

Fundamentos de POO

**Ingeniería en
Tecnologías de la
Información**

EDER LÓPEZ VILLARREAL

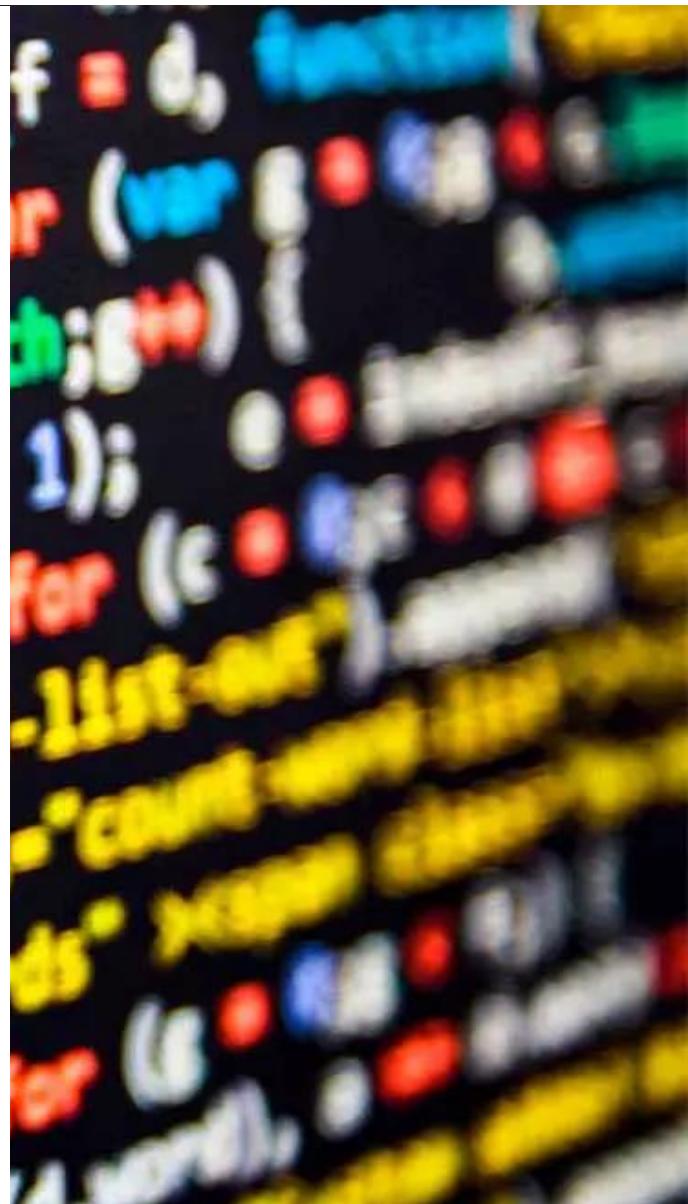
5H T/V

DOCENTE

**MTRO. SAUL OLAF
LOAIZA**

**Portafolio de
evidencia unidad**

II



Fundamentos de POO

Objetivos de cada Practica-Actividades

Práctica 6: Clase Carro con herencia

Objetivo: Realizar el diseño UML de la clase Carro y diseño UML de los objetos de prueba, donde se observe la herencia.

- 1-Realizar el diagrama de UML del clasificador de la clase Carro donde se observe la herencia.
- 2-Realizar el código de la clase Carro de acuerdo al diagrama UML de clase.
- 3-Realizar el diagrama UML de los objetos para verificar la prueba.
- 4-Realizar en la clase principal con 3 objetos con diferentes inicializaciones y que concuerden con el diagrama de objetos.

Práctica 7: Herencia múltiple

Objetivo: Realizar el diseño UML de las clases Persona con herencia sencilla, múltiple y jerárquica con el diseño UML de los objetos de prueba, donde se observe la herencia.

- 1-Realizar el diagrama UML de los objetos para verificar la prueba de la herencia e instancia.
- 2-Realizar en la clase principal con 3 objetos con diferentes inicializaciones y que concuerden con el diagrama de objetos.

Desarrollo diagrama UML de fiesta

Objetivo: Realizar el diseño UML de las clases con sus relaciones entre ellas de la Fiesta de Juan

- 1- Realizar el diagrama de UML del clasificador de cada clase identificada.
- 2- Dibujar las relaciones que existen entre las diferentes clases.
- 3- Subir el archivo realizado en pdf

Desarrollo clase #3 Clase Figura

Objetivo: Realizar el diseño UML de la clase Figura con herencia multinivel y jerarquía

- 1-Realizar el diagrama de UML del clasificador de la clase Figura, Circulo, Rectangulo y Cuadrado de acuerdo a las especificaciones enviadas por el grupo de Telegram
- 2-Realizar el código de las clases donde se observe la herencia multinivel y jerarquía
- 3-Realizar una clase de prueba que testee que todo lo anterior funciona (cada método).

Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

KAREN LETICIA GARCIA VASQUEZ

EDER LÓPEZ VILLARREAL

JAZMIN PORTILLO MICHICOL

ERIC JHONATHAN ANAYA MARQUEZ

5H T/V

DOCENTE

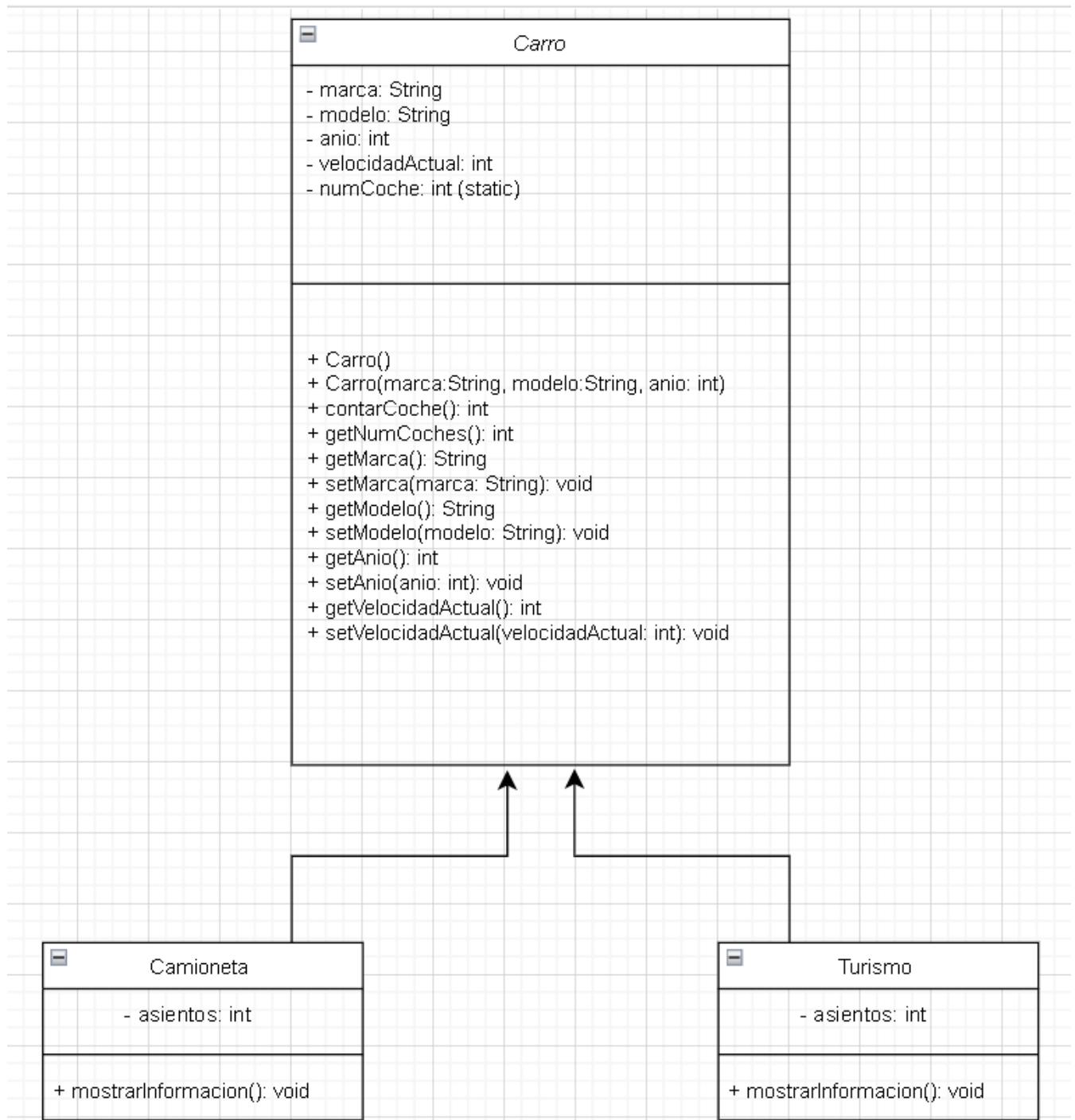
MTRO. SAUL OLAF LOAIZA

Practica 6
Clase Carro con herencia

30/01/2024



Diagrama UML del clasificador de la clase Carro con herencia



Código de clase Carro con herencia

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Practica6 > Version control
- File List:** Main.java, Carro.java (selected), Camioneta.java, Turismo.java
- Code Area:** The Carro.java file content is displayed.

```
1 package POO;
2
3 // La clase Carro representa un vehículo genérico.
4 // 6 usages 2 inheritors
5 public class Carro {
6     private String marca; // Marca del carro
7     private String modelo; // Modelo del carro
8     private int anio; // Año de fabricación del carro
9     private int velocidadActual; // Velocidad actual del carro
10    private static int numCoche; // Contador de cantidad de carros creados
11
12    // Constructores
13
14    // Constructor por defecto de la clase Carro.
15    public Carro() {
16        numCoche++; // Incrementa el contador de coches
17    }
18
19    // Constructor de la clase Carro con parámetros.
20    // 3 usages
```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Practica6 > Version control
- File List:** Main.java, Carro.java (selected), Camioneta.java, Turismo.java
- Code Area:** The Carro.java file content is displayed, showing additional methods and annotations.

```
19    public Carro(String marca, String modelo, int anio) {
20        this.marca = marca; // Asigna la marca del carro
21        this.modelo = modelo; // Asigna el modelo del carro
22        this.anio = anio; // Asigna el año de fabricación del carro
23        numCoche++; // Incrementa el contador de coches
24    }
25
26    // Métodos
27
28    // Método estático que devuelve la cantidad total de coches creados.
29    > public static int contarCoche() { return numCoche; // Devuelve la cantidad total de coches creados }
30
31    // Método que devuelve la cantidad total de coches creados.
32    no usages
33    > public int getNumCoches() { return numCoche; // Devuelve la cantidad total de coches creados }
34
35    /**
36     * Método que devuelve la marca del carro.
37     * @return la marca del carro.
38     */
39    3 usages
40    > public String getMarca() { return marca; // Devuelve la marca del carro }
41
42    // Método que establece la marca del carro.
43    no usages
```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Practica6 Version control
- File Tabs:** Main.java (selected), Carro.java (current), Camioneta.java, Turismo.java
- Code Area:** The Carro.java file contains the following code:

