INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE ADMINISTRACIÓN DE TECNOLOGÍA DE INFORMACIÓN

TI-2201 | PROGRAMACIÓN ORIENTADA A OBJETOS

PROFESOR:

Luis Javier Chavarría Sánchez

ACTIVIDAD ASINCRÓNICA 5

ESTUDIANTE:

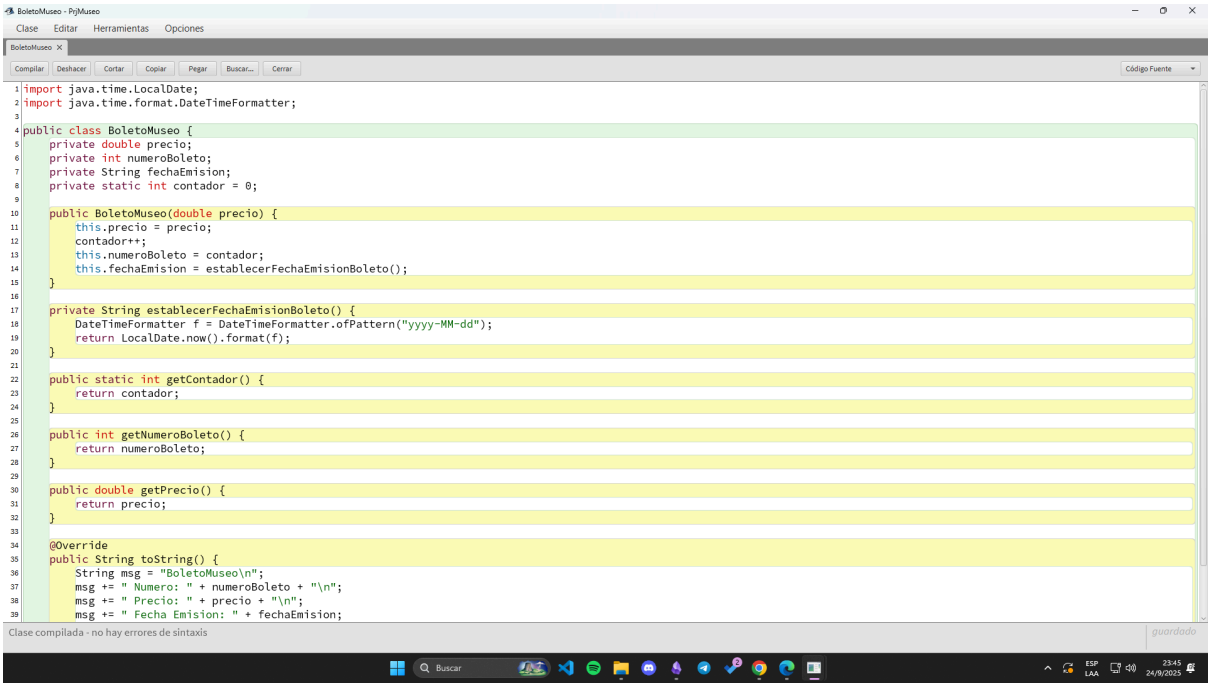
Eliam Vives Vallejos 2023172541

II Semestre

2025

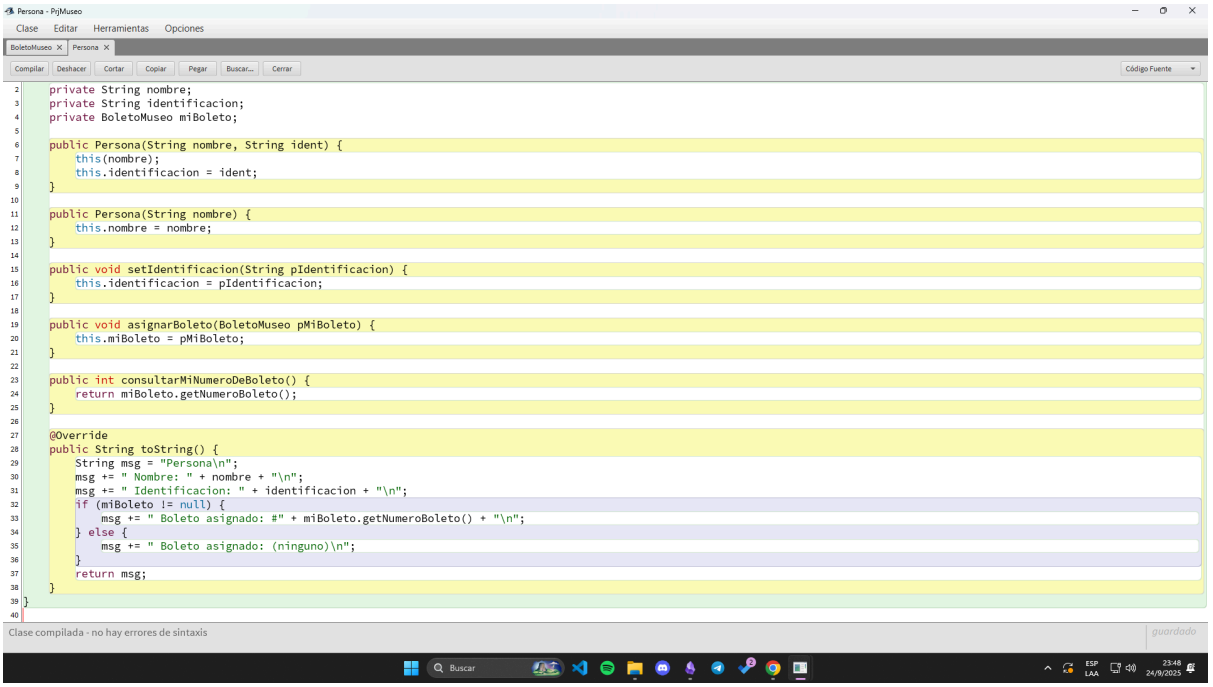
Capturas de Compilación de Código previo al Javadoc

Clase BoletoMuseo



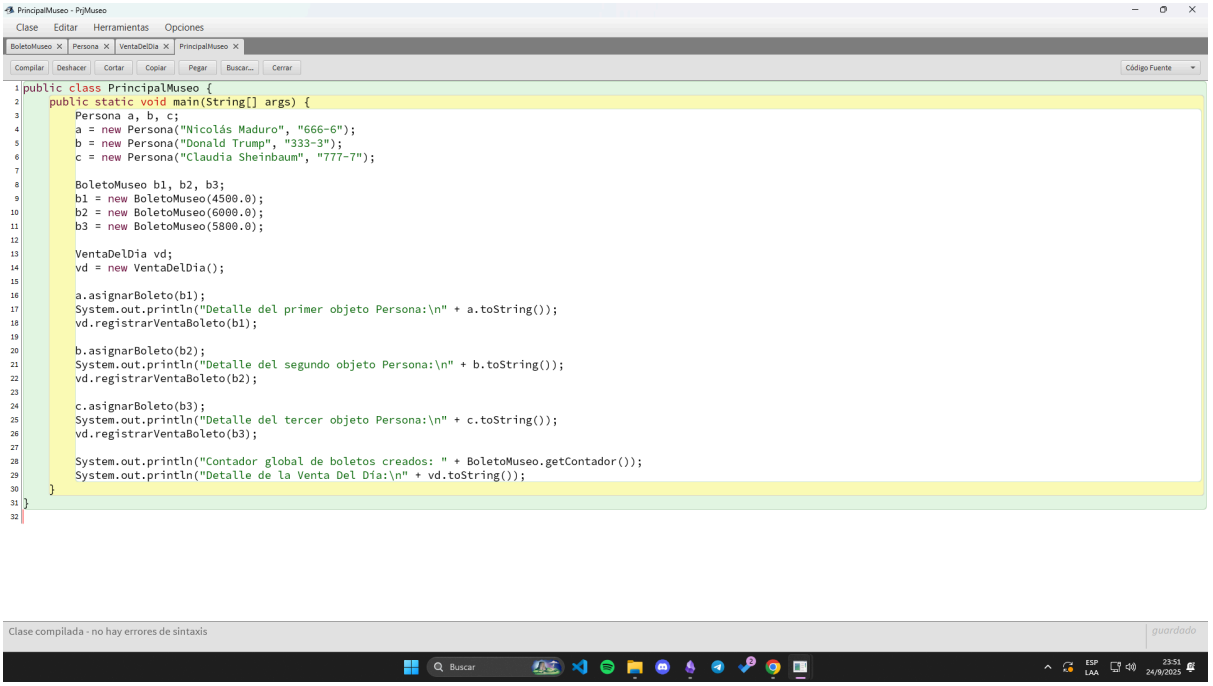
```java
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class BoletoMuseo {
    private double precio;
    private int numeroBoleto;
    private String fechaEmision;
    private static int contador = 0;

    public BoletoMuseo(double precio) {
        this.precio = precio;
        contador++;
        this.numeroBoleto = contador;
        this.fechaEmision = establecerFechaEmisionBoleto();
    }

    private String establecerFechaEmisionBoleto() {
        DateTimeFormatter f = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        return LocalDate.now().format(f);
    }

    public static int getContador() {
        return contador;
    }

    public int getNumeroBoleto() {
        return numeroBoleto;
    }

    public double getPrecio() {
        return precio;
    }

    @Override
    public String toString() {
        String msg = "BoletoMuseo\n";
        msg += " Numero: " + numeroBoleto + "\n";
        msg += " Precio: " + precio + "\n";
        msg += " Fecha Emision: " + fechaEmision;
```

