       }
7
8       // Añadir coche a la flota
9       public void agregarCoche(Coche coche) {
10          coches.add(coche);
11      }
12
13      // Devuelve una referencia al Coche
14      public Coche obtenerCoche(int i) {
```

```
        if (i >= 0 && i < coches.size()) {
            return coches.get(i);
        } else {
            return null; // o lanzar una excepci n
        }
    }

    // Devuelve una copia del Coche
    public Coche obtenerCocheCopia(int i) {
        if (i >= 0 && i < coches.size()) {
            return new Coche(coches.get(i));
        } else {
            return null; // o lanzar una excepci n
        }
    }
}
```

### Solución en Ruby

```
class Flota
    def initialize
        @coches = []
    end

    # A adir coche a la flota
    def agregar_coche(coche)
        @coches << coche
    end

    # Devuelve una referencia al Coche
    def obtener_coche(i)
        if i >= 0 && i < @coches.size
            @coches[i]
        else
            nil # o lanzar una excepci n
        end
    end

    # Devuelve una copia del Coche
    def obtener_coche_copia(i)
        if i >= 0 && i < @coches.size
            Coche.new(@coches[i])
        else
            nil # o lanzar una excepci n
        end
    end
end
```

## Ejercicio 5

Con respecto a la clase `Coche` del ejercicio 1:

- Añadir un consultor que devuelva un String con el estado del objeto.

- Añadir un modificador para actualizar el año de construcción del coche, solo permitiendo años mayores a 2000 e inferiores a 2025.
- Modificar los constructores del ejercicio 1 para no permitir años de construcción fuera del rango indicado.

## Solución en Java

```java
public class Coche {
    private String marca;
    private String modelo;
    private int anio;
    private String matricula;

    // Constructor con matr cula
    public Coche(String marca, String modelo, int anio,
    String matricula) {
        if (anio < 2000 || anio > 2025) {
            throw new IllegalArgumentException("El  a  o debe estar
            entre 2000 y 2025");
        }
        this.marca = marca;
        this.modelo = modelo;
        this.anio = anio;
        this.matricula = matricula;
    }

    // Constructor sin matr cula
    public Coche(String marca, String modelo, int anio) {
        this(marca, modelo, anio, "SIN_MATRICULA");
    }

    // Consultor que devuelve el estado del objeto
    @Override
    public String toString() {
        return "Coche[marca=" + marca + ", modelo=" +
        modelo + ", anio=" + anio + ", matricula=" +
        matricula + "]";
    }

    // Modificador para actualizar el  a  o de construcci n
    public void setAnio(int anio) {
        if (anio < 2000 || anio > 2025) {
            throw new IllegalArgumentException("El  a  o debe
            estar entre 2000 y 2025");
        }
        this.anio = anio;
    }
}
```

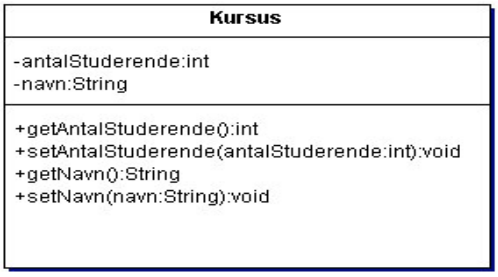
## Solución en Ruby

```ruby
class Coche
```

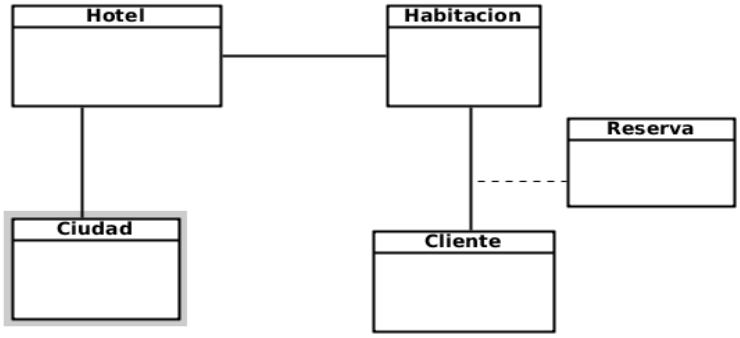
```
    attr_accessor :marca
    attr_reader :modelo, :matricula, :anio

    def initialize(marca, modelo, anio, matricula = "SIN_MATRICULA")
        if anio < 2000 || anio > 2025
            raise "El a o debe estar entre 2000 y 2025"
        end
        @marca = marca
        @modelo = modelo
        @anio = anio
        @matricula = matricula
    end

    # Consultor que devuelve el estado del objeto
    def to_s
        "Coche[marca=#{@marca}, modelo=#{@modelo},
        anio=#{@anio}, matricula=#{@matricula}]"
    end

