**DISEÑO DE SOFTWARE**

**PARALELO 3**

**TAREA 2- SISTEMA ENVIVOTICKETS**

**INTEGRANTES:**

**BARRIOS URETA ROBERTO CARLOS**

**MACIAS MENDOZA CHRISTIAN JAVIER**

[**ROCA MACIAS LUIS ERNESTO**](https://aulavirtual.espol.edu.ec/groups/134660/users/24377)

**TAPIA LOOR PAULO MARCELO**

**PAO II**

**2024-2025**

INDICE

[**Sección A: Identificación de Patrones de Diseño** 2](#_Toc184674325)

[**1. Observer:** 3](#_Toc184674326)

[**2. Decorator**: 3](#_Toc184674327)

[**3. Facade**: 3](#_Toc184674328)

[**Sección B: Diagrama de Casos de Uso** 4](#_Toc184674329)

[**Diagrama de casos de Uso** 4](#_Toc184674330)

[**Detalle de los 3 Casos de Uso Principales** 5](#_Toc184674331)

[1. **Adquirir Boletos** 5](#_Toc184674332)

[**2. Visualizar Disponibilidad de Asientos** 6](#_Toc184674333)

[3. **Configurar Precios y Políticas** 6](#_Toc184674334)

[**Sección C: Diagrama de Clases con Patrones de Diseño** 7](#_Toc184674335)

[**Diagrama de Clases** 7](#_Toc184674336)

[**Sección D: Diagramas de Secuencia** 7](#_Toc184674337)

[**Diagrama de Secuencias de Caso de Uso 1. Adquirir Boletos** 7](#_Toc184674338)

[**Diagrama de Secuencias de Caso de Uso 2. Visualizar Disponibilidad de Asientos** 7](#_Toc184674339)

[**Diagrama de Secuencias de Caso de Uso 3. Configurar Precios y Políticas** 7](#_Toc184674340)

[**Diagrama de Secuencias de Caso de Uso 4. Atender Incidentes de Cliente** 7](#_Toc184674341)

[**Sección E: Generación de Código en Java** 8](#_Toc184674342)

# **Sección A: Identificación de Patrones de Diseño**

### **1. Observer:**

Problema: El sistema necesita notificar a los usuarios sobre cambios en la programación de los eventos, como modificaciones en el elenco, reprogramaciones o cancelaciones.

Justificación:  
Este patrón resuelve el problema de mantener a los usuarios informados sin que el sistema tenga que gestionar manualmente el envío de notificaciones a cada uno.

Funcionamiento:

La clase Evento actúa como el sujeto que notifica cambios.

Las clases Usuario o Administrador implementan la interfaz de observador y reciben actualizaciones.

Cuando se reprograma un evento, Evento notifica automáticamente a los observadores.

Esto asegura que el sistema sea escalable: al agregar más usuarios, no es necesario modificar el código principal de Evento.

### 

### **2. Decorator**:

**Problema:** Los usuarios pueden adquirir boletos con opciones adicionales, como paquetes que incluyen bebidas o estacionamiento, y las configuraciones de precios deben ser flexibles.

**Justificación:**  
El patrón **Decorator** permite añadir dinámicamente funcionalidades a los tickets sin modificar las clases base. Resuelve el problema de gestionar múltiples combinaciones de paquetes adicionales sin crear una clase distinta para cada combinación posible.

**Funcionamiento:**

* + La clase base Ticket representa un boleto simple.
  + Los decoradores como PaqueteConBebidas y PaqueteConEstacionamiento añaden las opciones adicionales al ticket.
  + Se pueden combinar decoradores para permitir múltiples opciones en un solo ticket, como "bebidas + estacionamiento".

Este enfoque mantiene el código limpio, extensible y fácil de mantener cuando se agregan nuevas opciones.

### **3. Facade**:

**Problema:** El sistema integra varias funcionalidades complejas, como gestión de boletos, procesamiento de pagos y generación de notificaciones, lo que podría hacer que la interacción directa con múltiples clases sea complicada para los usuarios y desarrolladores.

**Justificación:**  
El patrón **Facade** proporciona una interfaz unificada para manejar operaciones comunes, como comprar boletos o consultar disponibilidad, ocultando la complejidad interna del sistema. Resuelve el problema de ofrecer una interacción sencilla con un sistema complejo.

**Funcionamiento:**

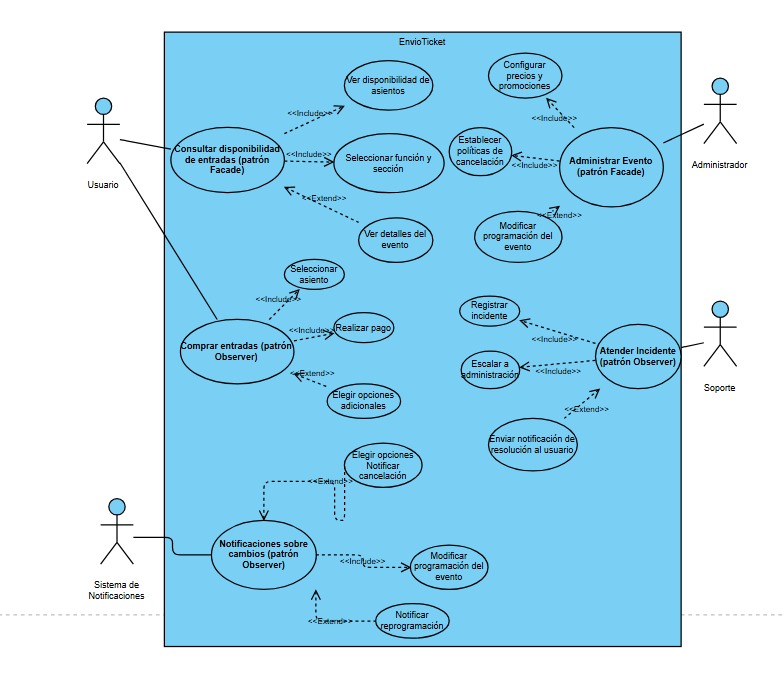
* + La clase SistemaEnVivoTickets actúa como la fachada, exponiendo métodos simples como comprarBoleto() o verDisponibilidad().
  + Internamente, esta clase coordina múltiples subsistemas como Compra, Pago y Notificacion.
  + Los desarrolladores no necesitan interactuar directamente con cada subsistema, lo que reduce el acoplamiento.