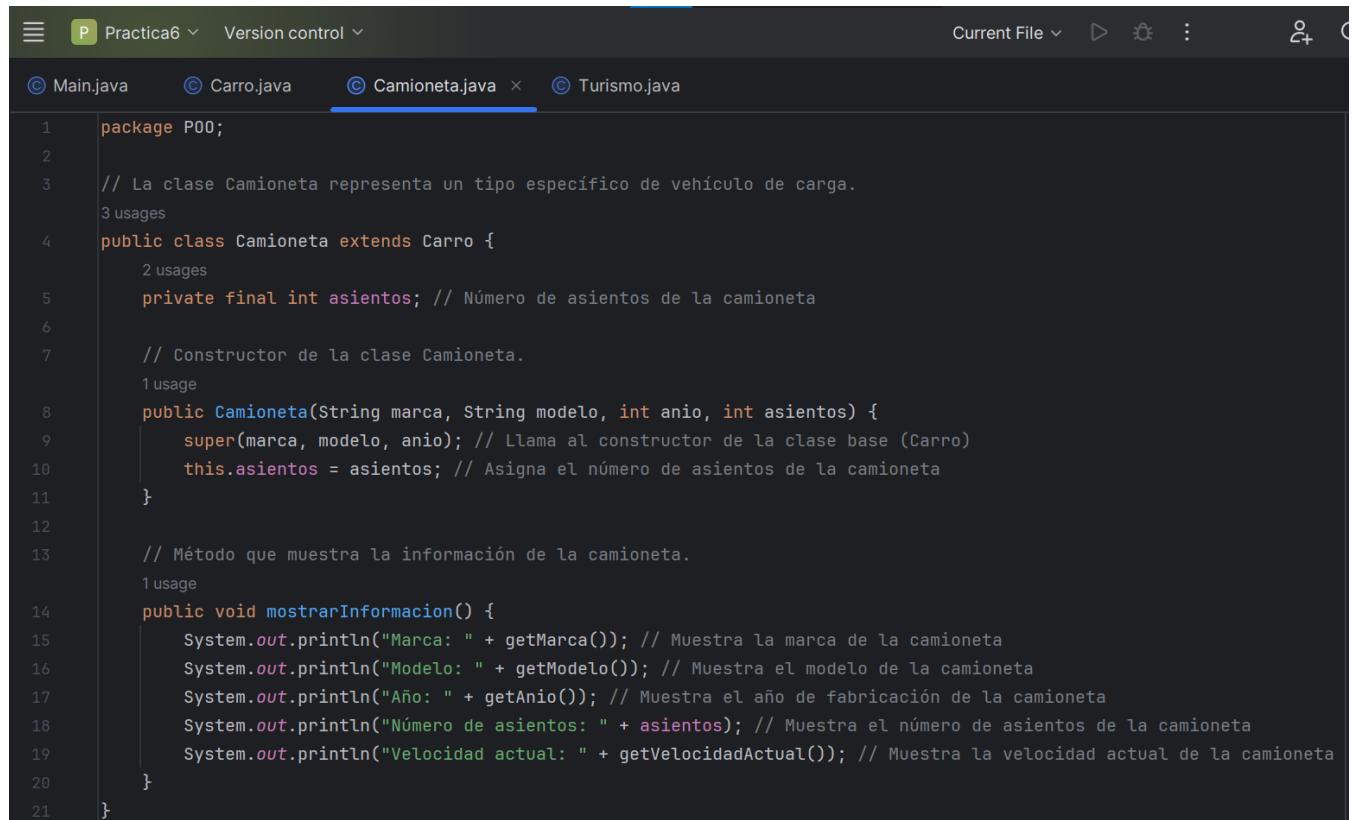
```
47 >     public void setMarca(String marca) { this.marca = marca; // Establece la marca del carro }
50
51     // Método que devuelve el modelo del carro.
52     3 usages
52 >     public String getModelo() { return modelo; // Devuelve el modelo del carro }
55
56     // Método que establece el modelo del carro.
57     no usages
57 >     public void setModelo(String modelo) { this.modelo = modelo; // Establece el modelo del carro }
58
59     //Método que devuelve el año de fabricación del carro.
60     3 usages
62 >     public int getAnio() { return anio; // Devuelve el año de fabricación del carro }
65
66     // Método que establece el año de fabricación del carro.
67     no usages
67 >     public void setAnio(int anio) { this.anio = anio; // Establece el año de fabricación del carro }
70
71     // Método que devuelve la velocidad actual del carro.
72     3 usages
72 >     public int getVelocidadActual() { return velocidadActual; // Devuelve la velocidad actual del carro }
75
76     //Método que establece la velocidad actual del carro.
77
78     3 usages
78 >     public void setVelocidadActual(int velocidadActual) {
```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Practica6 Version control
- File Tabs:** Main.java (selected), Carro.java (current), Camioneta.java, Turismo.java
- Code Area:** The Carro.java file contains the following code:

```
79         this.velocidadActual = velocidadActual; // Establece la velocidad actual del carro
80     }
81 }
```

Código de clase Camioneta hija

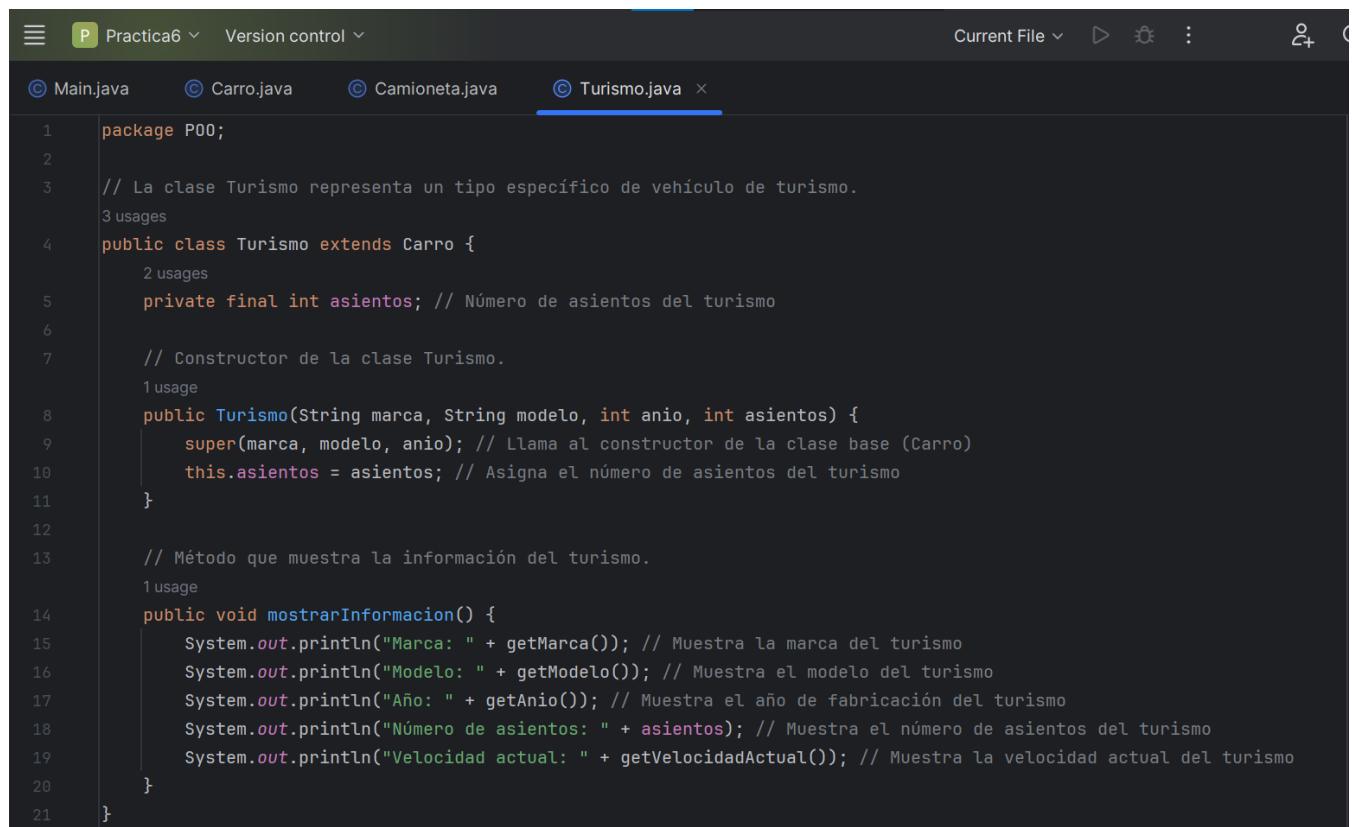


The screenshot shows a Java code editor interface with the following details:

- Project:** Practica6
- File:** Camioneta.java (highlighted)
- Code Content:**

```
1 package POO;
2
3 // La clase Camioneta representa un tipo específico de vehículo de carga.
4 public class Camioneta extends Carro {
5     private final int asientos; // Número de asientos de la camioneta
6
7     // Constructor de la clase Camioneta.
8     public Camioneta(String marca, String modelo, int anio, int asientos) {
9         super(marca, modelo, anio); // Llama al constructor de la clase base (Carro)
10        this.asientos = asientos; // Asigna el número de asientos de la camioneta
11    }
12
13    // Método que muestra la información de la camioneta.
14    public void mostrarInformacion() {
15        System.out.println("Marca: " + getMarca()); // Muestra la marca de la camioneta
16        System.out.println("Modelo: " + getModelo()); // Muestra el modelo de la camioneta
17        System.out.println("Año: " + getAnio()); // Muestra el año de fabricación de la camioneta
18        System.out.println("Número de asientos: " + asientos); // Muestra el número de asientos de la camioneta
19        System.out.println("Velocidad actual: " + getVelocidadActual()); // Muestra la velocidad actual de la camioneta
20    }
21 }
```
- Toolbar:** Includes icons for file operations, search, and help.

Código de clase Turismo hija

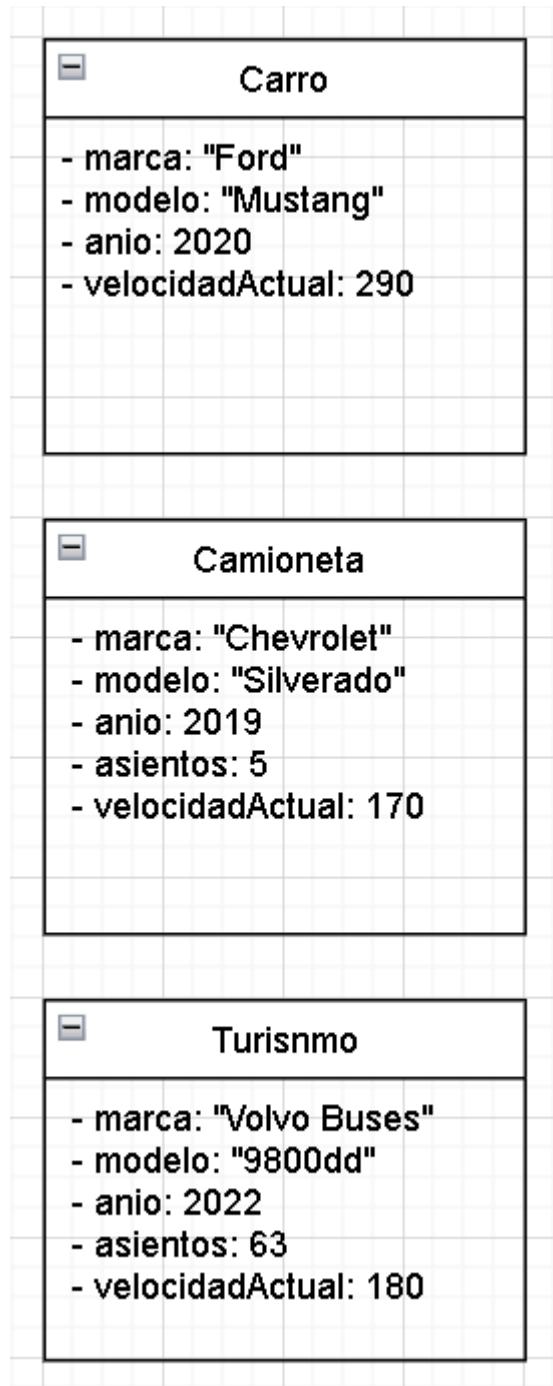


The screenshot shows a Java code editor interface with the following details:

- Project:** Practica6
- File:** Turismo.java (highlighted)
- Code Content:**

```
1 package POO;
2
3 // La clase Turismo representa un tipo específico de vehículo de turismo.
4 public class Turismo extends Carro {
5     private final int asientos; // Número de asientos del turismo
6
7     // Constructor de la clase Turismo.
8     public Turismo(String marca, String modelo, int anio, int asientos) {
9         super(marca, modelo, anio); // Llama al constructor de la clase base (Carro)
10        this.asientos = asientos; // Asigna el número de asientos del turismo
11    }
12
13    // Método que muestra la información del turismo.
14    public void mostrarInformacion() {
15        System.out.println("Marca: " + getMarca()); // Muestra la marca del turismo
16        System.out.println("Modelo: " + getModelo()); // Muestra el modelo del turismo
17        System.out.println("Año: " + getAnio()); // Muestra el año de fabricación del turismo
18        System.out.println("Número de asientos: " + asientos); // Muestra el número de asientos del turismo
19        System.out.println("Velocidad actual: " + getVelocidadActual()); // Muestra la velocidad actual del turismo
20    }
21 }
```
- Toolbars and Status Bar:** Current File, etc.