    # Modificador para actualizar el a o de construcci n
    def anio=(anio)
        if anio < 2000 || anio > 2025
            raise "El a o debe estar entre 2000 y 2025"
        end
        @anio = anio
    end
end
```

# Ejercicios: Elementos de agrupación y diagramas estructurales

1. UML es un lenguaje “universal”. Escribe el código Java y Ruby correspondiente a la declaración de la siguiente clase en danés (incluyendo la declaración de atributos y la cabecera de los métodos).

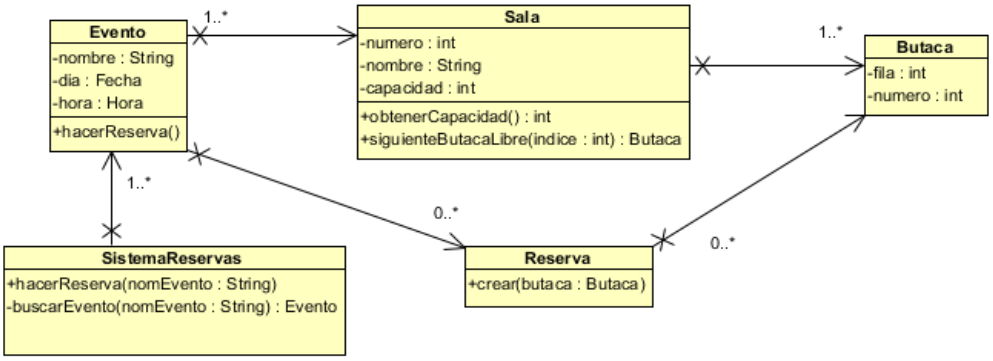



2. El siguiente diagrama de clases representa hoteles con sus habitaciones y las reservas hechas por sus clientes:



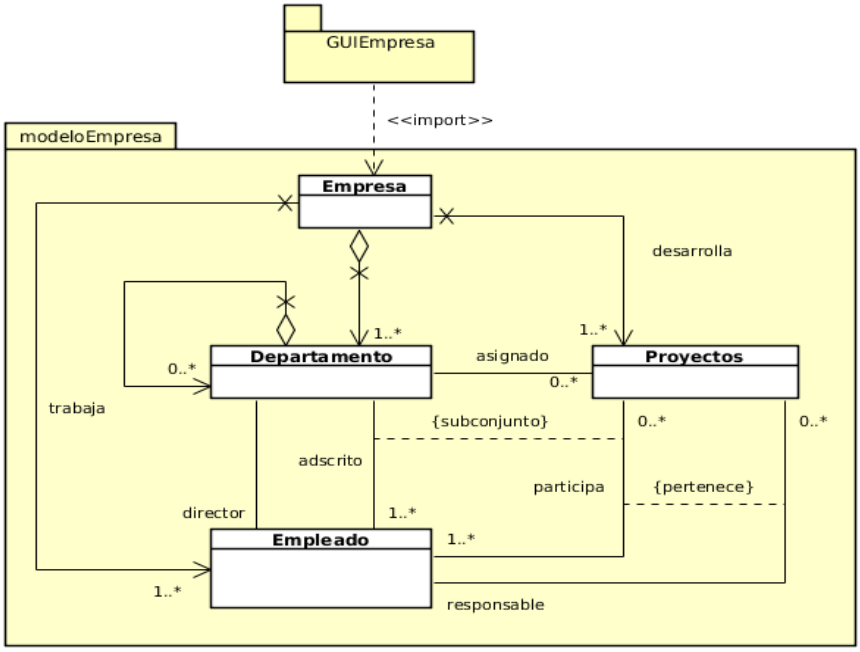

   a) Complétalo con los atributos de las clases y el nombre, roles, multiplicidad y navegabilidad de las asociaciones, haciendo las suposiciones que consideres oportunas.
   b) Implementa en Java el resultado obtenido.
   c) Implementa en Ruby el resultado obtenido.

3. A partir del siguiente diagrama de clases.



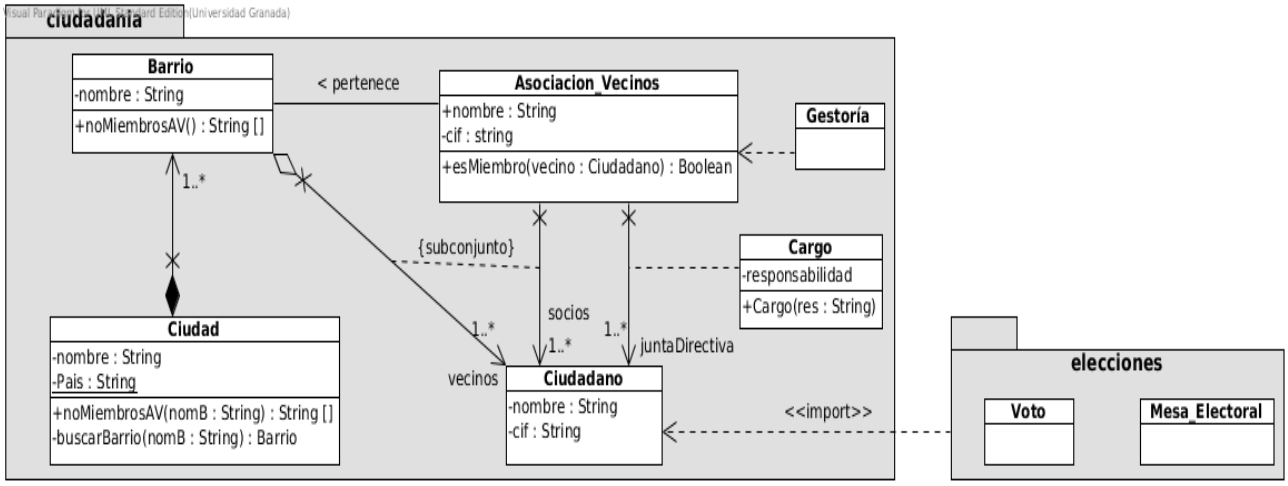

   a) ¿La asociación que existe entre Sala y Butaca podría ser una agregación?¿Y una composición?
   b) Partiendo de una sala, ¿podría saberse para qué eventos está siendo usada?
   c) Escribe el código Java correspondiente.
   d) Escribe el código Ruby correspondiente.

4. A partir del siguiente diagrama de clases, responde a las cuestiones plateadas:




a) ¿Qué son GUIEmpresa y modeloEmpresa?
b) La relación que hay entre GUIEmpresa y Empresa ¿de qué tipo es?, es decir ¿qué significa <<import>> sobre ella?
c) Desde las clases de GUIEmpresa, en Java, se puede instanciar cualquier clase de modeloEmpresa? ¿Cómo?
d) ¿Qué representa la línea discontinua entre las asociaciones adscrito y participa y cuál es su significado en este diagrama?
e) Hay dos asociaciones entre Proyectos y Empleado, ¿qué representan?
f) ¿Por qué hay 0..* y no 1..* en la multiplicidad de la asociación "participa" entre Empleado y Proyectos? ¿Cómo expresarías en lenguaje natural lo que representa ese 0?

5. Partiendo del siguiente diagrama de clases de UML, resolver las cuestiones planteadas a continuación.



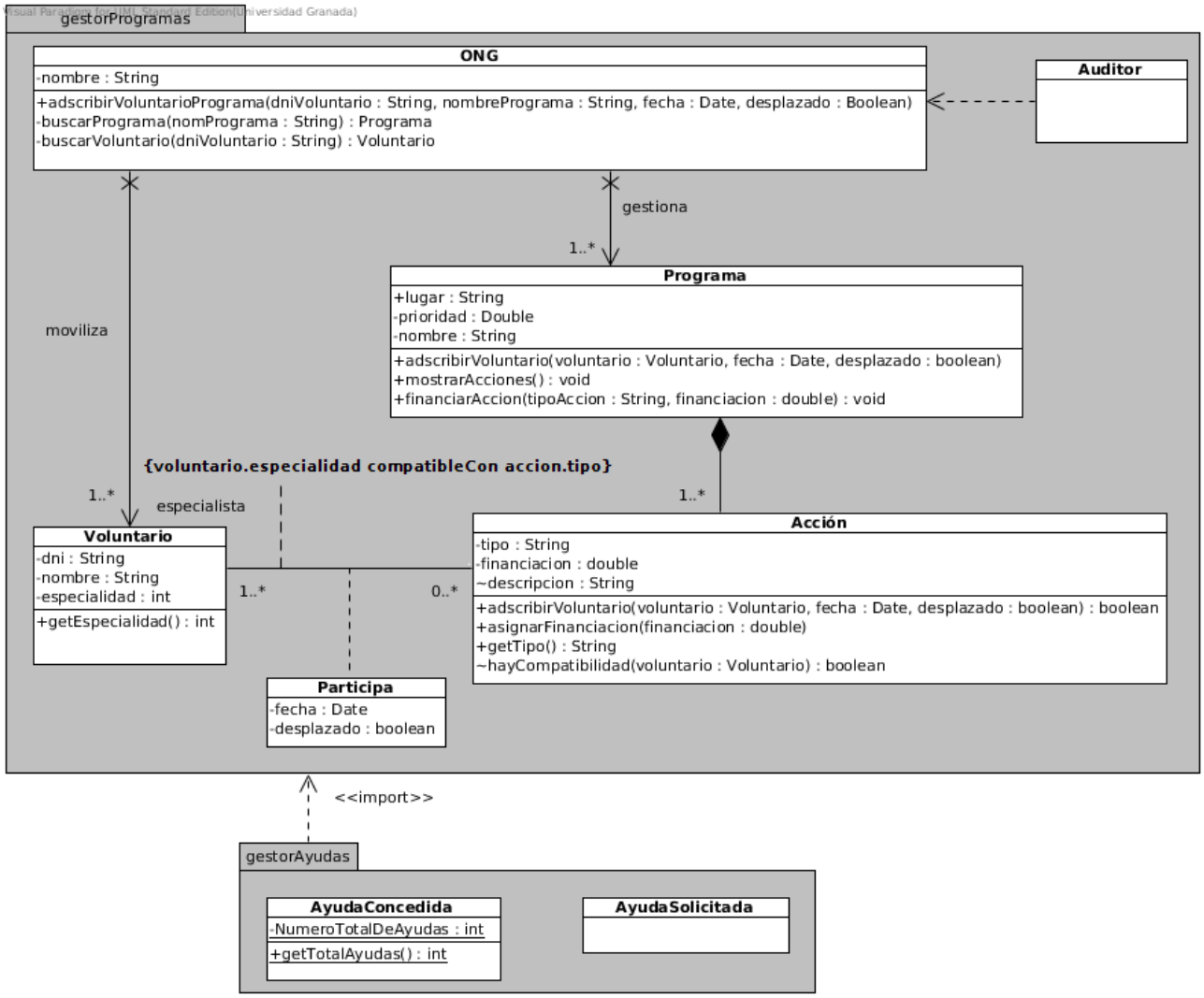

a) ¿Desde un objeto de la clase Ciudadano puede saberse si es presidente de la asociación de vecinos de su barrio?¿Por qué?
b) ¿De qué tipo es la relación entre Ciudad y Barrio? ¿qué significa?
c) ¿De qué tipo es la relación entre Barrio y Ciudadano? ¿qué significa? ¿podría ser de otro tipo?

d) Si la junta directiva de una asociación de vecinos está formada por exactamente 6 ciudadanos, ¿cómo lo indicarías en el diagrama de clases?

e) ¿Qué significa que la clase Cargo esté ligada a la asociación entre Asociacion_Vecinos y Ciudadano? ¿De qué otra forma podrías representar en un diagrama de clases las clases Asociacion_Vecinos, Cargo y Ciudadano?

f) Define en Java los atributos de referencia de la clase Barrio.

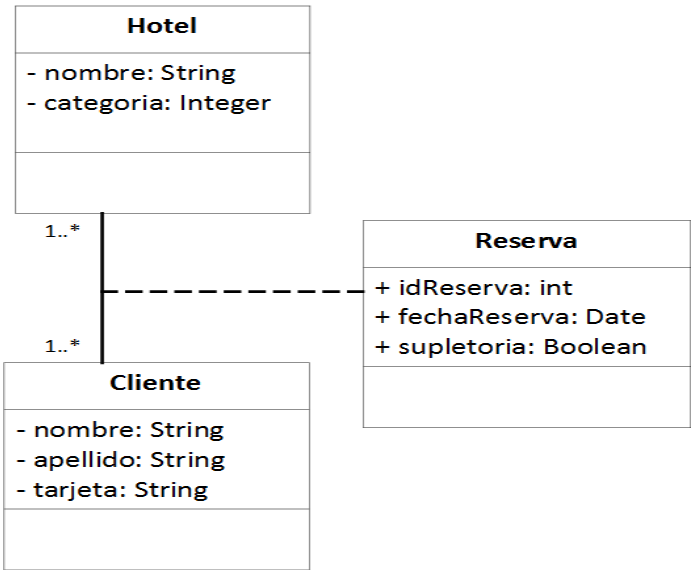
6. Partiendo del siguiente diagrama de clases de UML, responde verdadero (V) o falso (F) a las cuestiones:




| | |
|---|---|
| Desde la clase AyudaSolicitada se puede acceder a todos los elementos públicos del paquete GestorProgramas | |
| Un voluntario puede participar en cualquier acción de un programa sin ningún tipo de restricción | |
| El estado de un objeto de la clase Auditor viene determinado por el estado de un objeto de la clase ONG | |
| Un voluntario podría participar en acciones de distintos programas | |
| Un voluntario puede pertenecer a varias ONG | |
| Cuando se define un objeto de la clase Acción, éste tiene que asociarse a un determinado objeto de la clase Programa | |
| En una acción puede participar más de un voluntario como especialista | |

| | |
|---|---|
| Desde un objeto de la clase ONG se puede llegar a conocer a todos los especialistas de una determinada acción en un programa | |
| El estado de un objeto Voluntario está exclusivamente determinado por su dni, nombre y especialidad | |
| Todos los métodos de la clase Acción pueden ser accedidos desde la clases AyudaConcedida | |

7. Teniendo en cuenta el siguiente diagrama de clases, responde si las afirmaciones son verdaderas (V) o falsas (F)




| | |
|---|---|
| Hotel tendrá dos atributos de referencia, uno relativo a las reservas y otro relativo a los clientes | |
| Reserva tendrá los atributos de referencia:<br>`private Hotel hotel;`<br>`private Cliente cliente;` | |

# Ejercicio 1

**Enunciado:**

UML es un lenguaje "universal". Escribe el código Java y Ruby correspondiente a la declaración de la siguiente clase en danés (incluyendo la declaración de atributos y la cabecera de los métodos):

```
Kursus
- antalStuderende: int
- navn: String

+ getAntalStuderende(): int
+ setAntalStuderende(antalStuderende: int): void
+ getNavn(): String
+ setNavn(navn: String): void
```

Listing 1: Clase Kursus en Java (la implentación de los métodos no es necesaria)

**Solución en Java:**

```java
public class Kursus {
    private int antalStuderende;
    private String navn;

    public int getAntalStuderende() {
        return antalStuderende;
    }

    public void setAntalStuderende(int antalStuderende) {
        this.antalStuderende = antalStuderende;
    }

    public String getNavn() {
        return navn;
    }

    public void setNavn(String navn) {
        this.navn = navn;
    }
}
```

Listing 2: Clase Kursus en Java

**Solución en Ruby:**