Clase Persona



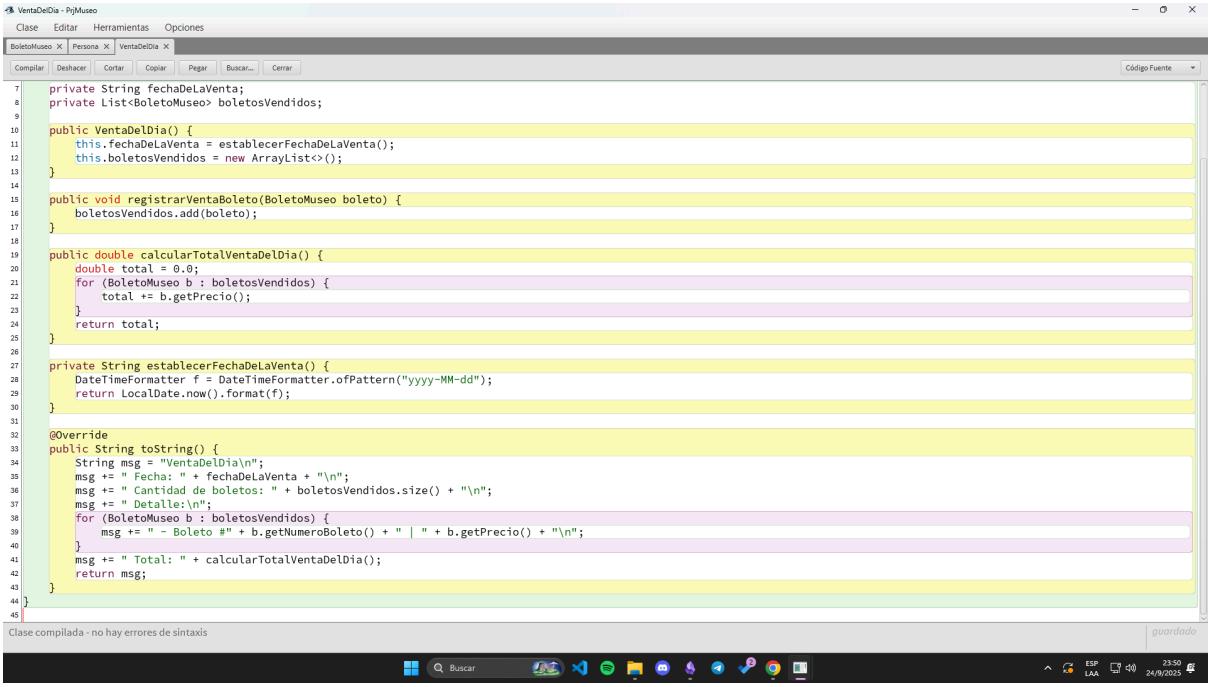
```java
    private String nombre;
    private String identificacion;
    private BoletoMuseo miBoleto;

    public Persona(String nombre, String ident) {
        this(nombre);
        this.identificacion = ident;
    }

    public Persona(String nombre) {
        this.nombre = nombre;
    }

    public void setIdentificacion(String pIdentificacion) {
        this.identificacion = pIdentificacion;
    }

    public void asignarBoleto(BoletoMuseo pMiBoleto) {
        this.miBoleto = pMiBoleto;
    }

    public int consultarMiNumeroDeBoleto() {
        return miBoleto.getNumeroBoleto();
    }

    @Override
    public String toString() {
        String msg = "Persona\n";
        msg += " Nombre: " + nombre + "\n";
        msg += " Identificacion: " + identificacion + "\n";
        if (miBoleto != null) {
            msg += " Boleto asignado: #" + miBoleto.getNumeroBoleto() + "\n";
        } else {
            msg += " Boleto asignado: (ninguno)\n";
        }
        return msg;
    }
}
```