Diagrama UML de los objetos para verificar la prueba



Clase principal Main

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows "Practica6" and "Version control".
- File List:** Shows "Main.java", "Carro.java", "Camioneta.java", and "Turismo.java".
- Code Area:** Displays the following Java code:

```
1 import POO.Carro;
2 import POO.Camioneta;
3 import POO.Turismo;
4
5 public class Main {
6     public static void main(String[] args) {
7         // Crear un objeto de tipo Carro
8         Carro carro = new Carro( marca: "Ford", modelo: "Mustang", anio: 2020);
9         carro.setVelocidadActual(290); // Definir la velocidad actual del carro
10
11        // Crear un objeto de tipo Camioneta
12        Camioneta camioneta = new Camioneta( marca: "Chevrolet", modelo: "Silverado", anio: 2019, asientos: 5);
13        camioneta.setVelocidadActual(170); // Definir la velocidad actual de la camioneta
14
15        // Crear un objeto de tipo Turismo (Autobús)
16        Turismo autobus = new Turismo( marca: "Volvo Buses", modelo: "9800dd", anio: 2022, asientos: 63);
17        autobus.setVelocidadActual(180); // Definir la velocidad actual del autobús
18
19        // Mostrar información de los vehículos
20        System.out.println("\nInformación del carro:");
21        System.out.println("Marca: " + carro.getMarca());
22        System.out.println("Modelo: " + carro.getModelo());
23        System.out.println("Año: " + carro.getAnio());
24        System.out.println("Velocidad actual: " + carro.getVelocidadActual());
25
26        System.out.println("\nInformación de la camioneta:");
27
28        camioneta.mostrarInformacion();
29
30        System.out.println("\nInformación del autobús:");
31        autobus.mostrarInformacion();
32
33        // Contar la cantidad de coches creados
34        System.out.println("\nNúmero total de coches: " + Carro.contarCoche());
35    }
}
```

The screenshot shows a continuation of the Java code editor interface with the following details:

- Title Bar:** Shows "Practica6" and "Version control".
- File List:** Shows "Main.java", "Carro.java", "Camioneta.java", and "Turismo.java".
- Code Area:** Displays the following Java code:

```
27     camioneta.mostrarInformacion();
28
29     System.out.println("\nInformación del autobús:");
30     autobus.mostrarInformacion();
31
32     // Contar la cantidad de coches creados
33     System.out.println("\nNúmero total de coches: " + Carro.contarCoche());
34 }
35 }
```

Prueba

```
Practica6 Version control

Main.java  Carro.java  Camioneta.java  Turismo.java

Run Main

C D | Camera | Open | ...

↑ Información del carro:
↓ Marca: Ford
↔ Modelo: Mustang
☰ Año: 2020
🖨️ Velocidad actual: 290
Trash

↑ Información de la camioneta:
↓ Marca: Chevrolet
↔ Modelo: Silverado
☰ Año: 2019
Nº de asientos: 5
🖨️ Velocidad actual: 170
Trash

↑ Información del autobús:
↓ Marca: Volvo Buses
↔ Modelo: 9800dd
☰ Año: 2022
Nº de asientos: 63
🖨️ Velocidad actual: 180
Trash

Nº total de coches: 3
```

Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

EDER LÓPEZ VILLARREAL
JAZMIN PORTILLO MICHICOL
KAREN LETICIA GARCIA VASQUEZ
ERIC JHONTHON ANAYA MARQUEZ

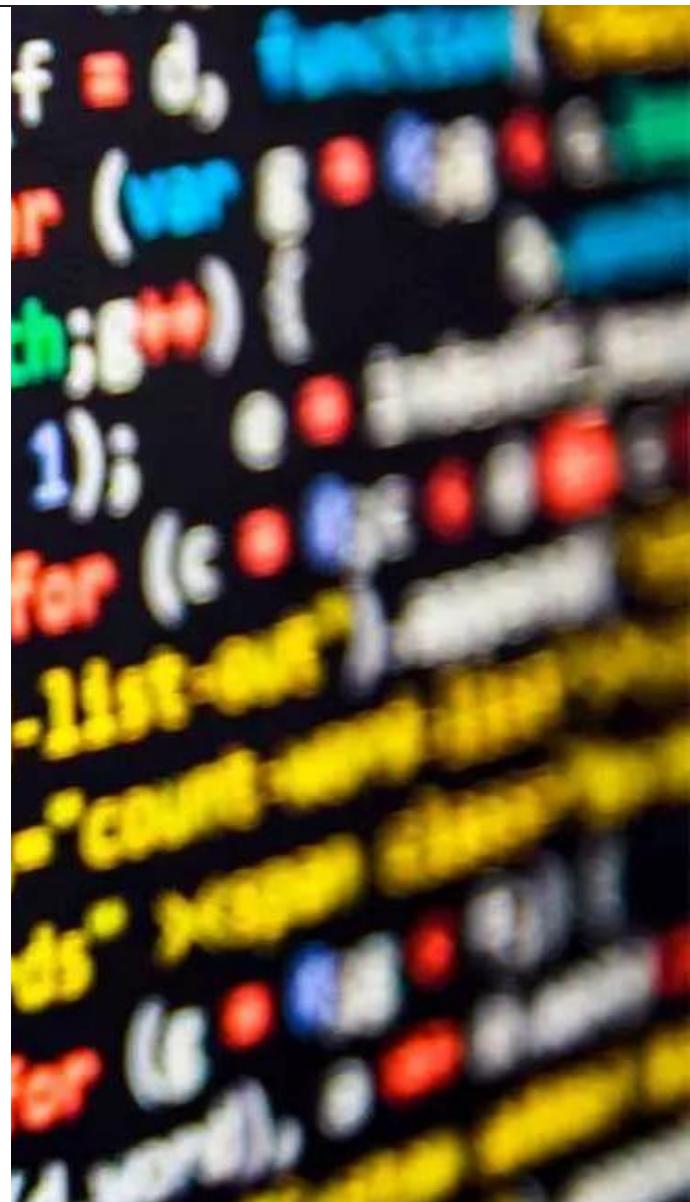
5H T/V

DOCENTE

MTRO. SAUL OLAF LOAIZA

Practica 6
Tipos de relaciones

30/01/2024



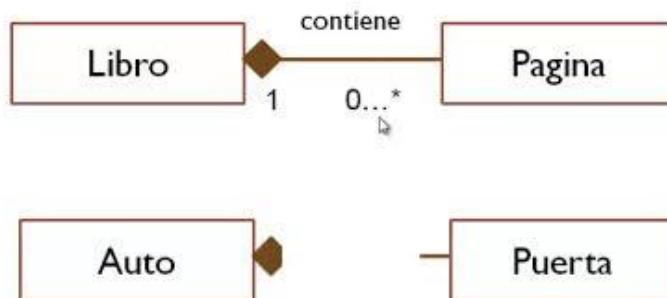
Tipos de relaciones

Asociación

Representa la relación entre dos clases sin que una dependa de la otra. Puede ser de uno a uno, uno a muchos o muchos a muchos.

Por ejemplo: una clase Biblioteca podría tener una asociación uno a muchos con una clase Libro, indicando que cada instancia de Libro pertenece a una instancia de Biblioteca.

Ejemplo

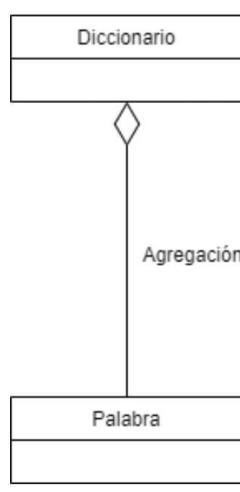


Miriam Lopez Surco

Agregación

Indica una relación "todo-parte" entre una clase principal (todo) y una clase secundaria (parte). La parte puede existir de manera independiente al todo.

Ejemplo: Tenemos una relación de agregación de las palabras (Partes) con un diccionario (Todo). El objetivo del sistema es determinar el significado de una palabra. No existe, conceptualmente, una estructura subyacente debajo de la relación.



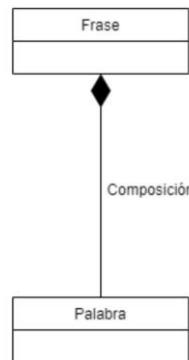
Composición

Similar a la agregación, pero con una relación más fuerte. La parte (objeto secundario) no puede existir sin el todo (objeto principal). Si el todo se destruye, las partes también se destruyen.

Ejemplo: Tenemos una relación de composición de las palabras (Partes) con una frase (Todo). El objetivo del sistema es modelizar frases. En este caso sí que existe, conceptualmente, una estructura subyacente dentro de la relación. Las palabras se deben estructurar en un determinado orden para que la frase sea correcta.

Si estamos modelizando el motor de un coche usamos una relación de composición con sus partes. Si estamos modelizando una tienda de piezas de coche estamos hablando de agregación.

Este concepto es muy importante porque el paso a composición indica que existe una estructura subyacente entre las partes.

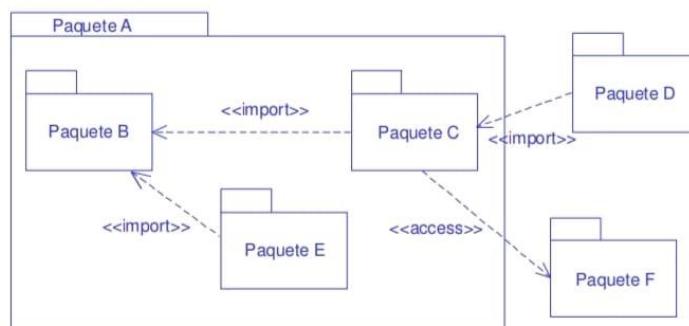


Dependencia

Indica que una clase depende de otra en algún aspecto, pero no implica una relación estructural permanente. Cambios en la clase independiente pueden afectar a la dependiente.

Ejemplo: – Una relación de dependencia entre paquetes significa que al menos un elemento del paquete cliente utiliza los servicios ofrecidos por al menos un elemento del paquete proveedor.

– Cuando se importa un paquete, sólo son accesibles sus elementos públicos.



Herencia

Permite que una clase herede atributos y métodos de otra clase. La clase que hereda se llama subclase o clase derivada, y la clase de la cual hereda se llama superclase o clase base.

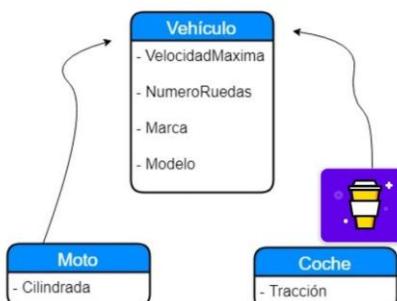
Ejemplo: el siguiente código

```
class Vehiculo
{
    public decimal VelocidadMaxima;
    public int NumeroRuedas { get; set; }
    public string Marca { get; set; }
    public string Modelo { get; set; }
}

class Moto : Vehiculo
{
    public int Cilindrada { get; set; }
}

class Coche : Vehiculo
{
    public string Traccion { get; set; }
}
```

Traducido a una imagen será la siguiente:



Como podemos observar tanto coche como Moto apuntan a Vehiculo, eso quiere decir que pueden usar tanto sus propiedades como sus métodos. Por lo que el siguiente código Moto.VelocidadMaxima o Moto.Cilindrada es igual de válido que Coche.VelocidadMaxima o Coche.Traccion.

Para indicar que una clase deriva de otra debemos indicarlo con los dos puntos : como vemos en el ejemplo.

Cuando heredamos de una clase padre únicamente podemos hacerlo de una sola clase. No podemos heredar de dos clases, por ejemplo class Triciclo : Mixto, Vehiculo no es válido. Pero si en este ejemplo, la clase Mixto hereda de Vehiculo, si podemos hacer lo siguiente: Como observamos Mixto hereda de Vehiculo, y Triciclo hereda de Mixto por lo que desde triciclo tendremos acceso a las propiedades de la clase Vehiculo como vemos en el ejemplo.

```
class Vehiculo
{
    public int NumeroRuedas { get; set; }
}

class Mixto : Vehiculo
{
    //Código
}

class Triciclo : Mixto
{
    public void AsignarRuedas()
    {
        NumeroRuedas = 3;
    }
}
```


Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

KAREN LETICIA GARCIA VASQUEZ

EDER LÓPEZ VILLARREAL

JAZMIN PORTILLO MICHICOL

ERIC JHONATHAN ANAYA MARQUEZ

5H T/V

DOCENTE

MTRO. SAUL OLAF LOAIZA

Practica 7
Herencia Multiple

02/02/2024

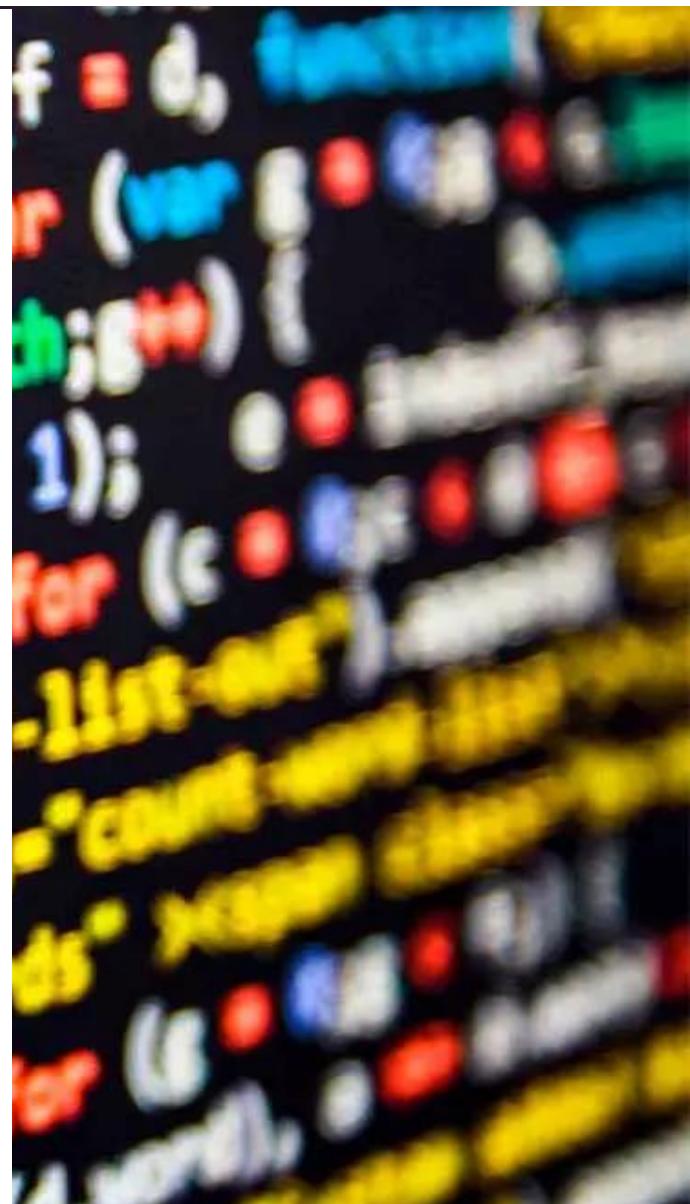
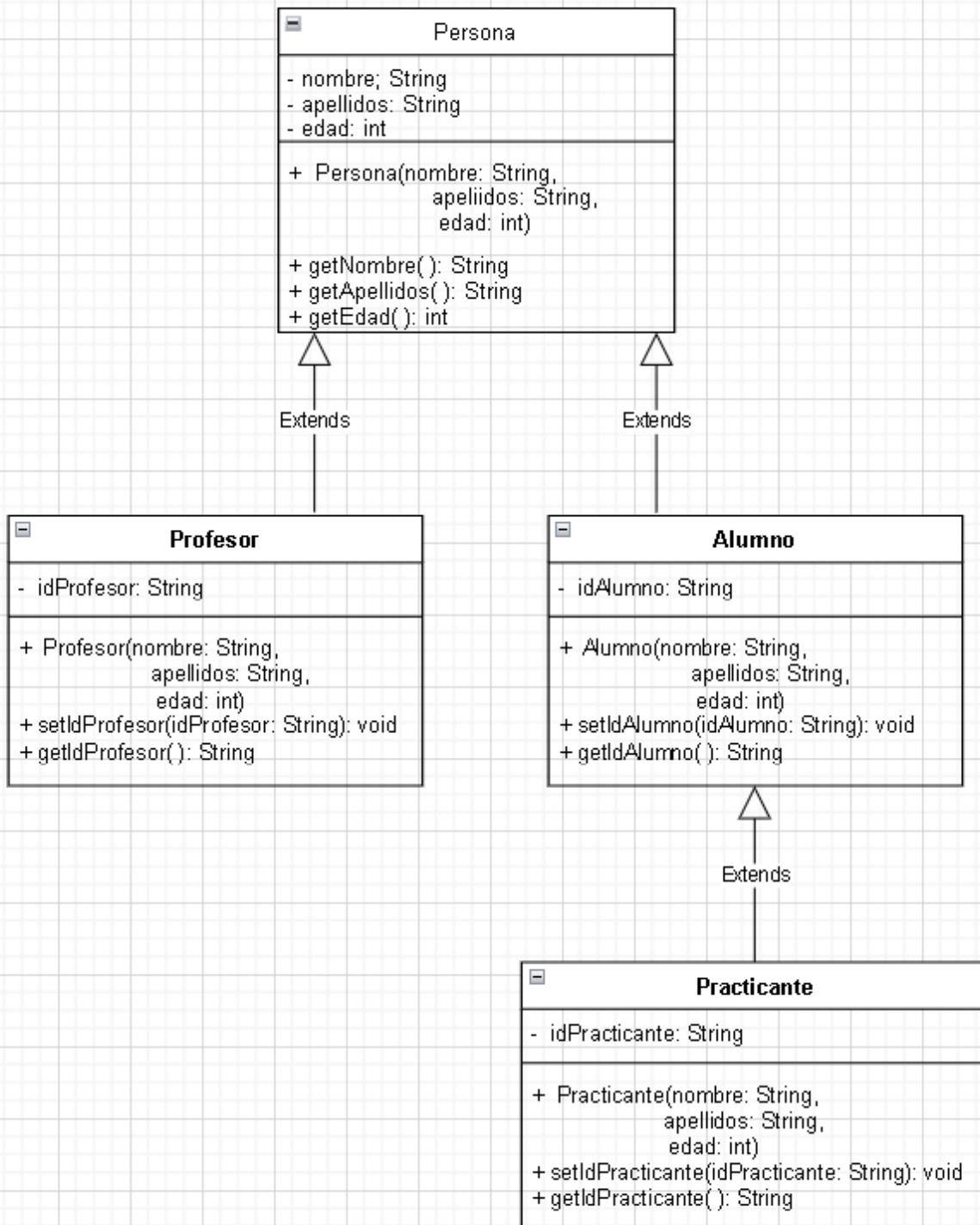


Diagrama UML del clasificador de la clase Persona con herencia sencilla, multiple y jerárquica



Código de clase Persona

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows "Practica7" and "Version control".
- File List:** Shows "Main.java", "Person.java" (selected), "Profesor.java", "Alumno.java", and "Practicante.java".
- Code Area:** Displays the Person class definition. The code includes:
 - package P00;
 - // Clase Persona
 - public class Persona { // La clase Persona representa a una persona con un nombre, apellidos y edad.
 - protected String nombre; // Nombre de la persona.
 - protected String apellidos; // Apellidos de la persona.
 - protected int edad; // Edad de la persona.
 - // Constructor para crear un objeto Persona con nombre, apellidos y edad especificados.
 - // @param nombre El nombre de la persona.
 - // @param apellidos Los apellidos de la persona.
 - // @param edad La edad de la persona.
 - public Persona(String nombre, String apellidos, int edad) {
 this.nombre = nombre;
 this.apellidos = apellidos;
 this.edad = edad;
}
 - // Método para obtener el nombre de la persona.
 - // @return El nombre de la persona.

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows "Practica7" and "Version control".
- File List:** Shows "Main.java", "Person.java" (selected), "Profesor.java", "Alumno.java", and "Practicante.java".
- Code Area:** Displays the implementation of the Person class methods. The code includes:
 - public String getNombre() {
 return nombre;
}
 - // Método para obtener los apellidos de la persona.
// @return Los apellidos de la persona.
 - public String getApellidos() {
 return apellidos;
}
 - // Método para obtener la edad de la persona.
// @return La edad de la persona.
 - public int getEdad() {
 return edad;
}

Código de clase Profesor

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows "Practica7" and "Version control".
- File List:** Includes Main.java, Persona.java, Profesor.java (highlighted in blue), Alumno.java, and Practicante.java.
- Code Area:** Displays the content of the Profesor.java file. The code defines a class Profesor that extends Persona. It includes a constructor that takes name, surnames, and age, and calls the super constructor. It also includes methods setIdProfesor and getIdProfesor.

```
1 package POO;
2
3 // Clase Profesor, hereda de Persona
4 public class Profesor extends Persona { // La clase Profesor extiende la clase Persona y representa a un profesor.
5
6     private String idProfesor; // Identificador único del profesor.
7
8     // Constructor para crear un objeto Profesor con nombre, apellidos y edad especificados.
9     // @param nombre El nombre del profesor.
10    // @param apellidos Los apellidos del profesor.
11    // @param edad La edad del profesor.
12    public Profesor(String nombre, String apellidos, int edad) {
13        super(nombre, apellidos, edad); // Llama al constructor de la clase padre Persona.
14    }
15
16    // Método para establecer el identificador del profesor.
17    // @param idProfesor El identificador del profesor.
18    public void setIdProfesor(String idProfesor) {
19        this.idProfesor = idProfesor;
20    }
21
22    // Método para obtener el identificador del profesor.
```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows "Practica7" and "Version control".
- File List:** Includes Main.java, Persona.java, Profesor.java (highlighted in blue), Alumno.java, and Practicante.java.
- Code Area:** Displays the content of the Profesor.java file. It includes a new method getIdProfesor that returns the idProfesor.

```
23     // @return El identificador del profesor.
24     public String getIdProfesor() {
25         return idProfesor;
26     }
27 }
```

Código de clase Alumno

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows "Practica7" and "Version control".
- File Tabs:** Main.java, Persona.java, Profesor.java, Alumno.java (selected), Practicante.java.
- Code Content:** The Alumno.java file contains the following code:

```
1 package POO;
2
3 // Clase Alumno, hereda de Persona
4 usages 1 inheritor
5
6 public class Alumno extends Persona {    // La clase Alumno extiende la clase Persona y representa a un alumno.
7
8     private String idAlumno;    // Identificador Único del alumno.
9
10    // Constructor para crear un objeto Alumno con nombre, apellidos y edad especificados.
11    // @param nombre El nombre del alumno.
12    // @param apellidos Los apellidos del alumno.
13    // @param edad La edad del alumno.
14
15    // Método para establecer el identificador del alumno.
16    // @param idAlumno El identificador del alumno.
17
18    public void setIdAlumno(String idAlumno) {
19        this.idAlumno = idAlumno;
20    }
21
22    // Método para obtener el identificador del alumno.
```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows "Practica7" and "Version control".
- File Tabs:** Main.java, Persona.java, Profesor.java, Alumno.java (selected), Practicante.java.
- Code Content:** The Alumno.java file contains the following code:

```
23     // @return El identificador del alumno.
24     public String getIdAlumno() {
25         return idAlumno;
26     }
27 }
```

Código de clase Practicante

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows "Practica7" and "Version control".
- File Tabs:** Main.java, Persona.java, Profesor.java, Alumno.java, and Practicante.java (highlighted).
- Code Area:** Displays the content of the Practicante.java file.

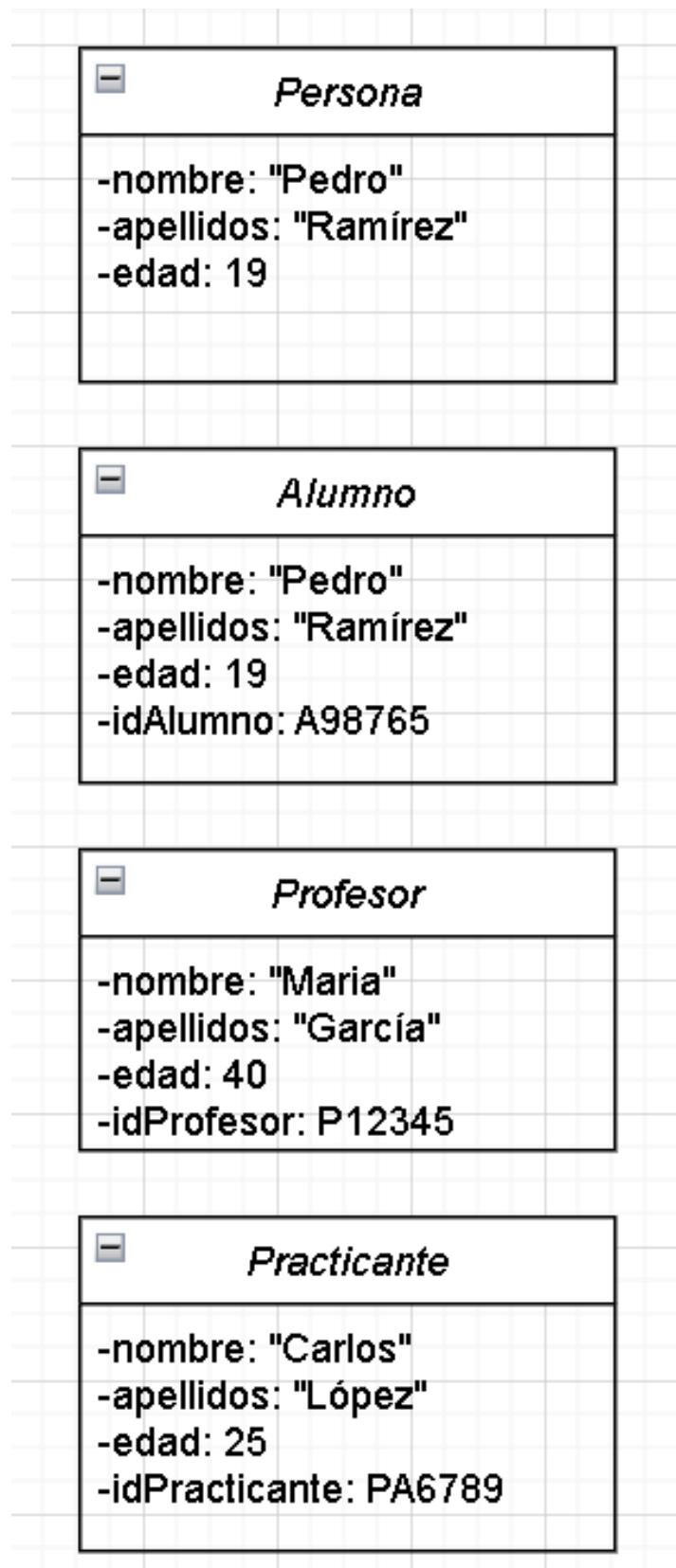
```
1 package POO;
2
3 // Clase Practicante, hereda de Alumno
4 public class Practicante extends Alumno {    // La clase Practicante extiende la clase Alumno y representa a un practicante
5
6     private String idPracticante;    // Identificador único del practicante.
7
8     // Constructor para crear un objeto Practicante con nombre, apellidos y edad especificados.
9     // @param nombre El nombre del practicante.
10    // @param apellidos Los apellidos del practicante.
11    // @param edad La edad del practicante.
12    public Practicante(String nombre, String apellidos, int edad) {
13        super(nombre, apellidos, edad);    // Llama al constructor de la clase padre Alumno.
14    }
15
16    // Método para establecer el identificador del practicante.
17    // @param idPracticante El identificador del practicante.
18    public void setIdPracticante(String idPracticante) {
19        this.idPracticante = idPracticante;
20    }
21
22    // Método para obtener el identificador del practicante.
```