```ruby
class Kursus
  attr_accessor :antal_studerende, :navn

  def initialize(antal_studerende, navn)
    @antal_studerende = antal_studerende
    @navn = navn
  end
end
```

Listing 3: Clase Kursus en Ruby

**Como podemos ver en el diagrama de clase**, las funciones que aparecen son getters y setters, por lo que la función del constructor es para evitar dejar de lado la estructura adecuada y/o coherente de los ficheros de lenguaje acorde a la teoría.

# Ejercicio 2

**Enunciado:**

El siguiente diagrama de clases representa hoteles con sus habitaciones y las reservas hechas por sus clientes:

```
Hotel-----------Habitacion
  |                 |
  |                 |
  |                 |
Ciudad            Reserva
                    |
                    |
                 Cliente
```

Listing 4: Diagrama de Clases

a) Complétalo con los atributos de las clases y el nombre, roles, multiplicidad y navegabilidad de las asociaciones, haciendo las suposiciones que consideres oportunas.

b) Implementa en Java el resultado obtenido.

c) Implementa en Ruby el resultado obtenido.

**Solución:**

a) Diagrama de Clases Completado:

```
Hotel           Habitacion
- nombre         - numero
- direccion      - tipo
- List de Hab       |
- ciudad            |
  |                 |
  |                 |
  |                 |
Ciudad            Reserva
- nombre           - fechaInicio
- pais             - fechaFin
                    |
                    |
                 Cliente
                 - nombre
                 - dni
```

Listing 5: Diagrama de Clases Completado

b) Implementación en Java:

```java
import java.util.ArrayList;
import java.util.List;

class Hotel {
    private String nombre;
    private String direccion;
    private List<Habitacion> habitaciones;

    public Hotel(String nombre, String direccion) {
        this.nombre = nombre;
        this.direccion = direccion;
        this.habitaciones = new ArrayList<>();
    }

    // M todos getters y setters
    //suponemos que la direccion contiene a la ciudad, en caso contrario un implementaci n acorde seria:

    private String nombre;
    private String direccion;
    private List<Habitacion> habitaciones;
    private Ciudad ciudad;

    public Hotel(String nombre, String direccion, Ciudad ciudad) {
        this.nombre = nombre;
        this.direccion = direccion;
        this.habitaciones = new ArrayList<>();
        this.ciudad=ciudad;
        //deber amos de a adir lo nuevo al diagrama del apartado a
    }

}

class Habitacion {
    private int numero;
    private String tipo;

    public Habitacion(int numero, String tipo) {
        this.numero = numero;
        this.tipo = tipo;
    }

    // M todos getters y setters
}

class Ciudad {
    private String nombre;
    private String pais;

    public Ciudad(String nombre, String pais) {
        this.nombre = nombre;
        this.pais = pais;
    }

    // M todos getters y setters
}
```

```java

class Reserva {
    private String fechaInicio;
    private String fechaFin;
    private Cliente cliente; //no se incluye en el diagrama del
    apartado a porque podemos considerar que viene inclusive o se
    deduce de la relaci n (LO MISMO PASAR A CON HOTEL)
    //Consideramos la opcion de asociar a una habitacion un hotel

    public Reserva(String fechaInicio, String fechaFin, Cliente
    cliente) {
        this.fechaInicio = fechaInicio;
        this.fechaFin = fechaFin;
        this.cliente = cliente;
    }
    //tambien podemos considerar la opcion de asociar a una reserva
     un cliente

    // M todos getters y setters

}

class Cliente {
    private String nombre;
    private String dni;

    public Cliente(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
    }

    // M todos getters y setters
}
```

Listing 6: Implementación en Java

c) Implementación en Ruby:

```ruby
class Hotel
  attr_accessor :nombre, :direccion, :habitaciones

  def initialize(nombre, direccion)
    @nombre = nombre
    @direccion = direccion
    @habitaciones = []
  end
end

class Habitacion
  attr_accessor :numero, :tipo

  def initialize(numero, tipo)
    @numero = numero
    @tipo = tipo
  end
end

class Ciudad
```

```ruby
  attr_accessor :nombre, :pais

  def initialize(nombre, pais)
    @nombre = nombre
    @pais = pais
  end
end

class Reserva
  attr_accessor :fecha_inicio, :fecha_fin, :cliente

  def initialize(fecha_inicio, fecha_fin, cliente)
    @fecha_inicio = fecha_inicio
    @fecha_fin = fecha_fin
    @cliente = cliente
  end
end

class Cliente
  attr_accessor :nombre, :dni

  def initialize(nombre, dni)
    @nombre = nombre
    @dni = dni
  end
end
```

Listing 7: Implementación en Ruby

Uno de los porques de estas implementaciones puede ser, debido a la teoría **hotel** y **habitacion** no son padre e hijo, son hermanos, por lo que no debemos de incluirlos, mientras que **reserva** esta incluida con líneas discontinuas que sabemos que eso quiere decir que puede o no tenerla. Por último, podemos tener en cuenta que se incluye en **dirección** la **ciudad**.

# Ejercicio 3

a) ¿La asociación que existe entre Sala y Butaca podría ser una agregación? ¿Y una composición?

b) Partiendo de una sala, ¿podría saberse para qué eventos está siendo usada?

c) Escribe el código Java correspondiente.

d) Escribe el código Ruby correspondiente.

# Solución

a) **Agregación vs Composición:**

La relación entre Sala y Butaca podría ser una agregación o una composición dependiendo del contexto:

- **Agregación:** Si una butaca puede existir sin estar necesariamente asociada a una sala (por ejemplo, una butaca se puede mover de una sala a otra). Yo diría que si puede existir ya que si puede existir una butaca que no necesariamente este asociada a una sala. - **Composición:** Si una butaca no puede existir sin una sala (por ejemplo, las butacas se crean y se destruyen junto con la sala). Es que depende del contexto en este caso no, debido a lo explicado en el anterior punto, pero si se antepone esa condición en el contexto, entonces sí.

b) **Relación entre Sala y Eventos:**

Si una sala está reservada para varios eventos, se podría rastrear esta información a través de las reservas hechas en el sistema. Cada reserva tendría información sobre la sala y el evento asociado.

c) **Código en Java:**

```java
import java.util.ArrayList;
import java.util.List;

// Clase Evento
class Evento {
    private String nombre;
    private Fecha dia;
    private Hora hora;
    private List<Sala> salas;// se puede usar listas o arrays
    private List<Reserva> reservas;

    public Evento(String nombre, Fecha dia, Hora hora) {
        this.nombre = nombre;
        this.dia = dia;
        this.hora = hora;
        this.salas = new ArrayList<>();
        this.reservas = new ArrayList<>();
    }

    //constructor para la clase SistemaReservas
    public Evento(string nombre){
        Fecha dia = null;
        Hora hora = null;
        Evento(nombre,dia,hora);
    }

    //podemos a adir otros constructores con solo unos parametros, como puede ser pas ndole solo el tama o de alguna de las listas

    public void hacerReserva() {
        Butaca butacaLibre=null;
        // Buscar la primera sala disponible con butacas libres
        for (Sala sala : salas) {
            butacaLibre = sala.siguienteButacaLibre();
            if (butacaLibre != null) {
                // Ocupar la butaca
                butacaLibre.ocupar();
                Reserva r = new Reserva(sala,this); //this hace referencia a este evento
                reservas.add(r);
                break;
```

```
40
41              }
42          }
43
44          // Si no hay butacas libres, no se puede hacer la reserva
45          if (butacaLibre == null ) System.out.println("No hay salas
        con butacas disponibles.");
46      }
47
48
49      // M todos getters y setters
50  }
51
52  // Clase Sala
53  class Sala {
54      private String nombre;
55      private int numero;
56      private int capacidad;
57      private List<Butaca> butacas;
58
59      public Sala(String nombre, int numero, int capacidad) {
60          this.nombre = nombre;
61          this.numero = numero;
62          this.capacidad = capacidad;
63          this.butacas = new ArrayList<>();
64      }
65
66      public void addButaca(Butaca butaca) {
67          butacas.add(butaca);
68      }
69
70      public int obtenerCapacidad() {
71          return capacidad;
72      }
73
74      public Butaca siguienteButacaLibre() {
75          for (Butaca butaca : butacas) {
76              if (butaca.isEmpty()) {
77                  return butaca;
78              }
79          }
80          return null; // No hay butacas libres
81      }
82
83      // M todos getters y setters
84  }
85
86  // Clase Butaca
87  class Butaca {
88      private int numero;
89      private int fila;
90      private boolean empty;
91
92      public Butaca(int numero, int fila) {
93          this.numero = numero;
94          this.fila = fila;
95          this.empty = true; // Por defecto est  libre
```

```
    }

    public boolean isEmpty() {
        return empty;
    }

    public void ocupar() {
        this.empty = false;
    }
}

// Clase SistemaReservas
class SistemaReservas {
    private List<Evento> eventos;

    public SistemaReservas() {
        this.eventos = new ArrayList<>();
    }

    public void hacerReserva(String nombreEvento) {
        Evento nuevo = new Evento(nombreEvento);
        eventos.add(nuevo);
    }

    public Evento buscarEvento(String nomEvento) {
        for (Evento evento : eventos) {
            if (evento.getNombre().equals(nomEvento)) {
                return evento;
            }
        }
        return null; // No se encontr  el evento
    }
}

// Clase Reserva
class Reserva {
    private Sala sala;
    private Evento evento;

    public Reserva(Sala sala, Evento evento) {
        this.sala = sala;
        this.evento = evento;
    }//podemos pensar en la opcion de incluir una clase cliente,
    que sea la persona a al que se le otorga la butaca

    // M todos getters y setters
}

// Clases auxiliares
class Fecha {
    // suponemos que ya esta implementada
}

class Hora {
    // suponemos que ya esta implementada
```

```
}
```

Listing 8: Implementación en Java

Esto estaría implementado de manera que cada clase tendría su propio fichero.

d) **Código en Ruby:**

```ruby
# Clase Evento
class Evento
  attr_accessor :nombre, :dia, :hora, :salas, :reservas

  def initialize(nombre, dia = nil, hora = nil)
    @nombre = nombre
    @dia = dia
    @hora = hora
    @salas = new Array
    @reservas = new Array
  end

     def hacer_reserva
      nueva_reserva = nil # Variable local para rastrear la reserva

      @salas.each do |sala|
        butaca_libre = sala.siguiente_butaca_libre
        if butaca_libre
          # Ocupar la butaca
          butaca_libre.ocupar