PrincipalMuseo



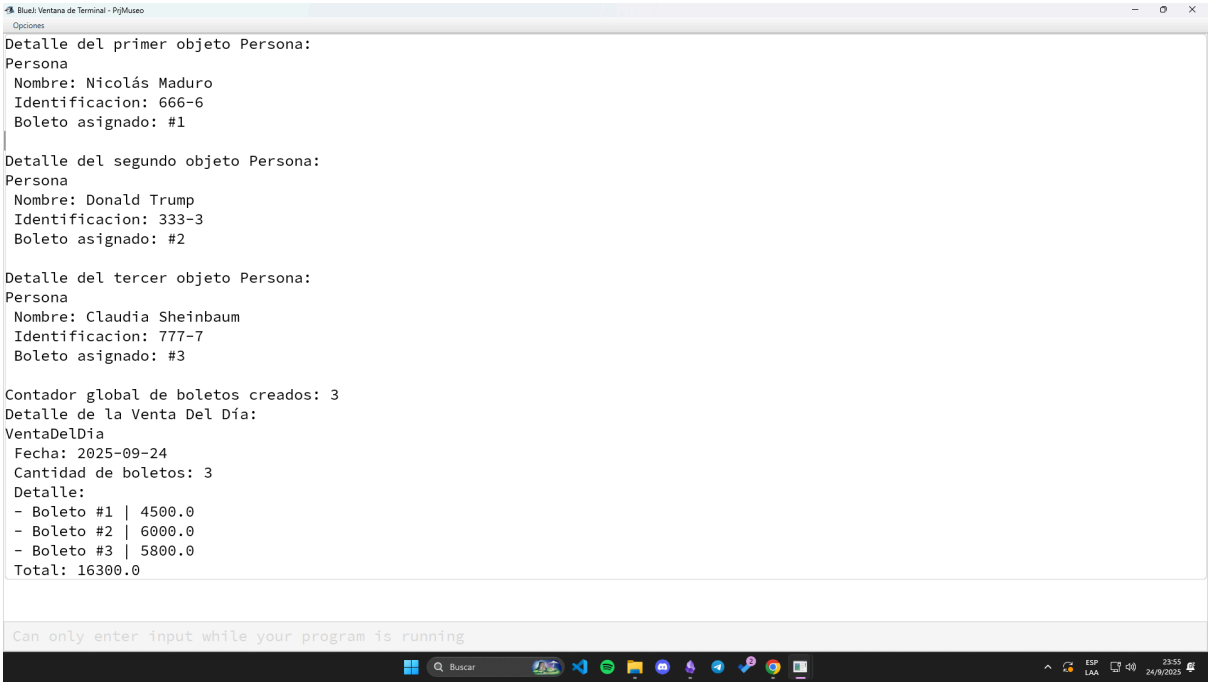
```java
public class PrincipalMuseo {
    public static void main(String[] args) {
        Persona a, b, c;
        a = new Persona("Nicolás Maduro", "666-6");
        b = new Persona("Donald Trump", "333-3");
        c = new Persona("Claudia Sheinbaum", "777-7");

        BoletoMuseo b1, b2, b3;
        b1 = new BoletoMuseo(4500.0);
        b2 = new BoletoMuseo(6000.0);
        b3 = new BoletoMuseo(5800.0);

        VentaDelDia vd;
        vd = new VentaDelDia();

        a.asignarBoleto(b1);
        System.out.println("Detalle del primer objeto Persona:\n" + a.toString());
        vd.registrarVentaBoleto(b1);

        b.asignarBoleto(b2);
        System.out.println("Detalle del segundo objeto Persona:\n" + b.toString());
        vd.registrarVentaBoleto(b2);

        c.asignarBoleto(b3);
        System.out.println("Detalle del tercer objeto Persona:\n" + c.toString());
        vd.registrarVentaBoleto(b3);

        System.out.println("Contador global de boletos creados: " + BoletoMuseo.getContador());
        System.out.println("Detalle de la Venta Del Día:\n" + vd.toString());
    }
}
```