The screenshot shows a Java code editor interface with the following details:

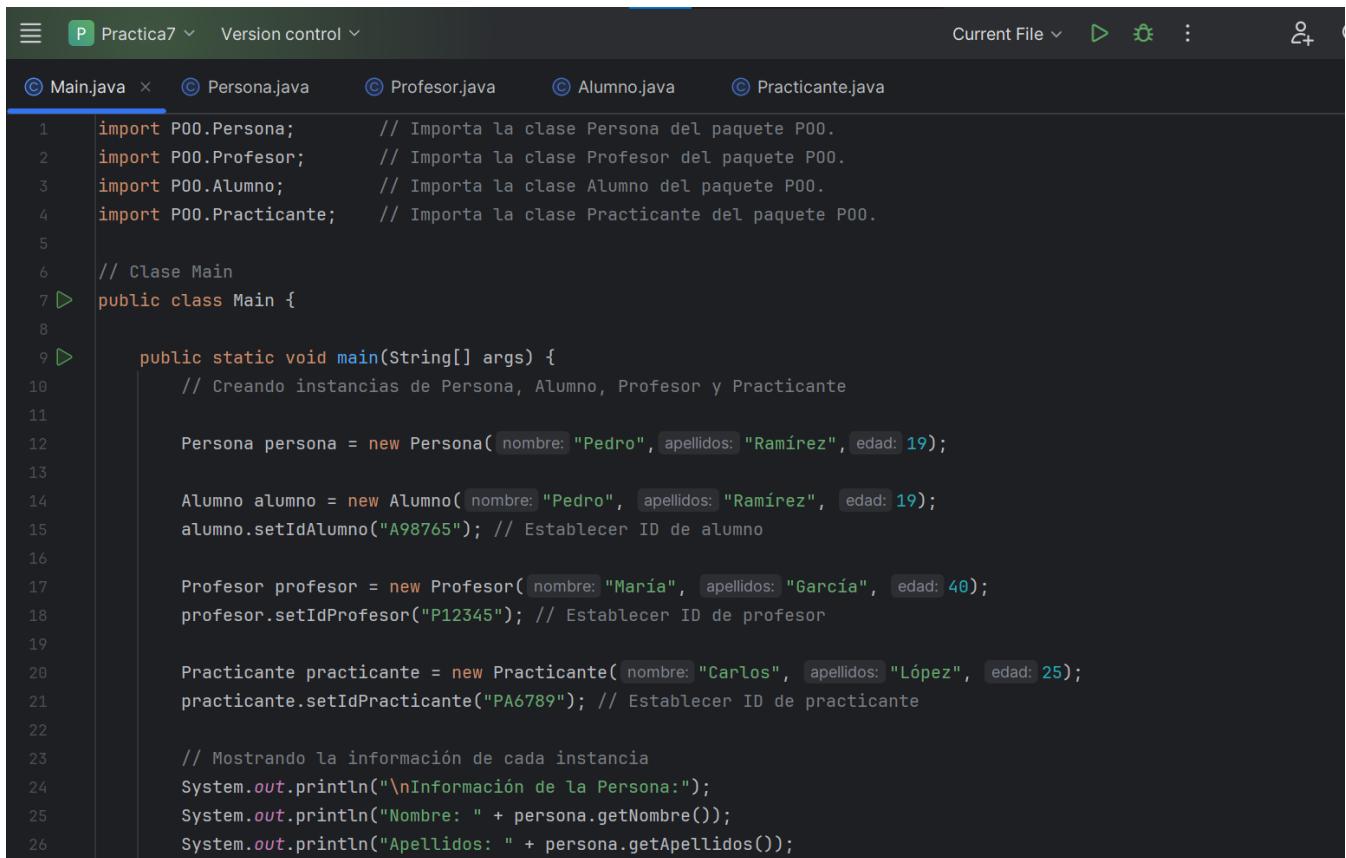
- Title Bar:** Shows "Practica7" and "Version control".
- File Tabs:** Main.java, Persona.java, Profesor.java, Alumno.java, and Practicante.java (highlighted).
- Code Area:** Displays the content of the Practicante.java file, specifically focusing on the getIdPracticante() method.

```
23     // @return El identificador del practicante.
24     public String getIdPracticante() {
25         return idPracticante;
26     }
27 }
```

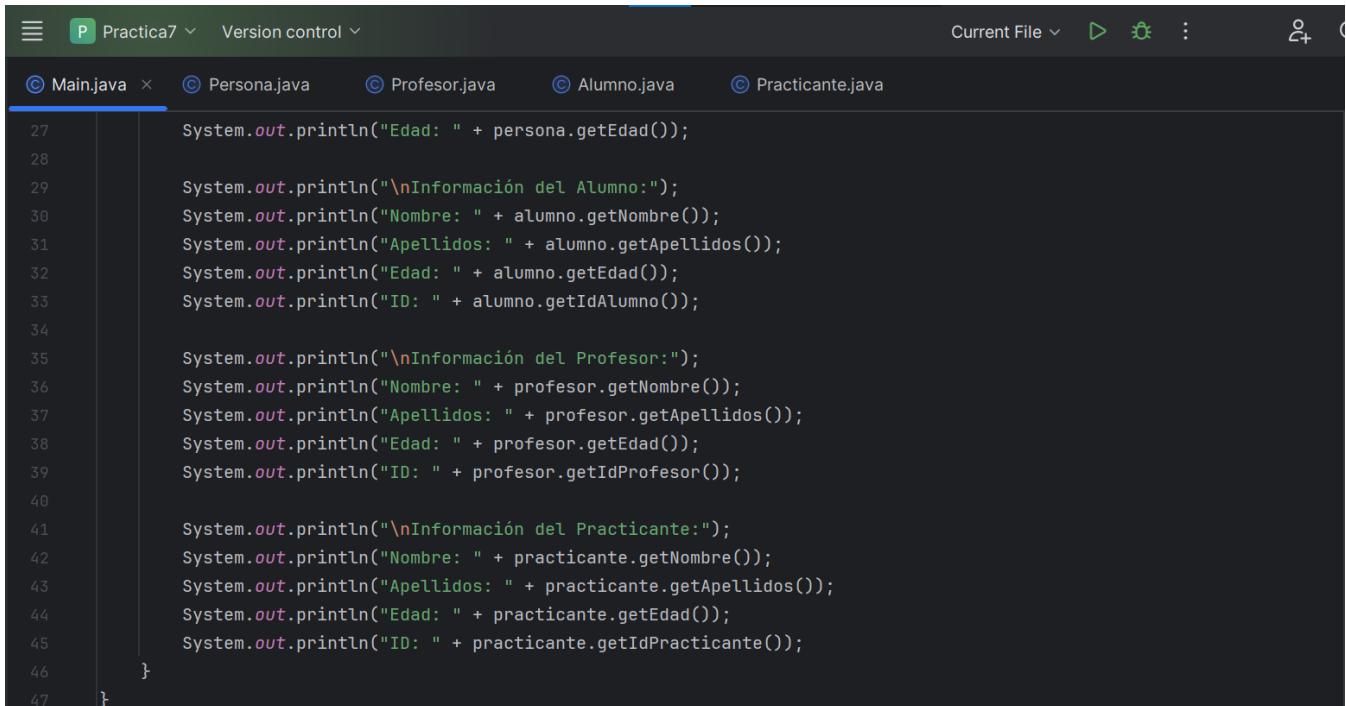
Diagrama UML de los objetos para verificar la prueba



Clase principal Main



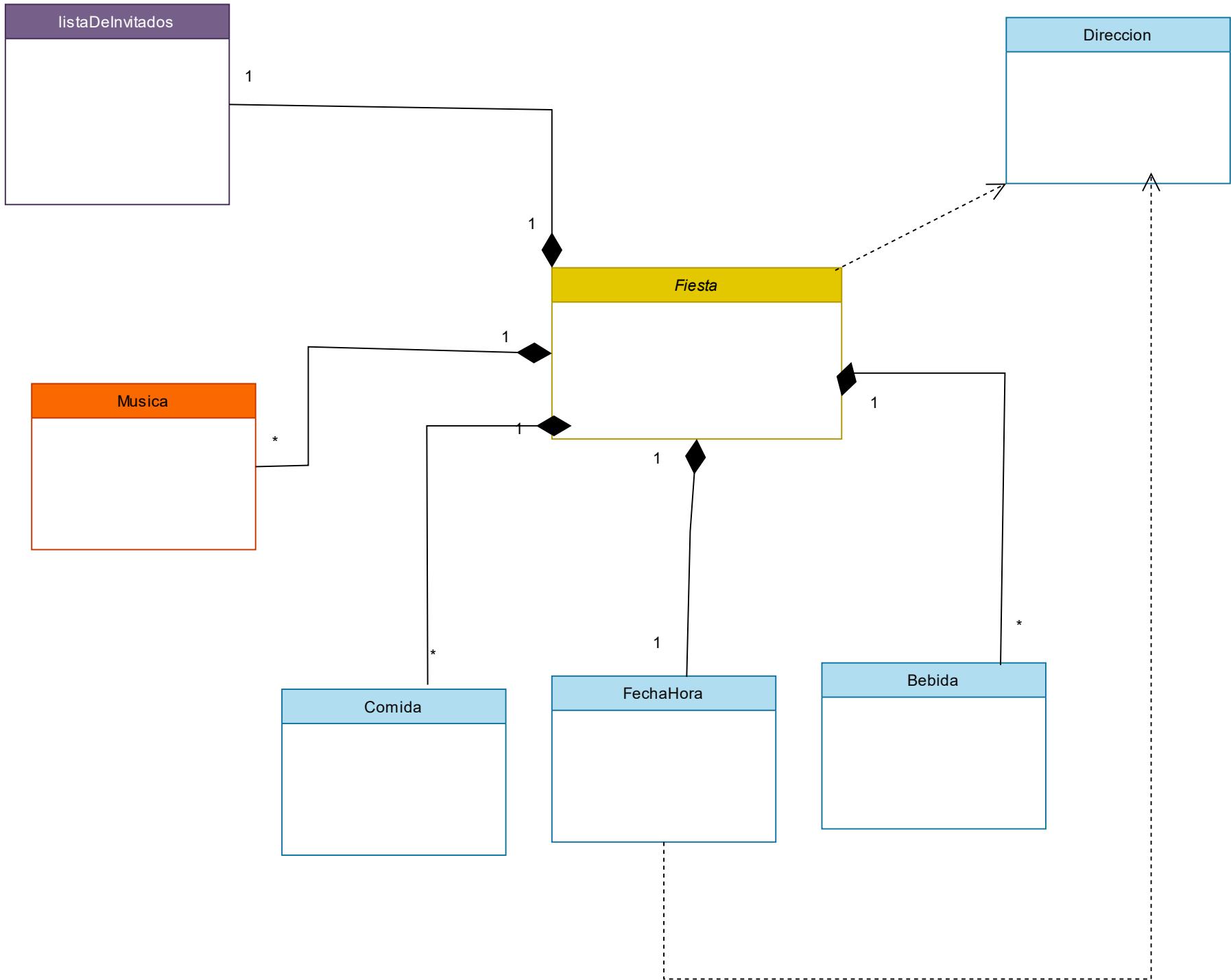
```
1 import POO.Persona;           // Importa la clase Persona del paquete POO.
2 import POO.Profesor;          // Importa la clase Profesor del paquete POO.
3 import POO.Alumno;            // Importa la clase Alumno del paquete POO.
4 import POO.Practicante;       // Importa la clase Practicante del paquete POO.
5
6 // Clase Main
7 public class Main {
8
9     public static void main(String[] args) {
10         // Creando instancias de Persona, Alumno, Profesor y Practicante
11
12         Persona persona = new Persona( nombre: "Pedro", apellidos: "Ramírez", edad: 19);
13
14         Alumno alumno = new Alumno( nombre: "Pedro", apellidos: "Ramírez", edad: 19);
15         alumno.setIdAlumno("A98765"); // Establecer ID de alumno
16
17         Profesor profesor = new Profesor( nombre: "María", apellidos: "García", edad: 40);
18         profesor.setIdProfesor("P12345"); // Establecer ID de profesor
19
20         Practicante practicante = new Practicante( nombre: "Carlos", apellidos: "López", edad: 25);
21         practicante.setIdPracticante("PA6789"); // Establecer ID de practicante
22
23         // Mostrando la información de cada instancia
24         System.out.println("\nInformación de la Persona:");
25         System.out.println("Nombre: " + persona.getNombre());
26         System.out.println("Apellidos: " + persona.getApellidos());
```



```
27         System.out.println("Edad: " + persona.getEdad());
28
29         System.out.println("\nInformación del Alumno:");
30         System.out.println("Nombre: " + alumno.getNombre());
31         System.out.println("Apellidos: " + alumno.getApellidos());
32         System.out.println("Edad: " + alumno.getEdad());
33         System.out.println("ID: " + alumno.getIdAlumno());
34
35         System.out.println("\nInformación del Profesor:");
36         System.out.println("Nombre: " + profesor.getNombre());
37         System.out.println("Apellidos: " + profesor.getApellidos());
38         System.out.println("Edad: " + profesor.getEdad());
39         System.out.println("ID: " + profesor.getIdProfesor());
40
41         System.out.println("\nInformación del Practicante:");
42         System.out.println("Nombre: " + practicante.getNombre());
43         System.out.println("Apellidos: " + practicante.getApellidos());
44         System.out.println("Edad: " + practicante.getEdad());
45         System.out.println("ID: " + practicante.getIdPracticante());
46     }
47 }
```