          # Crear y registrar la reserva
          nueva_reserva = Reserva.new(sala, self)
          @reservas.push(nueva_reserva)

          # Detener la b squeda una vez encontrada una sala
    disponible
          break
        end
      end

      # Informar si no se encontr  butaca libre
      if nueva_reserva.nil?
        puts "No hay salas con butacas disponibles."
      else
        puts "Reserva realizada con  xito ."
      end

      # No retornar nada (impl cito) para que sea void
      nil
    end
end

# Clase Sala
class Sala
  attr_accessor :nombre, :numero, :capacidad, :butacas

  def initialize(nombre, numero, capacidad)
    @nombre = nombre
    @numero = numero
```

```ruby
    @capacidad = capacidad
    @butacas = Array.new
  end

  def add_butaca(butaca)
    butacas.push(butaca)
  end

  def obtener_capacidad
    @capacidad
  end

  def siguiente_butaca_libre
    @butacas.find { |butaca| butaca.empty? } //creo que es as
  end
end

# Clase Butaca
class Butaca
  attr_accessor :numero, :fila, :empty

  def initialize(numero, fila)
    @numero = numero
    @fila = fila
    @empty = true # Por defecto est  libre
  end

  def ocupar
    @empty = false
  end

  def empty?
    @empty
  end
end

# Clase SistemaReservas
class SistemaReservas
  attr_accessor :eventos

  def initialize
    @eventos = Array.new
  end

  def hacer_reserva(nombre_evento)
    nuevo = Evento.new(nombre_evento)
    @eventos.push(nuevo)
  end

  def buscar_evento(nombre_evento)
    @eventos.find { |evento| evento.nombre == nombre_evento }
  end
end

# Clase Reserva
class Reserva
  attr_accessor :sala, :evento
```

```ruby
  #podemos incluir tambien la clase cliente para m s coherencia

  def initialize(sala, evento)
    @sala = sala
    @evento = evento
  end
end

# Clases auxiliares
class Fecha
  # suponer que esta implementada
end

class Hora
  # suponer que esta implementada
end
```

Listing 9: Implementación en Ruby

4. A partir del siguiente diagrama de clases, responde a las cuestiones planteadas:

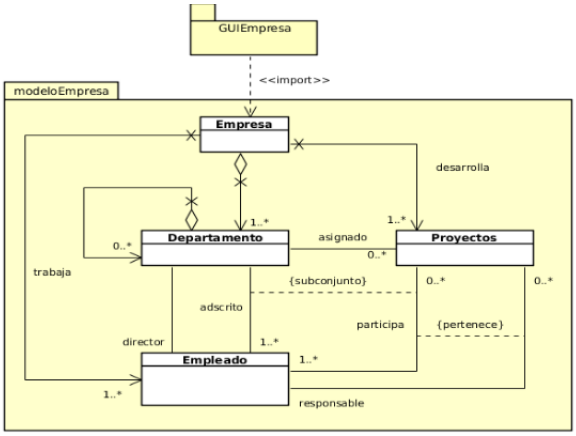



Figure 1: Diagrama de clases.

a) **¿Qué son GUIEmpresa y modeloEmpresa?**
GUIEmpresa: Se trata de otro paquete. modeloEmpresa: del nombre de clase.

b) **La relación que hay entre GUIEmpresa y Empresa ¿de qué tipo es?, es decir ¿qué significa ¡¡import¿¿ sobre ella?**
Quiere decir que se va a usar el paquete GUIEmpresa en la clase modeloEmpresa y para ello debe de importarlo.

c) **Desde las clases de GUIEmpresa, en Java, se puede instanciar cualquier clase de modeloEmpresa? ¿Cómo?**

- Sí, desde las clases de **GUIEmpresa** se puede instanciar cualquier clase de **modeloEmpresa**. Esto se hace importando las clases de **modeloEmpresa** en **GUIEmpresa** y creando instancias según sea necesario. Por ejemplo:

```
import modeloEmpresa.Empresa;
public class GUIEmpresa {
    public static void main(String[] args) {
    Empresa empresa = new Empresa(); // Operaciones con la
    instancia de Empresa
    }
}
```

Listing 10: Instanciación de clase de modeloEmpresa en GUIEmpresa

d) **¿Qué representa la línea discontinua entre las asociaciones adscrito y participa y cuál es su significado en este diagrama?**
La línea discontinua representa una **dependencia** entre las asociaciones **adscrito** y **participa**. Significa que la asociación **participa** depende de la asociación **adscrito** en cierto sentido, probablemente indicando que un empleado debe estar adscrito a un departamento antes de poder participar en un proyecto.

e) **Hay dos asociaciones entre Proyectos y Empleado, ¿qué representan?** - Las dos asociaciones entre **Proyectos** y **Empleado** representan distintas formas de interacción:

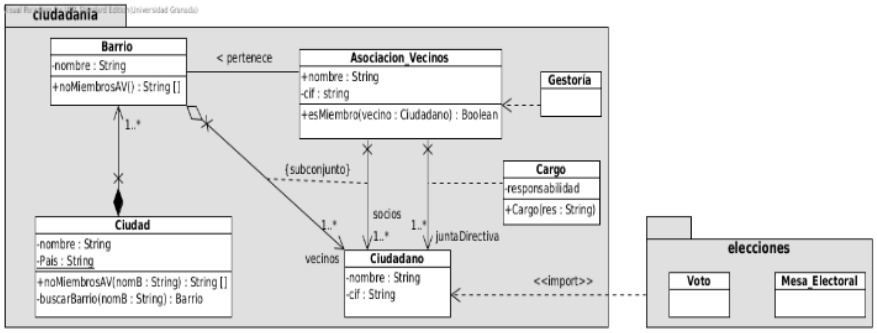
- participa: uno o varios empleados pueden participar en múltiples proyectos o en ninguno.
- Un empleado es responsable de múltiples proyectos o de ninguno.

f) **¿Por qué hay 0..* y no 1..* en la multiplicidad de la asociación "participa" entre Empleado y Proyectos? ¿Cómo expresarías en lenguaje natural lo que representa ese 0?**
El **0..*** en la multiplicidad de la asociación **participa** indica que un empleado puede no participar en ningún proyecto (0 proyectos) o puede participar en varios proyectos (* proyectos). Esto expresa la flexibilidad de que un empleado no está obligado a participar en proyectos. En lenguaje natural: "Un empleado puede no estar involucrado en ningún proyecto o puede estar involucrado en varios proyectos."

# Ejercicio 5

**Enunciado:**

Partiendo del siguiente diagrama de clases de UML, resolver las cuestiones planteadas a continuación.




a) ¿Desde un objeto de la clase Ciudadano puede saberse si es presidente de la asociación de vecinos de su barrio? ¿Por qué?

**Respuesta:** No, desde un objeto de la clase Ciudadano no se puede saber directamente si es presidente de la asociación de vecinos de su barrio porque la clase Ciudadano no tiene un atributo o método que almacene o verifique esta información.

b) ¿De qué tipo es la relación entre Ciudad y Barrio? ¿qué significa?

**Respuesta:** La relación entre Ciudad y Barrio es una relación de composición, lo que significa que una ciudad está compuesta de varios barrios y que los barrios no pueden existir sin la ciudad.

c) ¿De qué tipo es la relación entre Barrio y Ciudadano? ¿qué significa? ¿podría ser de otro tipo?

**Respuesta:** La relación entre Barrio y Ciudadano es una relación de agregación, lo que significa que un barrio puede tener varios ciudadanos y los ciudadanos pueden existir independientemente del barrio. Podría ser de otro tipo si se quisiera indicar, por ejemplo, que un ciudadano solo puede pertenecer a un barrio a la vez o en el caso de que se quiera decir que es de composición, pero en este caso si lo trasladamos a la vida real, no tiene sentido que un ciudadano no exista si no pertenece a un barrio.

d) Si la junta directiva de una asociación de vecinos está formada por exactamente 6 ciudadanos, ¿cómo lo indicarías en el diagrama de clases?

**Respuesta:** Se indicaría con una multiplicidad de 6 en la relación entre Ciudadano y Asociacion_Vecinos:

```
1     Asociacion_Vecinos
2   +nombre: String
```

```
  +junta: Ciudadano[6]
```

e) ¿Qué significa que la clase Cargo esté ligada a la asociación entre Asociacion_Vecinos y Ciudadano? ¿De qué otra forma podrías representar en un diagrama de clases las clases Asociacion_Vecinos, Cargo y Ciudadano?

**Respuesta:** Que la clase Cargo esté ligada a la asociación entre Asociacion_Vecinos y Ciudadano significa que los cargos están asignados a los ciudadanos en el contexto de su asociación vecinal, es decir, que un ciudadano puede tener una responsabilidad en la asociación de vecinos. Otra forma de representarlo sería mediante una relación de composición entre Asociacion_Vecinos y Cargo, y una relación de asociación entre Cargo y Ciudadano:

- Incluir las siguientes modificaciones:

```
Asociacion_Vecinos
  +nombre: String
  +cargos: Cargo[*]

Cargo
  +nombre: String
  +ocupante: Ciudadano
```

- O poner en el diagrama de clases la clase cargo en medio de las dos clases con únion de flechas continuas.

f) Define en Java los atributos de referencia de la clase Barrio.

**Solución en Java:**

```java
class Barrio {
    private String nombre;
    private Ciudad ciudad;
    private List<Ciudadano> ciudadanos;
    private AsociacionVecinos asociacionVecinos;