Esto facilita la integración de nuevas funcionalidades y mejora la experiencia del usuario final al simplificar la interacción con el sistema.

# **Sección B: Diagrama de Casos de Uso**

## **Diagrama de casos de Uso**

**LINK:** [**https://online.visual-paradigm.com/share.jsp?id=333639363230382d32**](https://online.visual-paradigm.com/share.jsp?id=333639363230382d32)



## **Detalle de los 3 Casos de Uso Principales**

### 1. **Consultar Disponibilidad de Entradas**

* Actores: Usuario
* Precondiciones:
* El evento debe estar registrado en el sistema.
* Deben existir funciones configuradas con secciones y asientos.
* El usuario debe estar autenticado para realizar reservas.
* Flujo de Eventos Principal:

1. El usuario selecciona un evento desde el listado general.
2. El sistema muestra las funciones disponibles para el evento (fecha y horario).
3. El usuario elige una función específica.
4. El sistema muestra las secciones disponibles (VIP, Platea, etc.).
5. El usuario selecciona una sección.
6. El sistema utiliza el patrón Facade para consultar la base de datos de asientos, y muestra un mapa interactivo con el estado de cada asiento (Disponible, Reservado, Agotado).
   * Flujos Alternativos:

* Si no hay boletos disponibles para la función seleccionada, el sistema notifica al

usuario y permite regresar al listado de funciones.

* + Postcondiciones:
* El usuario puede proceder a reservar o comprar boletos
* El estado de los asientos en el mapa refleja la última actualización en la base de datos.

### **2. Comprar Entradas**

* Actores: Usuario, Sistema de Notificaciones
* Precondiciones:
* El usuario debe haber seleccionado previamente los asientos.
* Los asientos seleccionados deben estar disponibles.
* El sistema de pagos debe estar operativo.
* Flujo de Eventos Principal:
  + - 1. El usuario selecciona los asientos desde el mapa interactivo.
      2. El sistema actualiza temporalmente el estado de los asientos a "Reservado".
      3. El usuario confirma la compra y es redirigido al proceso de pago.
      4. El sistema valida la transacción con el proveedor de pagos.
      5. Si el pago es exitoso:
* El estado de los asientos cambia a "Vendido"
* Se utiliza el patrón Observer para notificar al sistema de notificaciones sobre la compra completada.
* El usuario recibe un correo o notificación push con los detalles del boleto.

1. Si el pago falla:

* El sistema libera los asientos.
* El usuario recibe un mensaje con instrucciones para intentar nuevamente.
* Flujo Alternativos:

Si el usuario no completa la compra en el tiempo límite, los asientos reservados se liberan automáticamente.

* Postcondiciones:
* Los asientos seleccionados quedan vendidos.
* El usuario recibe la confirmación de la compra.

### 3. **Atender Incidentes**

* Actores: Usuario, Soporte, Administrador, Sistema de Notificaciones
* Precondiciones:
* El usuario debe haber registrado un incidente en el sistema.
* El soporte debe estar disponible para gestionar los casos.
* Flujo de Eventos Principal:

El usuario contacta al soporte y reporta un incidente.

* + - 1. El equipo de soporte registra el incidente en el sistema.
      2. El soporte evalúa el problema y realiza acciones correctivas si están dentro de su alcance
      3. Si el problema no puede resolverse:
* El incidente se escala al administrador.
* El administrador aplica soluciones específicas (ej. reembolso o reactivación de acceso).
  + - 1. El sistema utiliza el patrón Observer para notificar automáticamente al usuario sobre el estado del incidente.
* Flujo Alternativo:
* Si el soporte resuelve el incidente sin necesidad de escalarlo, el sistema actualiza directamente el estado del caso y notifica al usuario.
* Postcondiciones:
* El usuario recibe una notificación con el estado del incidente (resuelto o en proceso).
* El incidente queda cerrado en el sistema si se resolvió satisfactoriamente.

# **Sección C: Diagrama de Clases con Patrones de Diseño**

## **Diagrama de Clases**

**LINK:**

# **Sección D: Diagramas de Secuencia**

**Link:** <https://online.visual-paradigm.com/share/book/tarea-1-envivotickets---diagramas-de-secuencia--1yfmfh1l9s>

## **Diagrama de Secuencias de Caso de Uso 1. Adquirir Boletos**

## **Diagrama de Secuencias de Caso de Uso 2. Visualizar Disponibilidad de Asientos**

## **Diagrama de Secuencias de Caso de Uso 3. Configurar Precios y Políticas**

## **Diagrama de Secuencias de Caso de Uso 4. Atender Incidentes de Cliente**

# **Sección E: Generación de Código en Java**

**LINK:** [**https://github.com/ChristianMacias0/Tarea01-Sistema\_EnVivoTickets/tree/main**](https://github.com/ChristianMacias0/Tarea01-Sistema_EnVivoTickets/tree/main)