Prueba

```
Run  Main  X
      |  :  :
↑  Información de la Persona:
↓  Nombre: Pedro
≡  Apellidos: Ramírez
☰  Edad: 19
      Información del Alumno:
      Nombre: Pedro
      Apellidos: Ramírez
      Edad: 19
      ID: A98765
      Información del Profesor:
      Nombre: María
      Apellidos: García
      Edad: 40
      ID: P12345
      Información del Practicante:
      Nombre: Carlos
      Apellidos: López
      Edad: 25
      ID: PA6789
```



Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

EDER LÓPEZ VILLARREAL

ERIC JHONATHAN ANAYA
MARQUEZ

5H T/V

DOCENTE

MTRO. SAUL OLAF LOAIZA

Desarrollo clase 3 Clase
Figura

17/02/2024

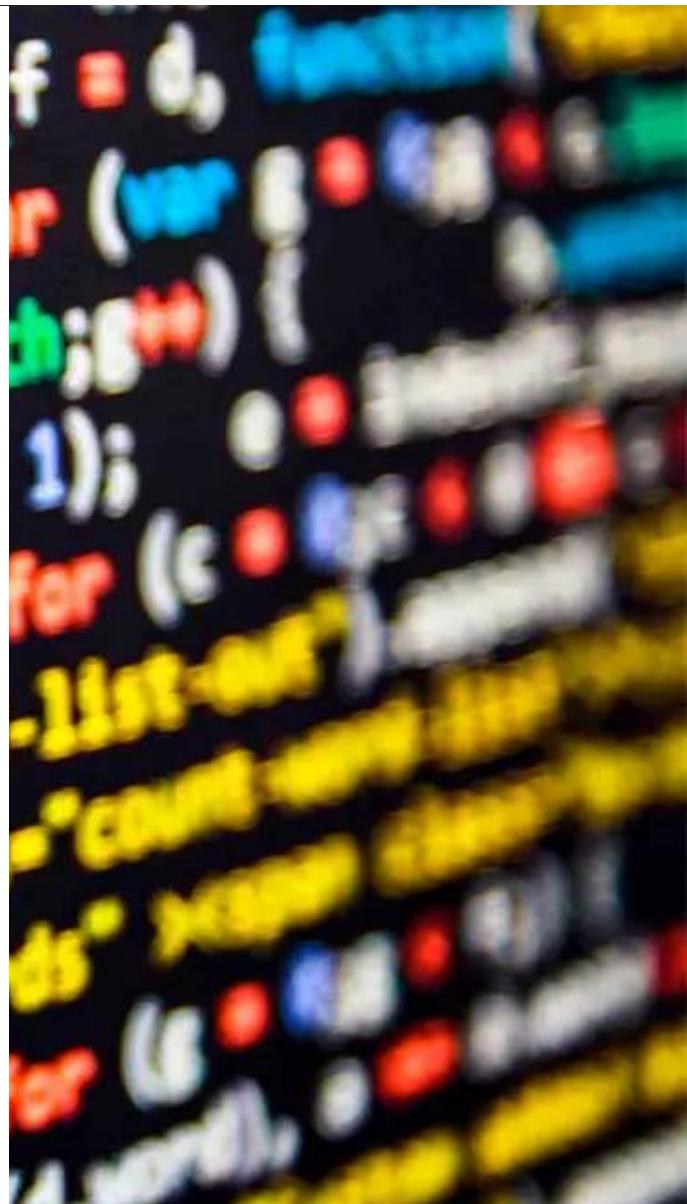
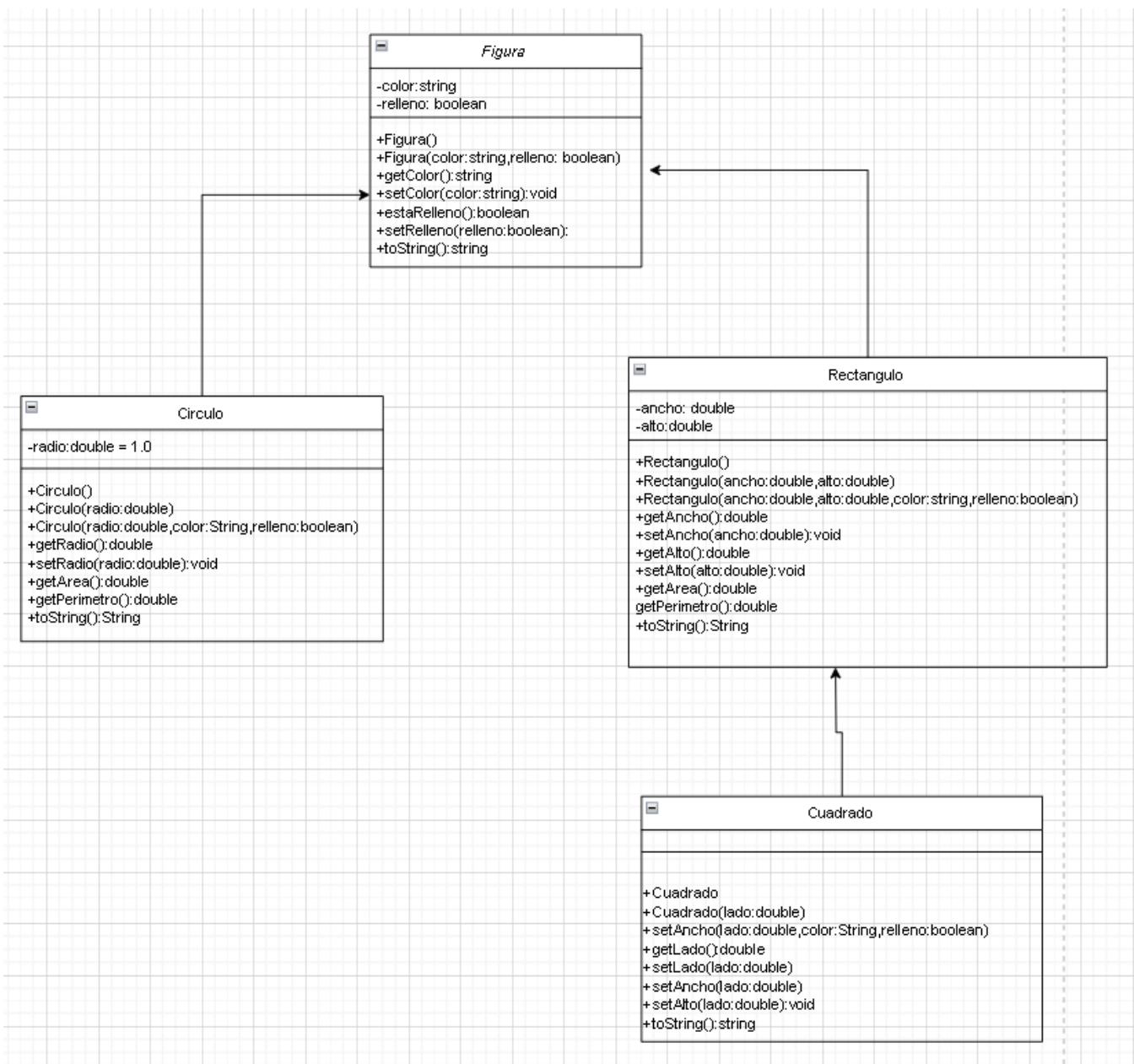


Diagrama UML del clasificador de la clase Figura

Con herencia multinivel y jerarquía



Código de clase Figura

```
Desarrollo clase 3 Clase Figura < Version control < Current

Main.java Figura.java Circulo.java Rectangulo.java Cuadrado.java

1 package POO;
2
3 @public class Figura {
4     5 usages
5     private String color; // Color de la figura
6     5 usages
7     private boolean relleno; // Indica si la figura está rellena o no
8
9     // Constructor sin argumentos que inicializa la figura como roja y rellena
10    5 usages
11    public Figura() {
12        this.color = "Rojo";
13        this.relleno = true;
14    }
15
16    // Constructor que permite especificar el color y si la figura está rellena
17    3 usages
18    public Figura(String color, boolean relleno) {
19        this.color = color;
20        this.relleno = relleno;
21    }
22
23    // Método para obtener el color de la figura
24    no usages
25    public String getColor() {
26        return color;
27    }
28
29    // Método para establecer el color de la figura
30    no usages
31    public void setColor(String color) {
32        this.color = color;
33    }
34
35    // Método para verificar si la figura está rellena
36    no usages
37    public boolean estaRelleno() {
38        return relleno;
39    }
40
41    // Método para establecer si la figura está rellena o no
42    no usages
43    public void setRelleno(boolean relleno) {
44        this.relleno = relleno;
45    }
46
47    // Método toString para representar la figura como una cadena
48    3 overrides
49    @Override
50    public String toString() {
51        String estadoRelleno = relleno ? "rellena" : "no rellena";
52        return "Soy una figura de color " + color + " y " + estadoRelleno;
53    }
54}
```

```
Desarrollo clase 3 Clase Figura < Version control < Current

Main.java Figura.java Circulo.java Rectangulo.java Cuadrado.java

25 public void setColor(String color) {
26     this.color = color;
27 }
28
29 // Método para verificar si la figura está rellena
30 no usages
31 public boolean estaRelleno() {
32     return relleno;
33 }
34
35 // Método para establecer si la figura está rellena o no
36 no usages
37 public void setRelleno(boolean relleno) {
38     this.relleno = relleno;
39 }
40
41 // Método toString para representar la figura como una cadena
42 3 overrides
43 @Override
44 public String toString() {
45     String estadoRelleno = relleno ? "rellena" : "no rellena";
46     return "Soy una figura de color " + color + " y " + estadoRelleno;
47 }
```

Código de clase Circulo

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Desarrollo clase 3 Clase Figura
- File List:** Main.java, Figura.java, Círculo.java (highlighted), Rectangulo.java, Cuadrado.java
- Code Area:**