VentaDelDia



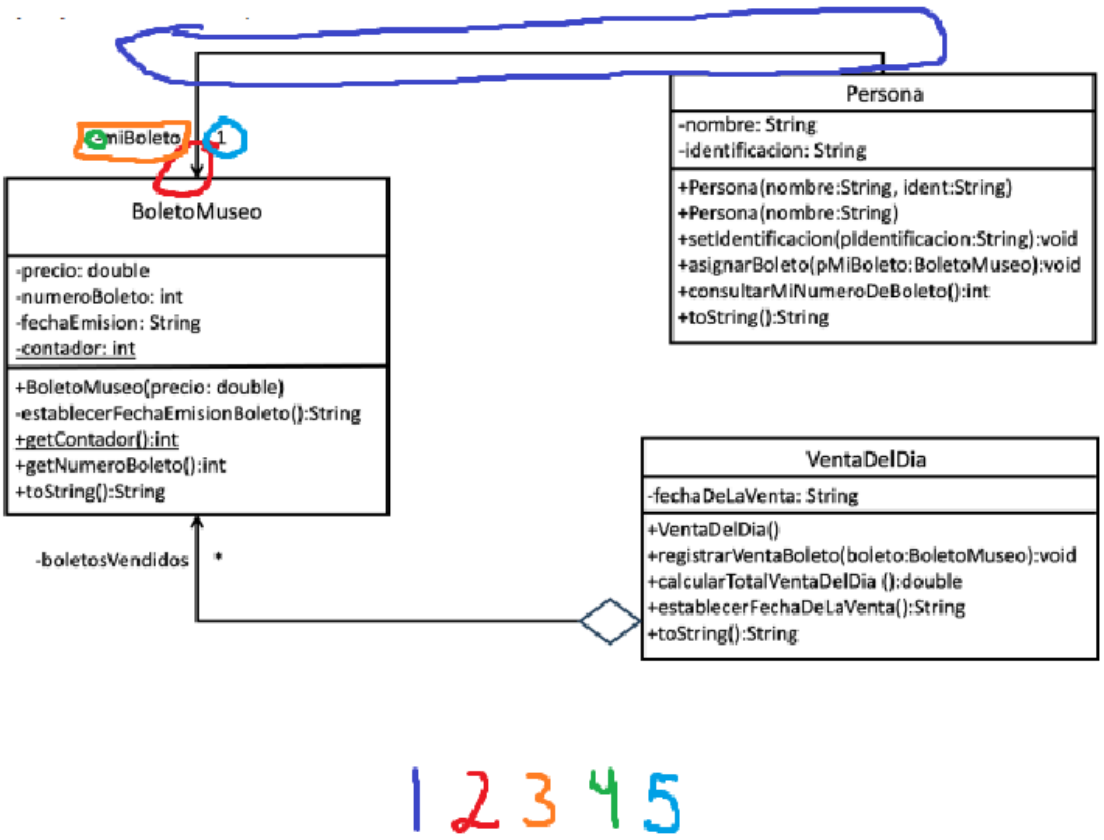
```java
    private String fechaDeLaVenta;
    private List<BoletoMuseo> boletosVendidos;

    public VentaDelDia() {
        this.fechaDeLaVenta = establecerFechaDeLaVenta();
        this.boletosVendidos = new ArrayList<>();
    }

    public void registrarVentaBoleto(BoletoMuseo boleto) {
        boletosVendidos.add(boleto);
    }

    public double calcularTotalVentaDelDia() {
        double total = 0.0;
        for (BoletoMuseo b : boletosVendidos) {
            total += b.getPrecio();
        }
        return total;
    }

    private String establecerFechaDeLaVenta() {
        DateTimeFormatter f = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        return LocalDate.now().format(f);
    }

    @Override
    public String toString() {
        String msg = "VentaDelDia\n";
        msg += " Fecha: " + fechaDeLaVenta + "\n";
        msg += " Cantidad de boletos: " + boletosVendidos.size() + "\n";
        msg += " Detalle:\n";
        for (BoletoMuseo b : boletosVendidos) {
            msg += " - Boleto #" + b.getNumeroBoleto() + " | " + b.getPrecio() + "\n";
        }
        msg += " Total: " + calcularTotalVentaDelDia();
        return msg;
    }
}
```