    // Constructor, getters y setters
}
```

Listing 11: Definición de la clase Barrio en Java

# Ejercicio 6

**Enunciado:**

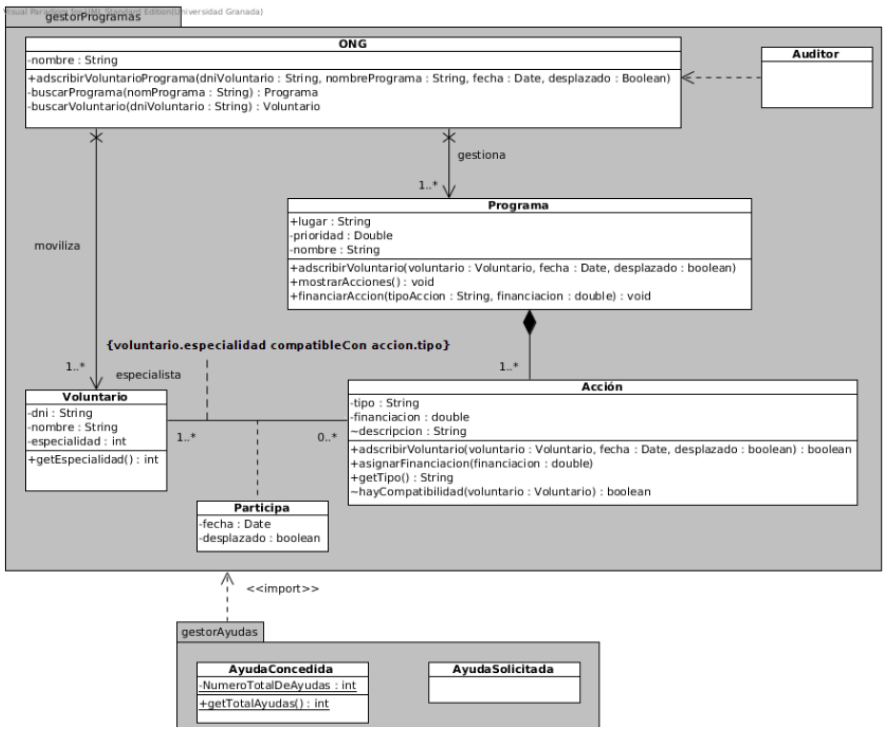



Figure 2: Caption

Partiendo del siguiente diagrama de clases de UML, responde verdadero (V) o falso (F) a las cuestiones:

Verdader o Falso

1. Desde la clase **AyudaSolicitada** se puede acceder a todos los elementos públicos del paquete **GestorProgramas**.
2. Un voluntario puede participar en cualquier acción de un programa sin ningún tipo de restricción.
3. El estado de un objeto de la clase **Auditor** viene determinado por el estado de un objeto de la clase **ONG**.
4. Un voluntario podría participar en acciones de distintos programas.
5. Un voluntario puede pertenecer a varias **ONG**.
6. Cuando se define un objeto de la clase **Acción**, éste tiene que asociarse a un determinado objeto de la clase **Programa**.
7. En una acción puede participar más de un voluntario como especialista.
8. Desde un objeto de la clase **ONG** se puede llegar a conocer a todos los especialistas de una determinada acción en un programa.
9. El estado de un objeto **Voluntario** está exclusivamente determinado por su `dni`, `nombre` y `especialidad`.
10. Todos los métodos de la clase **Acción** pueden ser accedidos desde la clase **AyudaConcedida**.

## Respuestas y Justificaciones

1) Desde la clase **AyudaSolicitada** se puede acceder a todos los elementos públicos del paquete **GestorProgramas**: **Falso (F)**

   **Justificación:** El acceso a elementos públicos de un paquete depende de la visibilidad y alcance definido en el mismo. No necesariamente todas las clases tienen acceso a todos los elementos públicos de otro paquete sin importar su relación.

2) Un voluntario puede participar en cualquier acción de un programa sin ningún tipo de restricción: **Falso (F)**

   **Justificación:** Los voluntarios pueden tener restricciones basadas en su rol, habilidades o en las necesidades específicas del programa, por lo que no pueden participar en cualquier acción sin restricciones.

3) El estado de un objeto de la clase **Auditor** viene determinado por el estado de un objeto de la clase **ONG**: **Falso (F)**

**Justificación:** Aunque el Auditor puede estar asociado a la ONG, su estado puede ser independiente y determinado por sus propios atributos y métodos.

4) Un voluntario podría participar en acciones de distintos programas: **Verdadero (V)**

   **Justificación:** Es posible que un voluntario tenga la capacidad de participar en múltiples programas, especialmente si tiene habilidades o experiencias relevantes para diferentes acciones.

5) Un voluntario puede pertenecer a varias ONG: **Verdadero (V)**

   **Justificación:** No hay una restricción inherente que impida que un voluntario sea miembro de múltiples organizaciones no gubernamentales (ONG).

6) Cuando se define un objeto de la clase **Acción**, éste tiene que asociarse a un determinado objeto de la clase **Programa**: **Verdadero (V)**

   **Justificación:** Según el diagrama de clases, una acción forma parte de un programa, por lo que debe estar asociada a un programa específico al ser definida.

7) En una acción puede participar más de un voluntario como especialista: **Verdadero (V)**

   **Justificación:** Una acción puede requerir múltiples especialistas en diferentes áreas, permitiendo así la participación de varios voluntarios en la misma acción.

8) Desde un objeto de la clase **ONG** se puede llegar a conocer a todos los especialistas de una determinada acción en un programa: **Verdadero (V)**

   **Justificación:** La clase ONG puede gestionar y conocer todos los especialistas involucrados en sus acciones y programas, teniendo acceso a esta información.

9) El estado de un objeto **Voluntario** está exclusivamente determinado por su dni, nombre y especialidad: **Falso (F)**

   **Justificación:** El estado de un voluntario puede depender de más factores además de su dni, nombre y especialidad, como su disponibilidad, ubicación, entre otros atributos.

10) Todos los métodos de la clase **Acción** pueden ser accedidos desde la clase **AyudaConcedida**: **Falso (F)**

    **Justificación:** El acceso a métodos depende de su visibilidad (público, protegido, privado). No todos los métodos de la clase Acción serán necesariamente accesibles desde AyudaConcedida.

# Ejercicio 7

## Enunciado:

Teniendo en cuenta el siguiente diagrama de clases, responde si las afirmaciones son verdaderas o falsas.

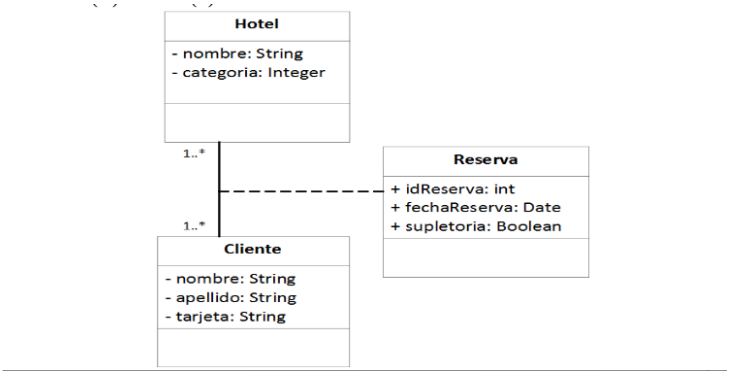



Figure 3: Diagrama de Clases UML para el sistema de reservas de hotel

## Clases y Atributos

- **Hotel**
  - `nombre: String`
  - `categoria: Integer`
- **Cliente**
  - `nombre: String`
  - `apellido: String`
  - `tarjeta: String`
- **Reserva**
  - `idReserva: int`
  - `fechaReserva: Date`
  - `supletoria: Boolean`

## Respuesta

**Enunciado del Ejercicio**

Resuelve el siguiente problema:

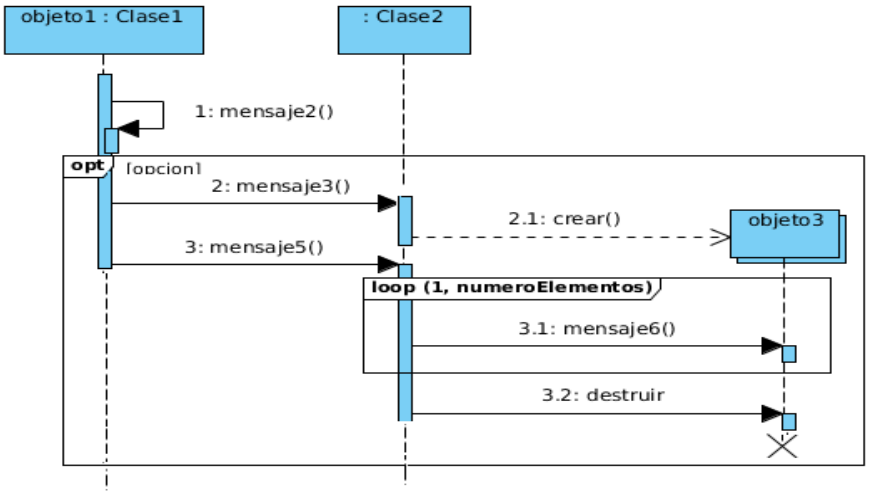
1. Hotel tendrá dos atributos de referencia, uno relativo a las reservas y otro relativo a los clientes.
2. Reserva tendrá los atributos de referencia:
   - private Hotel hotel;
   - private Cliente cliente;

## Respuestas:

1. Falso.
2. Verdadero.

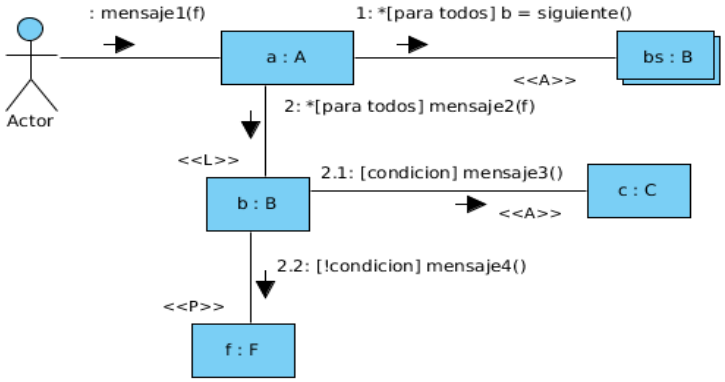
# Ejercicios: Diagramas de interacción

1. Dado el siguiente Diagrama de secuencia, responde verdadero (V) o falso (F) a las siguientes cuestiones:




| | |
|---|---|
| El envío del mensaje n.º 1 (**mensaje2()**) es un envío de mensaje a **self** y además **recursivo** | |
| El objeto **objeto3** es un objeto que vive sólo en esta operación | |
| En la clase **Clase2** tienen que estar definidos los siguientes métodos: **mensaje3()**, **mensaje5()** y **mensaje6()** | |
| El **fragmento combinado** tipo **loop** solo se puede usar para el envío de mensajes a multiobjetos, tal y como está representado en el ejemplo | |
| La representación del multiobjeto está mal, falta especificar la clase a la que pertenecen los objetos que forman ese multiobjeto | |
| La numeración está mal, los envíos de mensaje 3.1 y 3.2 deberían ser 2.2 y 2.3 | |
| El siguiente código Ruby es correcto:<br>`class Clase 2`<br>`  def mensaje5`<br>`  objeto3.each [\|obj\| obj.mensaje6()]`<br>`  objeto3=nil`<br>`  end`<br>`end` | |

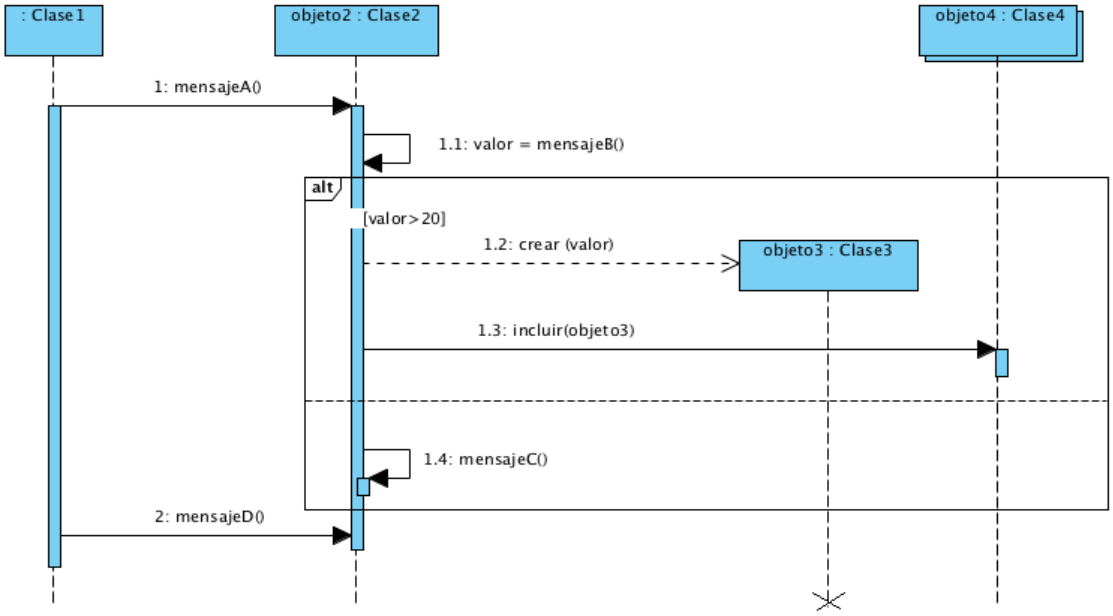
2. Dado el siguiente Diagrama de comunicación, responde verdadero (V) o falso (F) a las siguientes cuestiones



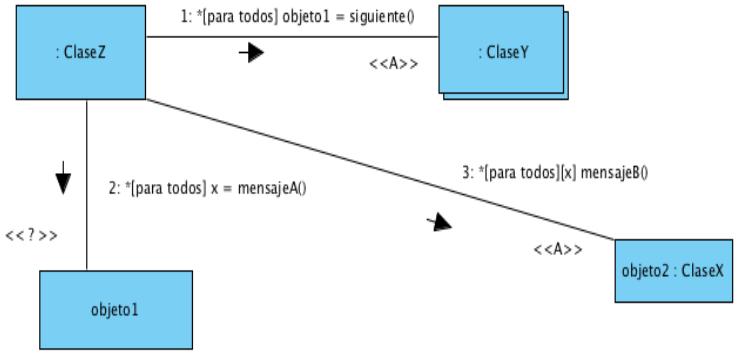

| | |
|---|---|
| El enlace o canal de comunicación estereotipado como **<<P>>** no puede ser de ese tipo ya que el objeto **f:F** no ha entrado como parámetro a la operación | |
| La estructura de control usada en los envíos de mensajes números **2.1** y **2.2** es la estructura **if** (condicion) {...} **else** {...} | |
| En la clase B debe estar implementado el método **siguiente()** para poder responder al envío de mensaje número 1 | |
| A los envíos de mensaje 2.1 y 2.2 les falta ***[para todos]** | |
| El siguiente código Java es correcto:<br>`public class A {`<br>`public void mensaje1(F f){`<br>`for(B b : bs){`<br>`b.mensaje2(f); } }`<br>`}` | |

3. Dados los siguientes diagramas de secuencia y comunicación, implementarlos tanto en Java como en Ruby

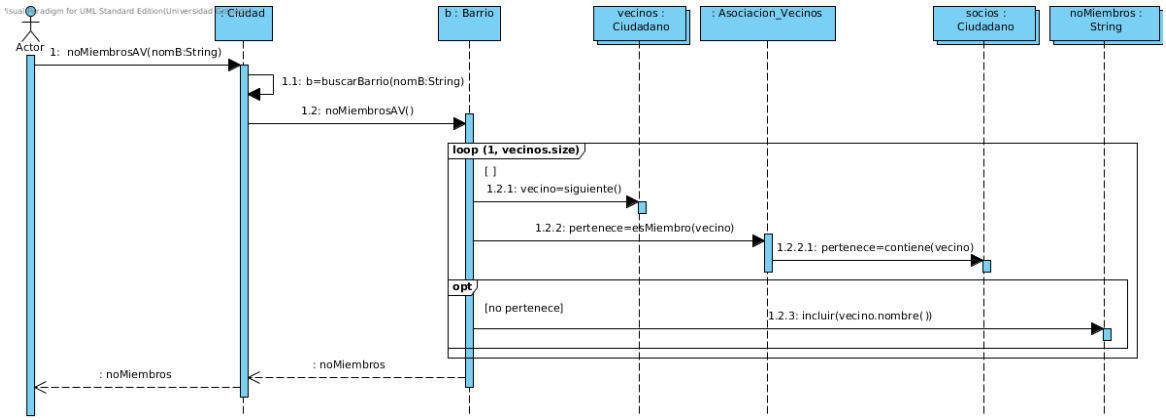
a)



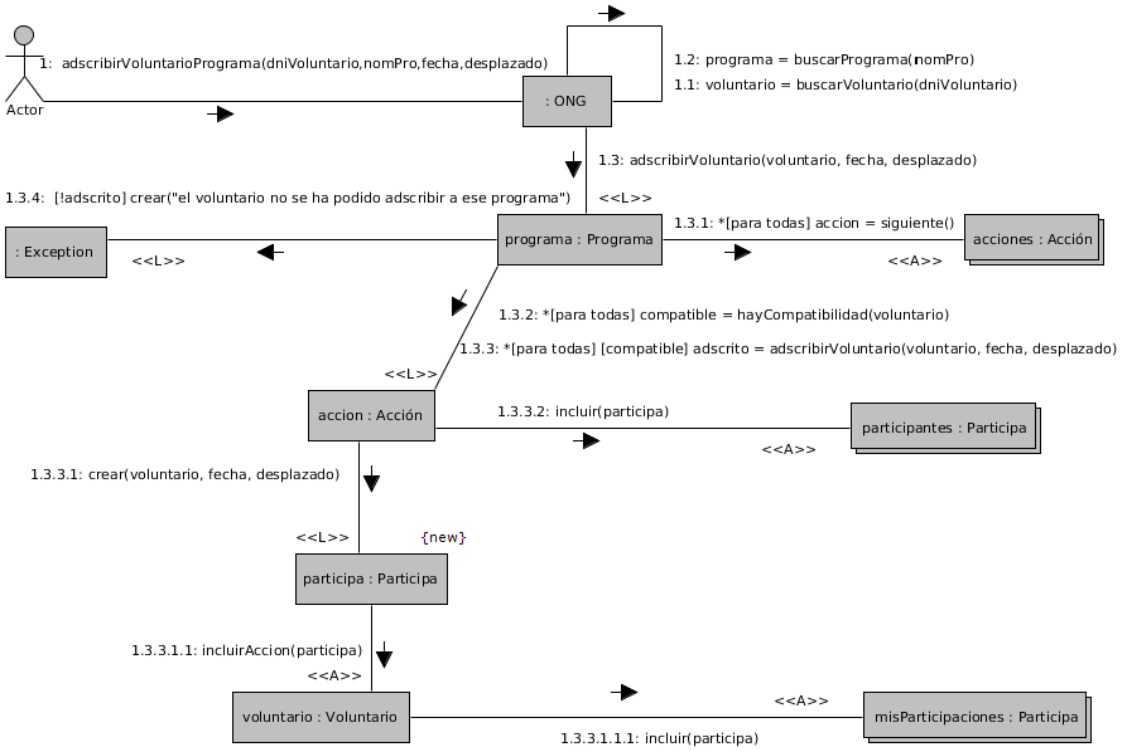
b)


: ClaseZ
1: *[para todos] objeto1 = siguiente()
<<A>>
: ClaseY
2: *[para todos] x = mensajeA()
<<?>>
3: *[para todos][x] mensajeB()
<<A>>
objeto1
objeto2 : ClaseX

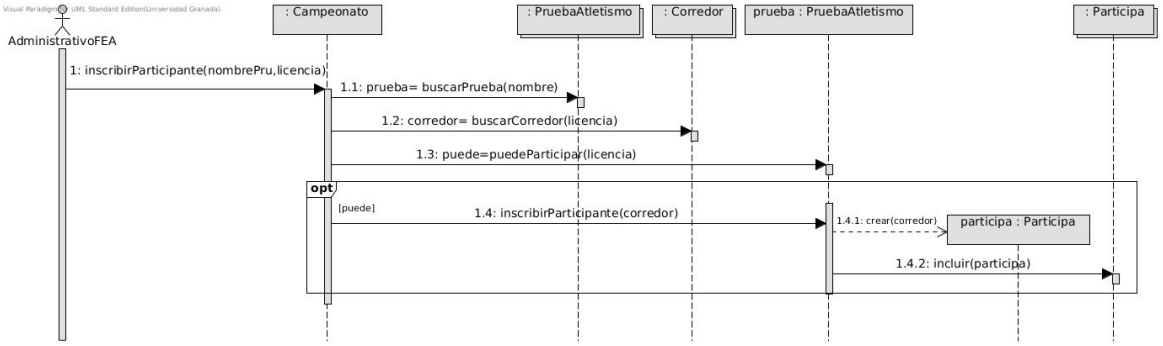
c)


Actor
: Ciudad
b : Barrio
vecinos : Ciudadano
: Asociacion_Vecinos
socios : Ciudadano
noMiembros : String
1: noMiembrosAV(nomB:String)
1.1: b=buscarBarrio(nomB:String)
1.2: noMiembrosAV()
loop (1, vecinos.size)
[ ]
1.2.1: vecino=siguiente()
1.2.2: pertenece=esMiembro(vecino)
1.2.2.1: pertenece=contiene(vecino)
opt
[no pertenece]
1.2.3: incluir(vecino.nombre())
: noMiembros
: noMiembros

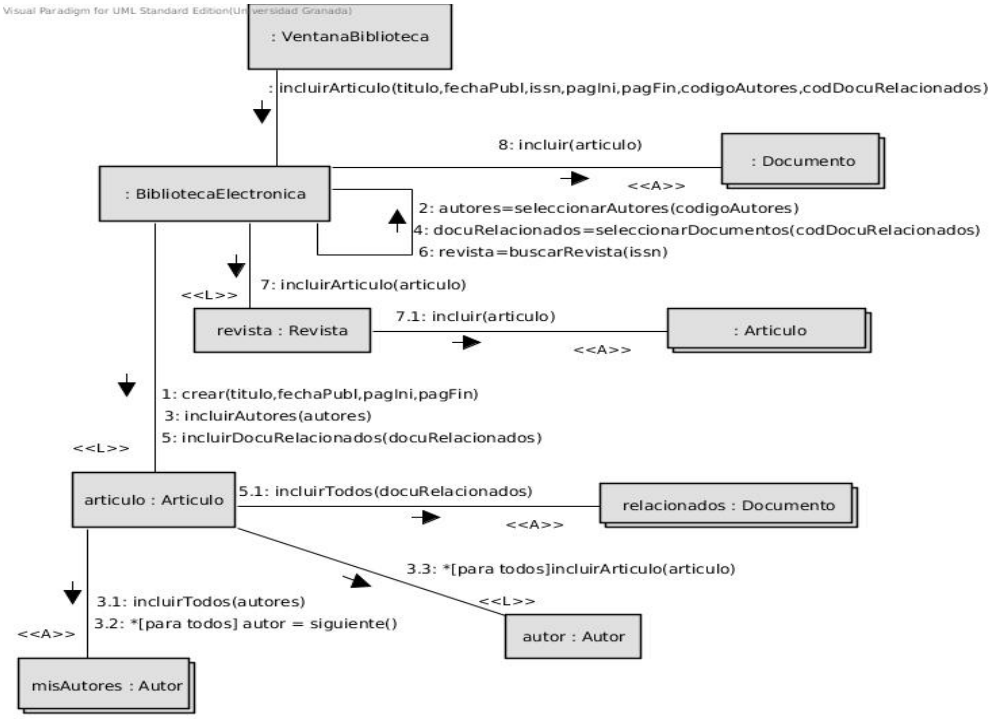
d)


Actor
1: adscribirVoluntarioPrograma(dniVoluntario,nomPro,fecha,desplazado)
: ONG
1.2: programa = buscarPrograma(nomPro)
1.1: voluntario = buscarVoluntario(dniVoluntario)
1.3: adscribirVoluntario(voluntario, fecha, desplazado)
1.3.4: [!adscrito] crear("el voluntario no se ha podido adscribir a ese programa")
<<L>>
: Exception
<<L>>
programa : Programa
1.3.1: *[para todas] accion = siguiente()
<<A>>
acciones : Acción
1.3.2: *[para todas] compatible = hayCompatibilidad(voluntario)
1.3.3: *[para todas] [compatible] adscrito = adscribirVoluntario(voluntario, fecha, desplazado)
<<L>>
accion : Acción
1.3.3.2: incluir(participa)
<<A>>
participantes : Participa
1.3.3.1: crear(voluntario, fecha, desplazado)
<<L>>
{new}
participa : Participa
1.3.3.1.1: incluirAccion(participa)
<<A>>
voluntario : Voluntario
<<A>>
misParticipaciones : Participa
1.3.3.1.1.1: incluir(participa)

e)


AdministrativoFEA
: Campeonato
: PruebaAtletismo
: Corredor
prueba : PruebaAtletismo
: Participa
1: inscribirParticipante(nombrePru,licencia)
1.1: prueba= buscarPrueba(nombre)
1.2: corredor= buscarCorredor(licencia)
1.3: puede=puedeParticipar(licencia)
opt
[puede]
1.4: inscribirParticipante(corredor)
1.4.1: crear(corredor)
participa : Participa
1.4.2: incluir(participa)

f)


: VentanaBiblioteca
: incluirArticulo(titulo,fechaPubl,issn,pagIni,pagFin,codigoAutores,codDocuRelacionados)
8: incluir(articulo)
: Documento
<<A>>
: BibliotecaElectronica
2: autores=seleccionarAutores(codigoAutores)
4: docuRelacionados=seleccionarDocumentos(codDocuRelacionados)
6: revista=buscarRevista(issn)
7: incluirArticulo(articulo)
<<L>>
revista : Revista
7.1: incluir(articulo)
: Articulo
<<A>>
1: crear(titulo,fechaPubl,pagIni,pagFin)
3: incluirAutores(autores)
5: incluirDocuRelacionados(docuRelacionados)
<<L>>
articulo : Articulo
5.1: incluirTodos(docuRelacionados)
relacionados : Documento
<<A>>
3.3: *[para todos]incluirArticulo(articulo)
<<L>>
autor : Autor
3.1: incluirTodos(autores)
3.2: *[para todos] autor = siguiente()
<<A>>
misAutores : Autor

# Ejercicio 1

Dado el siguiente Diagrama de secuencia, responde verdadero (V) o falso (F) a las siguientes cuestiones:

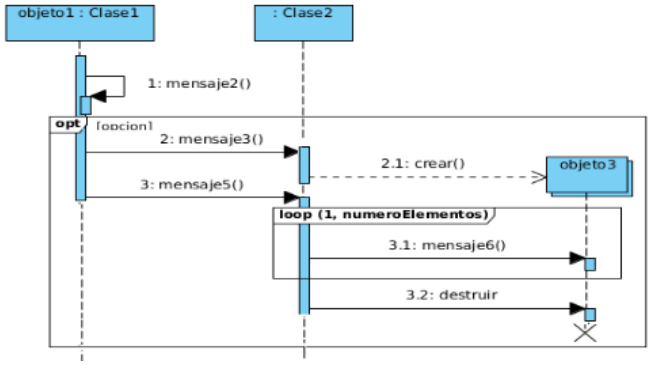



Figure 1: Diagrama de secuencia

- El envío del mensaje n.º 1 (`mensaje2()`) es un envío de mensaje a `self` y además recursivo.
- El objeto `objeto3` es un objeto que vive sólo en esta operación.
- En la clase `Clase2` tienen que estar definidos los siguientes métodos: `mensaje3()`, `mensaje5()` y `mensaje6()`.
- El fragmento combinado tipo `loop` solo se puede usar para el envío de mensajes a multiobjetos, tal y como está representado en el ejemplo.
- La representación del multiobjeto está mal, falta especificar la clase a la que pertenecen los objetos que forman ese multiobjeto.
- La numeración está mal, los envíos de mensaje 3.1 y 3.2 deberían ser 2.2 y 2.3.
- El siguiente código Ruby es correcto:

```ruby
class Clase2
  def mensaje5
    objeto3.each {|obj| obj.mensaje6()}
    objeto3=nil
  end
end
```

### Respuestas

- **Verdadero (V):** El envío del mensaje n.º 1 (`mensaje2()`) es un envío de mensaje a `self` y además recursivo.
- **Falso (F):** El objeto `objeto3` no vive sólo en esta operación, ya que aparece en otros puntos del diagrama.
- **Verdadero (V):** En la clase `Clase2` deben estar definidos los métodos `mensaje3()`, `mensaje5()` y `mensaje6()`.
- **Falso (F):** El fragmento combinado tipo `loop` puede usarse para el envío de mensajes a cualquier tipo de objeto, no solo a multiobjetos.
- **Verdadero (V):** La representación del multiobjeto está mal, falta especificar la clase a la que pertenecen los objetos que forman ese multiobjeto.
- **Verdadero (V):** La numeración está mal, los envíos de mensaje 3.1 y 3.2 deberían ser 2.2 y 2.3.
- **Verdadero (V):** El código Ruby proporcionado es correcto.

## Ejercicio 2

Dado el siguiente Diagrama de comunicación, responde verdadero (V) o falso (F) a las siguientes cuestiones:

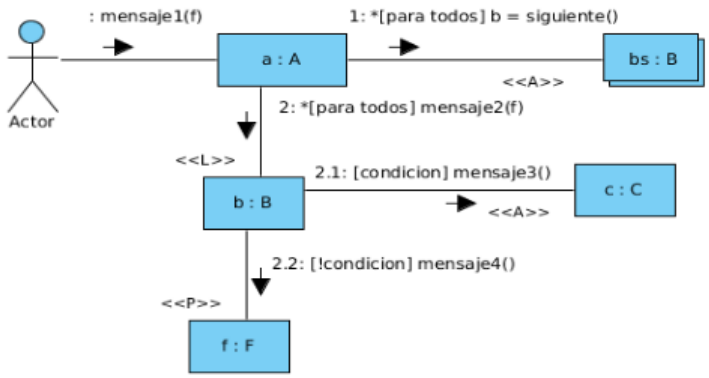



Figure 2: Diagrama de comunicación

- El enlace o canal de comunicación estereotipado como `<<P>>` no puede ser de ese tipo ya que el objeto `f:F` no ha entrado como parámetro a la operación.

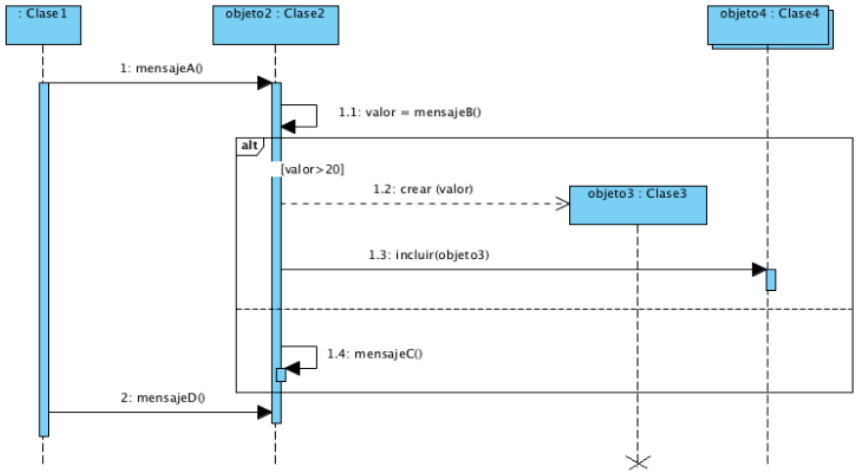
- La estructura de control usada en los envíos de mensajes números 2.1 y 2.2 es la estructura `if (condicion) ... else ...`.
- En la clase `B` debe estar implementado el método `siguiente()` para poder responder al envío de mensaje número 1.
- A los envíos de mensaje 2.1 y 2.2 les falta `*[para todos]`.
- El siguiente código Java es correcto:

```
1 public class A {
2     public void mensaje1(F f){
3         for(B b : bs){
4             b.mensaje2(f);
5         }
6     }
7 }
```

## Respuestas

- **Falso (F):** El enlace o canal de comunicación estereotipado como `<<P>>` no puede ser de ese tipo ya que el objeto `f:F` no ha entrado como parámetro a la operación.
- **Verdadero (V):** La estructura de control usada en los envíos de mensajes números 2.1 y 2.2 es la estructura `if (condicion) ... else ...`.
- **Verdadero (V):** En la clase `B` debe estar implementado el método `siguiente()` para poder responder al envío de mensaje número 1.
- **Falso (F):** A los envíos de mensaje 2.1 y 2.2 les falta `*[para todos]`.
- **Verdadero (V):** El siguiente código Java es correcto:

## Ejercicio 3: Implementar códigos en Java y en Ruby




### Java

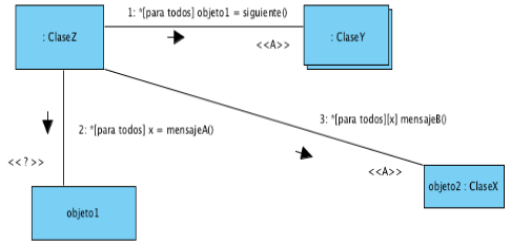
```java
public class Clase1{
    public Tipo mensajeA(){
        int valor = objeto2.mensajeB();
        if(valor>20){
            Clase3 objeto3 = new Clase3(valor);
            objeto4.incluir(objeto3);
        }
        else{
            objeto2.mensajeC();
        }
    }
    public Tipo mensajeD();
}