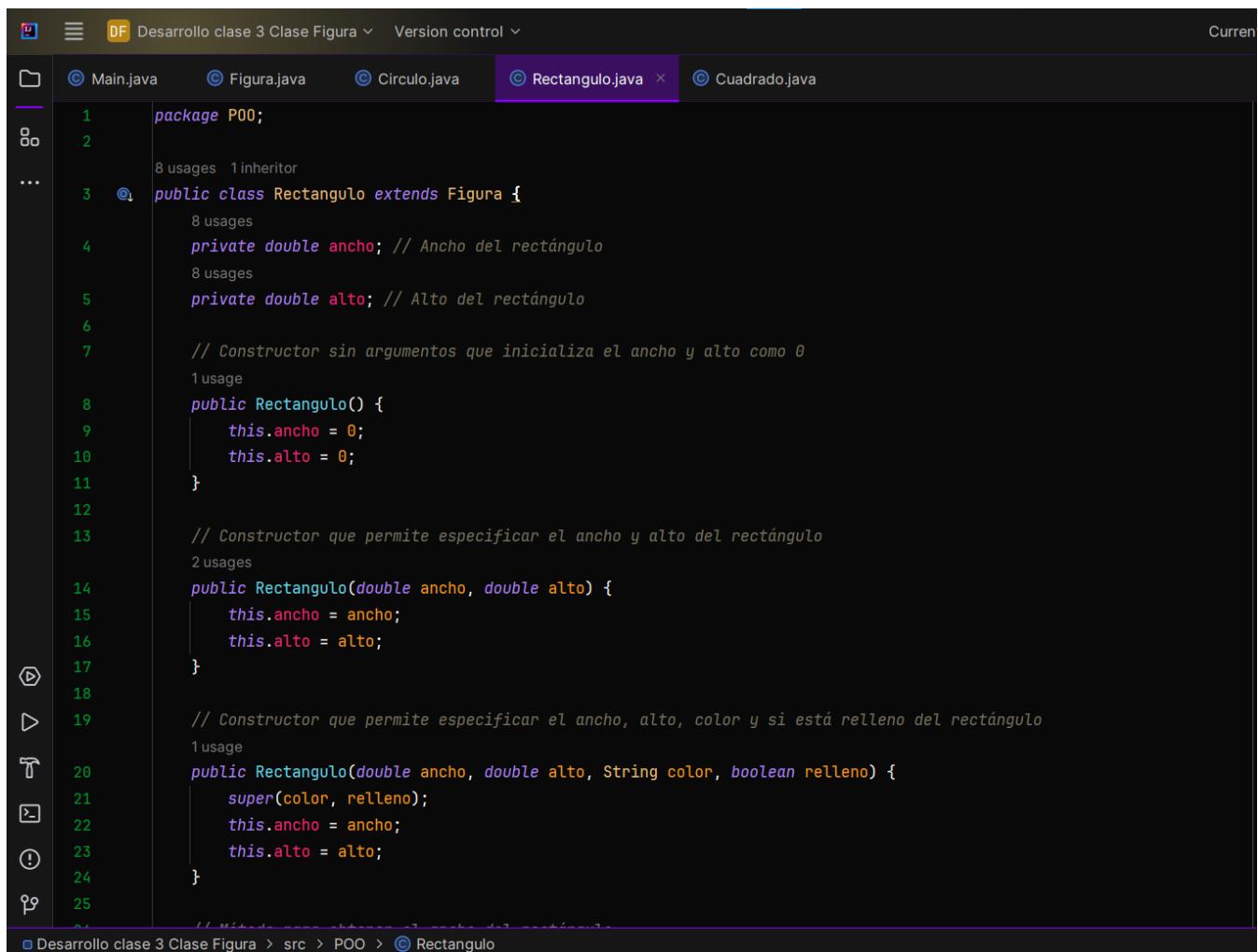
```
1 package POO;
2
3     7 usages
4
5     public class Círculo extends Figura {
6         9 usages
7             private double radio; // Radio del círculo
8
9
10            // Constructor sin argumentos que inicializa el radio como 1.0
11            1 usage
12            public Círculo() {
13                this.radio = 1.0;
14            }
15
16            // Constructor que permite especificar el radio del círculo
17            1 usage
18            public Círculo(double radio) {
19                this.radio = radio;
20            }
21
22            // Método para obtener el radio del círculo
23            no usages
24            public double getRadio() {
25                return radio;
26            }
```
- Status Bar:** Desarrollo clase 3 Clase Figura > src > POO > Círculo

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Desarrollo clase 3 Clase Figura
- File List:** Main.java, Figura.java, Círculo.java (highlighted), Rectangulo.java, Cuadrado.java
- Code Area:**

```
26
27        // Método para establecer el radio del círculo
28        no usages
29        public void setRadio(double radio) {
30            this.radio = radio;
31        }
32
33        // Método para calcular el área del círculo
34        3 usages
35        public double getArea() {
36            return Math.PI * radio * radio;
37        }
38
39        // Método para calcular el perímetro del círculo
40        3 usages
41        public double getPerimetro() {
42            return 2 * Math.PI * radio;
43        }
44
45        // Método toString para representar el círculo como una cadena
46        @Override
47        public String toString() {
48            return "\nSoy un círculo con radio = " + radio + ", esta es mi superclase: " + super.toString();
49        }
```
- Status Bar:** Desarrollo clase 3 Clase Figura > src > POO > Círculo

Código de clase Rectangulo



The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Desarrollo clase 3 Clase Figura > Version control
- File List:** Main.java, Figura.java, Circulo.java, Rectangulo.java (highlighted), Cuadrado.java
- Code Area:**

```
1 package POO;
2
3     8 usages 1 inheritor
4     @ public class Rectangulo extends Figura {
5         8 usages
6         private double ancho; // Ancho del rectángulo
7         8 usages
8         private double alto; // Alto del rectángulo
9
10        // Constructor sin argumentos que inicializa el ancho y alto como 0
11        1 usage
12        public Rectangulo() {
13            this.ancho = 0;
14            this.alto = 0;
15        }
16
17        // Constructor que permite especificar el ancho y alto del rectángulo
18        2 usages
19        public Rectangulo(double ancho, double alto) {
20            this.ancho = ancho;
21            this.alto = alto;
22        }
23
24        // Constructor que permite especificar el ancho, alto, color y si está relleno del rectángulo
25        1 usage
26        public Rectangulo(double ancho, double alto, String color, boolean relleno) {
27            super(color, relleno);
28            this.ancho = ancho;
29            this.alto = alto;
30        }
31    }
```
- Toolbars and Icons:** On the left side, there are various icons for navigation, search, and file operations.
- Status Bar:** Desarrollo clase 3 Clase Figura > src > POO > Rectangulo

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Desarrollo clase 3 Clase Figura > Version control
- File List:** Main.java, Figura.java, Circulo.java, **Rectangulo.java**, Cuadrado.java
- Code Area:** Displays the content of the Rectangulo.java file.

```
26 // Método para obtener el ancho del rectángulo
27 1 usage
28 public double getAncho() {
29     return ancho;
30 }
31
32 // Método para establecer el ancho del rectángulo
33 2 usages 1 override
34 public void setAncho(double ancho) {
35     this.ancho = ancho;
36 }
37
38 // Método para obtener el alto del rectángulo
39 no usages
40 public double getAlto() {
41     return alto;
42 }
43
44 // Método para establecer el alto del rectángulo
45 2 usages 1 override
46 public void setAlto(double alto) {
47     this.alto = alto;
48 }
49
50 // Método para calcular el área del rectángulo
51 4 usages
52 public double getArea() {
53     return ancho * alto;
54 }
55
56 // Método toString para representar el rectángulo como una cadena
57 1 override
58 @Override
59 public String toString() {
60     return "\nSoy un rectángulo con ancho = " + ancho + " y alto = " + alto + ", esta es mi superclase: " + super.toString();
61 }
```

Below the code area, there is a breadcrumb navigation: Desarrollo clase 3 Clase Figura > src > POO > Rectangulo.java

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Desarrollo clase 3 Clase Figura > Version control
- File List:** Main.java, Figura.java, Circulo.java, **Rectangulo.java**, Cuadrado.java
- Code Area:** Displays the content of the Rectangulo.java file.

```
52 public double getPerimetro() {
53     return 2 * (ancho + alto);
54 }
55
56 // Método toString para representar el rectángulo como una cadena
57 1 override
58 @Override
59 public String toString() {
60     return "\nSoy un rectángulo con ancho = " + ancho + " y alto = " + alto + ", esta es mi superclase: " + super.toString();
61 }
```

Código de clase Cuadrado

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Desarrollo clase 3 Clase Figura > Version control
- File List:** Main.java, Figura.java, Circulo.java, Rectangulo.java, Cuadrado.java
- Cuadrado.java Content:**

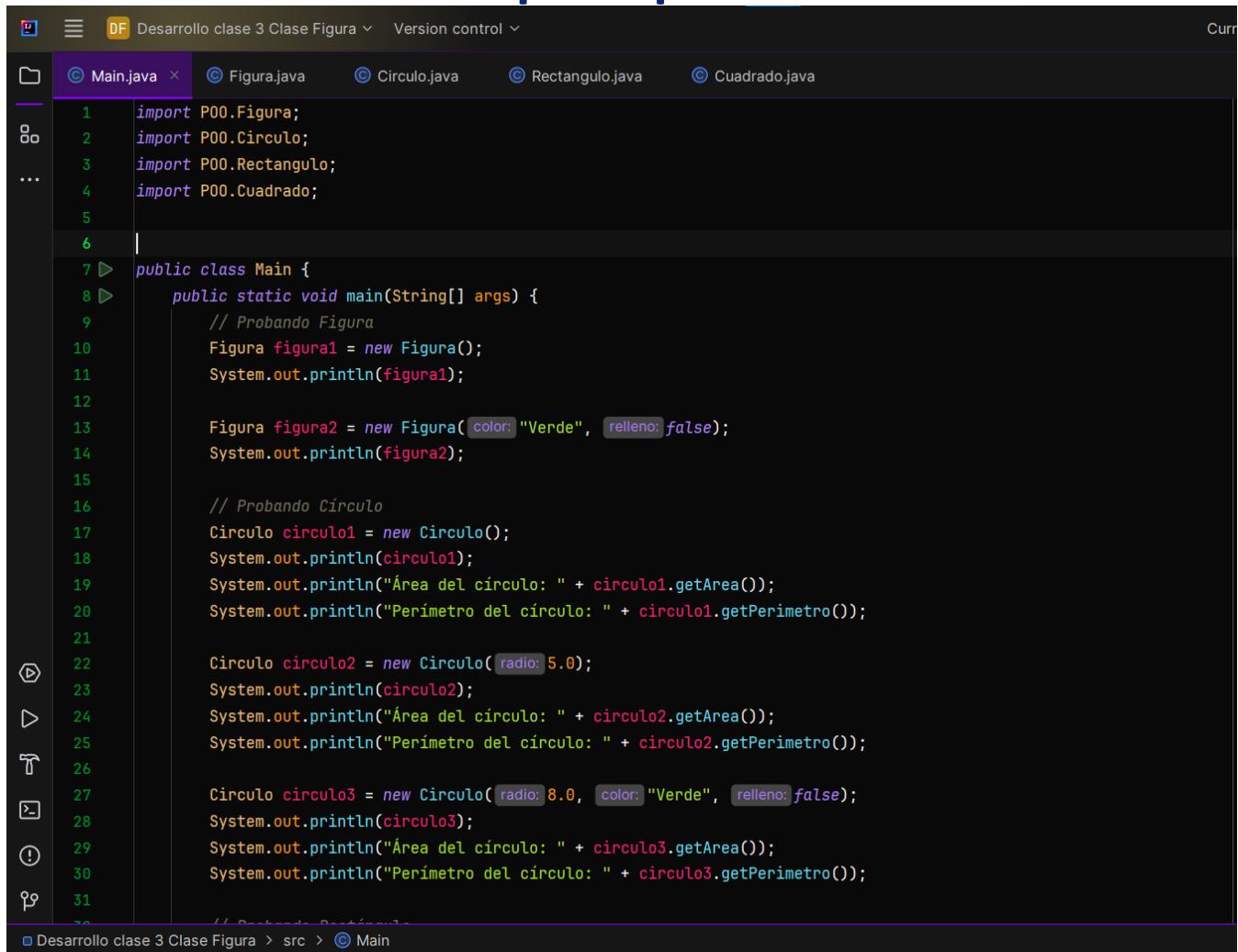
```
1 package POO;
2
3     3 usages
4
5 public class Cuadrado extends Rectangulo {
6     // Constructor que inicializa el cuadrado con un lado dado
7     1 usage
8     public Cuadrado(double lado) {
9         super(lado, lado); // llamada al constructor de la superclase Rectangulo
10    }
11
12    // Sobreescribe el método toString para representar el cuadrado como una cadena
13    @Override
14    public String toString() {
15        return "\nSoy un cuadrado con lado = " + getAncho() + ", esta es mi superclase: " + super.toString();
16    }
17
18    // Sobreescribe el método setAncho para mantener el lado del cuadrado igual al ancho
19    @Override
20    public void setAncho(double lado) {
21        super.setAncho(lado);
22        super.setAlto(lado);
23    }
24
25    // Sobreescribe el método setAlto para mantener el lado del cuadrado igual al alto
26    @Override
27    public void setAlto(double lado) {
28        super.setAncho(lado);
29        super.setAlto(lado);
30    }
31}
```
- Bottom Status Bar:** Desarrollo clase 3 Clase Figura > src > POO > Cuadrado