Ejecución del Programa



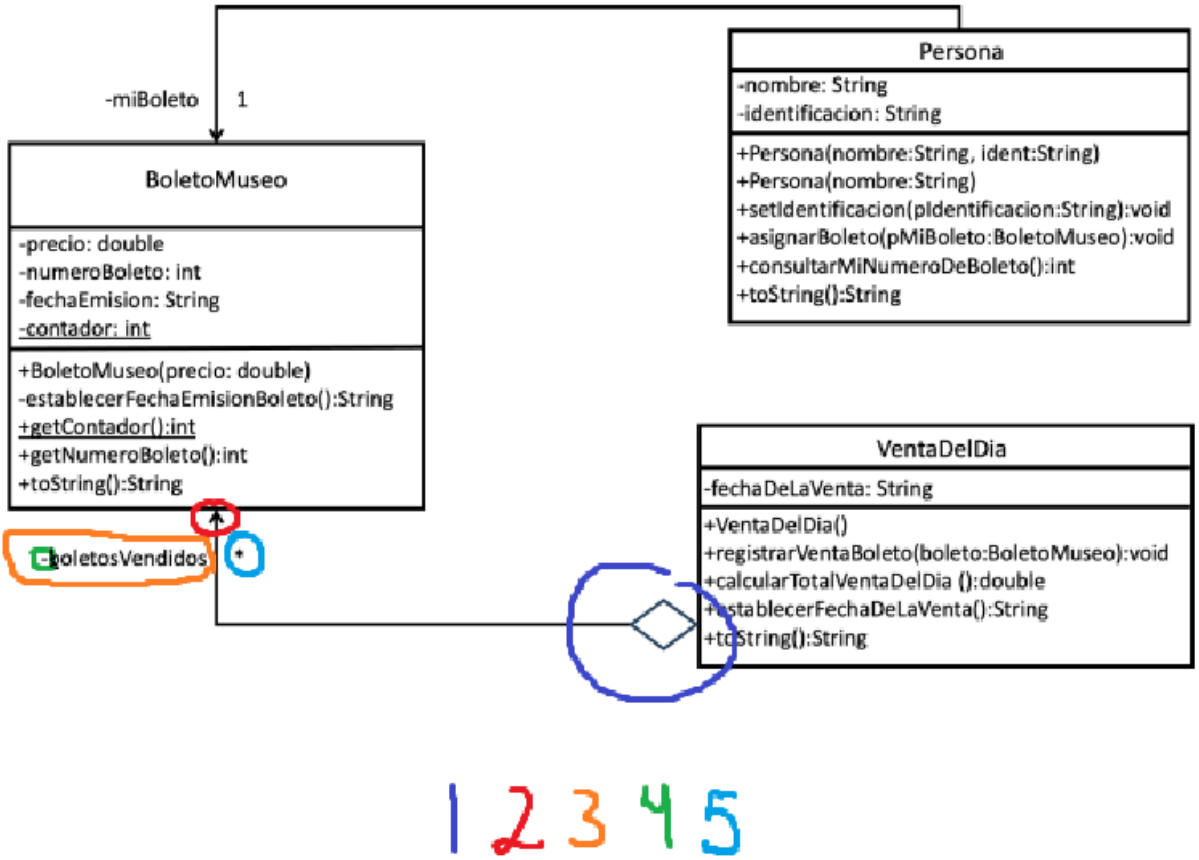
Análisis de Diagramas UML

Elementos de la relación de ASOCIACIÓN



1. Representación UML (la línea continua)
2. Navegabilidad (la punta de flecha abierta)
3. Rol (-miBoleto)
4. Modificador del rol (el signo de -)
5. Cardinalidad (el 1)

Elementos de la relación de AGREGACIÓN



1. Representación UML (el rombo blanco)
2. Navegabilidad (la punta de flecha abierta)
3. Rol (-boletosVendidos)
4. Modificador del rol (el signo de -)
5. Cardinalidad (el *)