//Es lo que podemos dedudcir con la informaci n que se nos
    proporciona.
```

### Ruby

```ruby
class Clase1
    def mensajeA
        int valor = objeto.mensajeB();
        if valor>20
            objeto3 = Clase3.new(valor);
            objeto4.incluir(objeto3);
```

```
        else
            objeto2.mensajeC();
        end
    end
    def mensajeD
end
```

b)



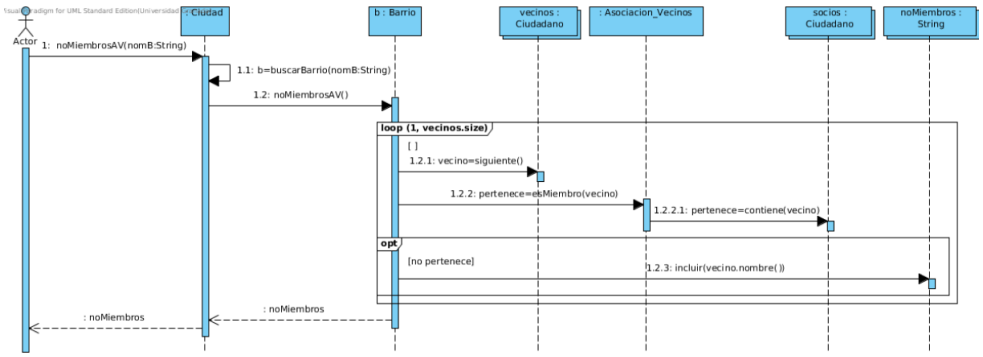

## Java

```
1
2  public class ClaseZ{
3      //no recibe entrada por lo que denotamos al metodo como
       nombreMetodo
4      public Tipo nombreMetodo{
5          for(ClaseY objeto : ElementosDeLaClaseY){
6              tipo x = objeto.mensajeA();
7              if(x){
8                  objeto2.mensajeB();
9              }
10         }
11     }
12 }
```

## Ruby

```
1  # C digo Ruby para el diagrama de secuencia
2  class ClaseZ
3    def nombreMetodo
4      objetoY.each do |objeto1|
5        x = objeto1.mensajeA
6        if x
7          objeto2.mensajeB
8        end
9      end
10   end
11 end
```

c)




## Java

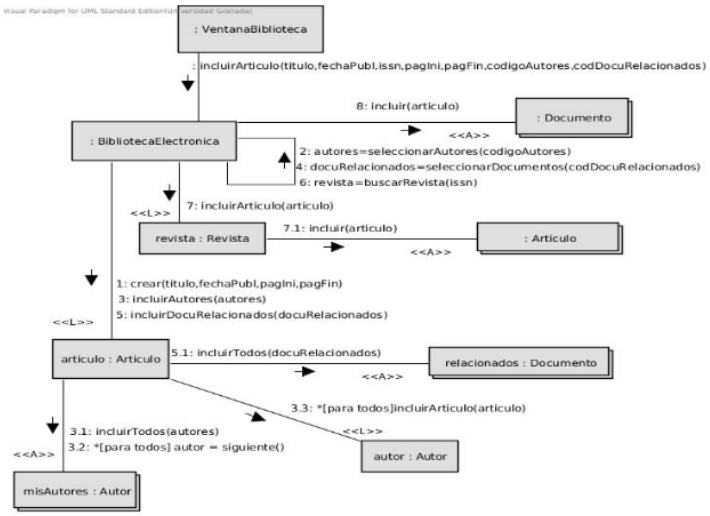
```java
public class Ciudad{
    public String noMiembrosAV(String nomB){
        Barrio b = buscarBarrio(nomB);
        String noMiembros = b.noMiembrosAV();
        return noMiembros;
    }
}

public class Barrio{
    public String noMiembrosAV(){
        for(int i=1;i<vecinos.size();i++){
            Ciudadano vecino = vecinos[i];
            bool pertenece = Asociacion_Vecinos.esMiembro(vecino);
            if(!pertence){
                noMiembros.incluir(vecino.getNombre());
            }
        }
        return noMiembros;
    }
}

public class Asociacion_Vecinos{
    public bool esMiembro(Vecino vecino){
        bool pertence;
        pertenece = socios.contiene(vecino);
        return pertenece;
    }
}
```

## Ruby

```ruby
class Ciudad
    def noMiembrosAV(nomB)
        b = buscarBarrio(nomB)
        noMiembros = b.noMiembrosAV
```

```
        noMiembros
    end
end

class Barrio
    #suponnemos que noMiembros ya esta declarado
    def noMiembrosAV
        for i in 1...vecinos.size
            vecino = vecinos[i]
            pertenece = Asociacion_Vecinos.esMiembro(vecino)
            unless pertenece
                @noMiembros.incluir(vecino.nombre)
            end
        end
        @noMiembros
    end
end

class Asociacion_Vecinos
    def esMiembro(vecino)
        pertenece = socios.contiene(vecino)
        pertenece
    end
end
```

f)




## Java

```
//en la cabecera falta el tipo de cada atributo
//suponemos que los objetos no est n creados debido a que no
    tenemos el diagrama de clases, es mas logico que ya esten
```

```
    declarados
public class BibliotecaElectronica{

    public void incluirArticulo(titulo,fechaPubl,issn,pagIni,pagFin
    ,codigoAutores,codDocuRelacionados){
        Articulo articulo = new Articulo(titulo, fechaPubl,pagIni,
    pagFin);
        Autor autores = seleccionarAutores(codigoAutores);
        articulo.incluirAutores(autores);
        Documento [] docuRelacionados = selecionarDocumentos(
    codDocRelacionados);
        articulo.incluirDocuRelacionados(docuRelacionados);
        Revista revista = buscarRevista(issn);
        revista.incluirArticulo(articulo);
        Documento.incluir(articulo);
    }

}

public class Articulo{
    public void incluirAutores(Autor autores){
        misAutores.incluirTodos(autores);
        for(Autor autor : autores){
            autor.incluirArticulos(articulo);
            //suponemos que articulo esta en la parte privada de la
     clase.
        }
    }

    public void IncluirDocuRelacionados(Documento[]
    docuRelacionados){
        relacionados.incluirTodos(docuRelacionados);
    }
}

public class Revista{
    public void incluirArticulo(Articulo articulo){
        Articulo.incluir(articulo);
    }
}
```

## Ruby

```
class BibliotecaElectronica
  def incluir_articulo(titulo, fecha_publ, issn, pag_ini, pag_fin,
    codigo_autores, cod_docu_relacionados)
    articulo = Articulo.new(titulo, fecha_publ, pag_ini, pag_fin)
    autores = seleccionar_autores(codigo_autores)
    articulo.incluir_autores(autores)
    docu_relacionados = seleccionar_documentos(
    cod_docu_relacionados)
    articulo.incluir_docu_relacionados(docu_relacionados)
    revista = buscar_revista(issn)
    revista.incluir_articulo(articulo)
    Documento.incluir(articulo)
  end
```

```ruby
end

class Articulo
  def incluir_autores(autores)
    mis_autores.incluir_todos(autores)
    autores.each do |autor|
      autor.incluir_articulos(@articulo)
      # suponemos que @articulo est  en la parte privada de la clase.
    end
  end

  def incluir_docu_relacionados(docu_relacionados)
    relacionados.incluir_todos(docu_relacionados)
  end
end

class Revista
  def incluir_articulo(articulo)
    Articulo.incluir(articulo)
  end
end
```