The screenshot shows a Java code editor interface with the following details:

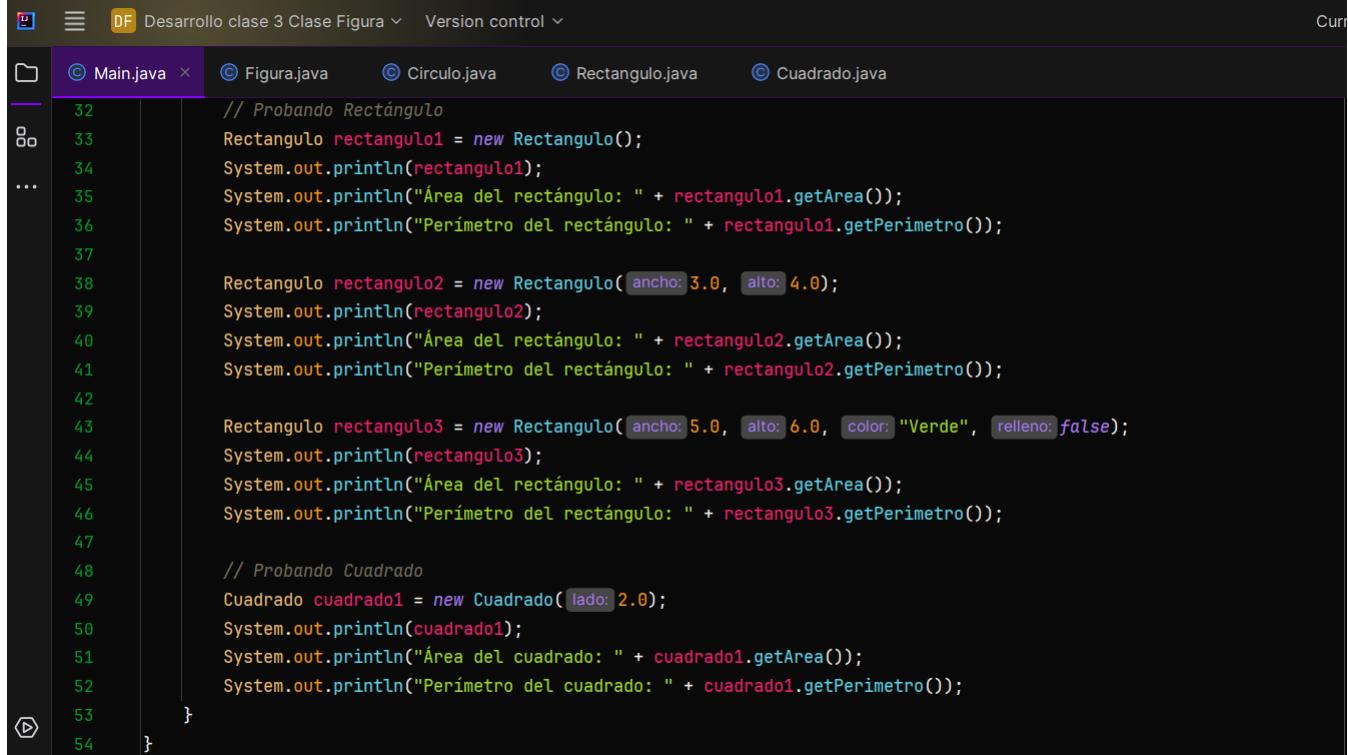
- Title Bar:** Desarrollo clase 3 Clase Figura > Version control
- File List:** Main.java, Figura.java, Circulo.java, Rectangulo.java, Cuadrado.java
- Cuadrado.java Content:**

```
28 }
29 }
```

Clase principal Main



```
1 import POO.Figura;
2 import POO.Circulo;
3 import POO.Rectangulo;
4 import POO.Cuadrado;
5
6
7 public class Main {
8     public static void main(String[] args) {
9         // Probando Figura
10        Figura figura1 = new Figura();
11        System.out.println(figura1);
12
13        Figura figura2 = new Figura(color: "Verde", relleno: false);
14        System.out.println(figura2);
15
16        // Probando Círculo
17        Circulo circulo1 = new Circulo();
18        System.out.println(circulo1);
19        System.out.println("Área del círculo: " + circulo1.getArea());
20        System.out.println("Perímetro del círculo: " + circulo1.getPerimetro());
21
22        Circulo circulo2 = new Circulo(radio: 5.0);
23        System.out.println(circulo2);
24        System.out.println("Área del círculo: " + circulo2.getArea());
25        System.out.println("Perímetro del círculo: " + circulo2.getPerimetro());
26
27        Circulo circulo3 = new Circulo(radio: 8.0, color: "Verde", relleno: false);
28        System.out.println(circulo3);
29        System.out.println("Área del círculo: " + circulo3.getArea());
30        System.out.println("Perímetro del círculo: " + circulo3.getPerimetro());
31    }
32
33    // Probando Rectángulo
34    Rectangulo rectangulo1 = new Rectangulo();
35    System.out.println(rectangulo1);
36    System.out.println("Área del rectángulo: " + rectangulo1.getArea());
37    System.out.println("Perímetro del rectángulo: " + rectangulo1.getPerimetro());
38
39    Rectangulo rectangulo2 = new Rectangulo(ancho: 3.0, alto: 4.0);
40    System.out.println(rectangulo2);
41    System.out.println("Área del rectángulo: " + rectangulo2.getArea());
42    System.out.println("Perímetro del rectángulo: " + rectangulo2.getPerimetro());
43
44    Rectangulo rectangulo3 = new Rectangulo(ancho: 5.0, alto: 6.0, color: "Verde", relleno: false);
45    System.out.println(rectangulo3);
46    System.out.println("Área del rectángulo: " + rectangulo3.getArea());
47    System.out.println("Perímetro del rectángulo: " + rectangulo3.getPerimetro());
48
49    // Probando Cuadrado
50    Cuadrado cuadrado1 = new Cuadrado(lado: 2.0);
51    System.out.println(cuadrado1);
52    System.out.println("Área del cuadrado: " + cuadrado1.getArea());
53    System.out.println("Perímetro del cuadrado: " + cuadrado1.getPerimetro());
54 }
```



```
32
33    // Probando Rectángulo
34    Rectangulo rectangulo1 = new Rectangulo();
35    System.out.println(rectangulo1);
36    System.out.println("Área del rectángulo: " + rectangulo1.getArea());
37    System.out.println("Perímetro del rectángulo: " + rectangulo1.getPerimetro());
38
39    Rectangulo rectangulo2 = new Rectangulo(ancho: 3.0, alto: 4.0);
40    System.out.println(rectangulo2);
41    System.out.println("Área del rectángulo: " + rectangulo2.getArea());
42    System.out.println("Perímetro del rectángulo: " + rectangulo2.getPerimetro());
43
44    Rectangulo rectangulo3 = new Rectangulo(ancho: 5.0, alto: 6.0, color: "Verde", relleno: false);
45    System.out.println(rectangulo3);
46    System.out.println("Área del rectángulo: " + rectangulo3.getArea());
47    System.out.println("Perímetro del rectángulo: " + rectangulo3.getPerimetro());
48
49    // Probando Cuadrado
50    Cuadrado cuadrado1 = new Cuadrado(lado: 2.0);
51    System.out.println(cuadrado1);
52    System.out.println("Área del cuadrado: " + cuadrado1.getArea());
53    System.out.println("Perímetro del cuadrado: " + cuadrado1.getPerimetro());
54 }
```

Prueba

```
Soy una figura de color Rojo y rellena
Soy una figura de color Verde y no rellena

Soy un círculo con radio = 1.0, esta es mi superclase: Soy una figura de color Rojo y rellena
Área del círculo: 3.141592653589793
Perímetro del círculo: 6.283185307179586

Soy un círculo con radio = 5.0, esta es mi superclase: Soy una figura de color Rojo y rellena
Área del círculo: 78.53981633974483
Perímetro del círculo: 31.41592653589793

Soy un círculo con radio = 8.0, esta es mi superclase: Soy una figura de color Verde y no rellena
Área del círculo: 201.06192982974676
Perímetro del círculo: 50.26548245743669

Soy un rectángulo con ancho = 0.0 y alto = 0.0, esta es mi superclase: Soy una figura de color Rojo y rellena
Área del rectángulo: 0.0
Perímetro del rectángulo: 0.0

Soy un rectángulo con ancho = 3.0 y alto = 4.0, esta es mi superclase: Soy una figura de color Rojo y rellena
Área del rectángulo: 12.0
Perímetro del rectángulo: 14.0

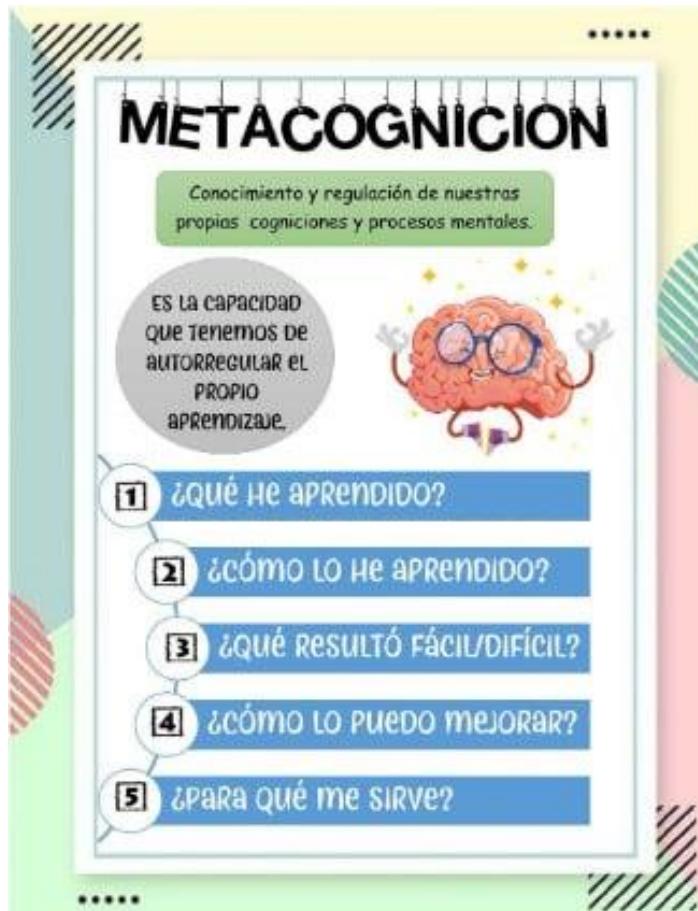
Soy un rectángulo con ancho = 5.0 y alto = 6.0, esta es mi superclase: Soy una figura de color Verde y no rellena
Área del rectángulo: 30.0
Perímetro del rectángulo: 22.0

Soy un cuadrado con lado = 2.0, esta es mi superclase:
Soy un rectángulo con ancho = 2.0 y alto = 2.0, esta es mi superclase: Soy una figura de color Rojo y rellena
```

```
Área del cuadrado: 4.0
Perímetro del cuadrado: 8.0

Process finished with exit code 0
```

AUTOEVALUACIÓN



- 1.- Lo que aprendí a lo largo de este tiempo en clase fue mejorar mi lógica matemática al momento de hacer conversiones de valores, también aprendí a mejorar mi distribución de variables y documentar mejor mis códigos y a diseñar mejor mis diagramas UML.
- 2.- Lo aprendí en clase con explicaciones que daba el profesor.
- 3.- Lo que más me resultó difícil fue diseñar bien los diagramas UML y dar sus herencias o relaciones.
- 4.- Puedo mejorar si me dedico más tiempo a practicar programación en mis tiempos libres.
- 5.- Esto me va a servir para seguir motivándome a seguir programando y cumplir mis objetivos.

AUTOEVALUACIÓN

¿CÓMO LO HE APRENDIDO?

Lo aprendí en clase y tanto por mis propios medios viendo videos en YouTube

¿CÓMO LO PUEDO MEJORAR?

Practicando más en mis tiempos libres

¿QUÉ HE APRENDIDO?

Aprendí a mejorar mi lógica matemática para las conversaciones de valores, y a usar mejor mis valores, definir bien mis constructores y diseñar diagramas UML

¿QUÉ RESULTÓ FÁCIL/ DIFÍCIL?

Me resultó algo difícil el diseñar las relaciones o herencias en los diagramas UML

Todo esto me va a servir para motivarme más y seguir a delante

¿PARA QUÉ SIRVE?