Respuestas a Preguntas

a. **Si la clase A está vinculada con la clase B mediante una relación de asociación. ¿La estructura de la clase B se ve impactada? Explique con detalle.**
Respuesta: No, la estructura de la clase B no se ve impactada. En una relación de asociación unidireccional donde A se asocia con B, la clase A es la que se modifica para incluir un atributo cuyo tipo es la clase B. Esto permite que los objetos de tipo A puedan enviar mensajes a un objeto de tipo B. La clase B, sin embargo, no contiene ninguna referencia a la clase A y su código fuente permanece inalterado por la relación; es estructuralmente independiente.

b. **Si la clase P está vinculada con la clase Q mediante una relación de agregación. ¿La estructura de la clase B se ve impactada? Explique con detalle.**
Respuesta: (Suponiendo que la pregunta contiene un error tipográfico y debería referirse a la clase Q). No, la estructura de la clase Q (la PARTE) no se ve impactada. En una agregación, la clase P juega el papel del TODO y se modifica para contener una colección o referencia a objetos de tipo Q. La clase Q, que representa la parte, no necesita conocer al todo y su estructura interna no se modifica. Puede existir de forma independiente.

c. **Si la clase A está vinculada con la clase B mediante una relación de asociación y la clase B está vinculada con la clase A mediante una relación de asociación. ¿La estructura de ambas clases se ve impactada? Explique con detalle.**
Respuesta: Sí, en este caso la estructura de ambas clases se ve impactada. Esto se conoce como una asociación bidireccional. Para que la relación funcione en ambos sentidos, la clase A debe tener un atributo de tipo B, y la clase B debe tener un atributo de tipo A. Por lo tanto, el código fuente de ambas clases debe ser modificado para materializar el vínculo.

d. **Un objeto de tipo Z podría enviar mensajes a otro objeto de tipo W, aun cuando no exista un vínculo (de asociación o agregación) entre la clase Z y la clase W? Explique con detalle.**
Respuesta: Sí, es posible. Aunque no exista una relación estructural permanente como la asociación o la agregación (es decir, un atributo), un objeto de tipo Z puede interactuar con un objeto de tipo W si recibe una referencia a este último a través de un parámetro en uno de sus métodos. Esto se conoce como una relación de dependencia ("usa-un"). El vínculo es temporal y solo dura mientras el método se está ejecutando, pero es suficiente para permitir el envío de mensajes.

e. **En un diagrama de clase con detalles de implementación, suponga que existe una relación de asociación entre la clase P y la clase Q... ¿Eso es suficiente para establecer de forma completa el vínculo de asociación entre P y Q? Explique con detalle.**
Respuesta: No, no es suficiente. El diagrama de clases con detalles de implementación es un diseño, un plano que representa la estructura y las relaciones que deben existir. Para que el vínculo de asociación se establezca de forma "completa" y funcional, es indispensable programar dicha relación en el código fuente. Esto significa que en la clase P se debe declarar un atributo del tipo de la clase Q, materializando así lo que el diagrama especifica. Sin la implementación en el código, la relación solo existe en el papel

Reflexión Final

En esta actividad comprendí que las clases en Java no existen de manera aislada, sino que su verdadero poder surge cuando establecen relaciones entre sí. Pude diferenciar claramente entre asociación y agregación: la primera reflejada en el vínculo entre una Persona y su BoletoMuseo, donde ambos existen de forma independiente pero pueden relacionarse; y la segunda en la VentaDelDia, que agrupa varios boletos como un todo que depende de sus partes. Esto me permitió ver cómo los conceptos de UML se traducen directamente al código mediante atributos y colecciones.

Además, entendí que un diagrama de clases es solo el punto de partida y que la implementación en código es lo que materializa realmente esas relaciones. Al llevarlo a Java, identifiqué atributos, cardinalidades y roles de cada clase, lo que me permitió visualizar de manera concreta la interacción de los objetos. El uso de Javadoc también me mostró la importancia de documentar profesionalmente, no solo con comentarios, sino con un estándar que facilita la creación de una API navegable y clara.

Finalmente, al ejecutar el programa principal pude observar cómo los objetos colaboran entre sí para cumplir un objetivo común. Ver a las personas asignarse boletos y registrar las ventas del día me ayudó a internalizar la idea de que los objetos no son entidades aisladas, sino piezas que cooperan dentro de un sistema. En conjunto, la práctica me permitió pasar de la teoría de UML a la implementación de un sistema funcional, con interacciones reales y una documentación profesional.