**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** 3](#_Toc184673030)

[**1. Single Responsibility Principle (SRP):** Cada clase en el sistema tiene una única responsabilidad. Por ejemplo: 3](#_Toc184673031)

[**2. Open-Closed Principle (OCP):** Para hacer que el sistema sea extensible sin modificar las clases existentes, los métodos y clases se diseñan para permitir agregar nuevas funcionalidades. Por ejemplo: 3](#_Toc184673032)

[**3. Liskov Substitution Principle (LSP):** Las subclases deben poder reemplazar a sus clases base sin que el sistema falle. Por ejemplo: 3](#_Toc184673033)

[**4. Interface Segregation Principle (ISP):** En lugar de que una clase implemente una interfaz general y grande, dividimos las interfaces en funciones más específicas. 4](#_Toc184673034)

[**5. Dependency Inversion Principle (DIP):** Las clases de alto nivel (como Sistema\_EnVivoTickets) no deben depender directamente de las clases de bajo nivel (como Pago o Ticket), sino de abstracciones o interfaces que puedan implementarse de diferentes maneras. 4](#_Toc184673035)

[**Sección B: Diagrama de Casos de Uso** 5](#_Toc184673036)

[**Diagrama de casos de Uso** 5](#_Toc184673037)

[**Detalle de los 3 Casos de Uso Principales** 5](#_Toc184673038)

[1. **Adquirir Boletos** 5](#_Toc184673039)

[**2. Visualizar Disponibilidad de Asientos** 6](#_Toc184673040)

[3. **Configurar Precios y Políticas** 6](#_Toc184673041)

[**Sección C: Diagrama de Clases con Patrones de Diseño** 7](#_Toc184673042)

[**Diagrama de Clases** 7](#_Toc184673043)

[**Sección D: Diagramas de Secuencia** 7](#_Toc184673044)

[**Diagrama de Secuencias de Caso de Uso 1. Adquirir Boletos** 7](#_Toc184673045)

[**Diagrama de Secuencias de Caso de Uso 2. Visualizar Disponibilidad de Asientos** 7](#_Toc184673046)

[**Diagrama de Secuencias de Caso de Uso 3. Configurar Precios y Políticas** 7](#_Toc184673047)

[**Diagrama de Secuencias de Caso de Uso 4. Atender Incidentes de Cliente** 8](#_Toc184673048)

[**Sección E: Generación de Código en Java** 8](#_Toc184673049)

# **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:**

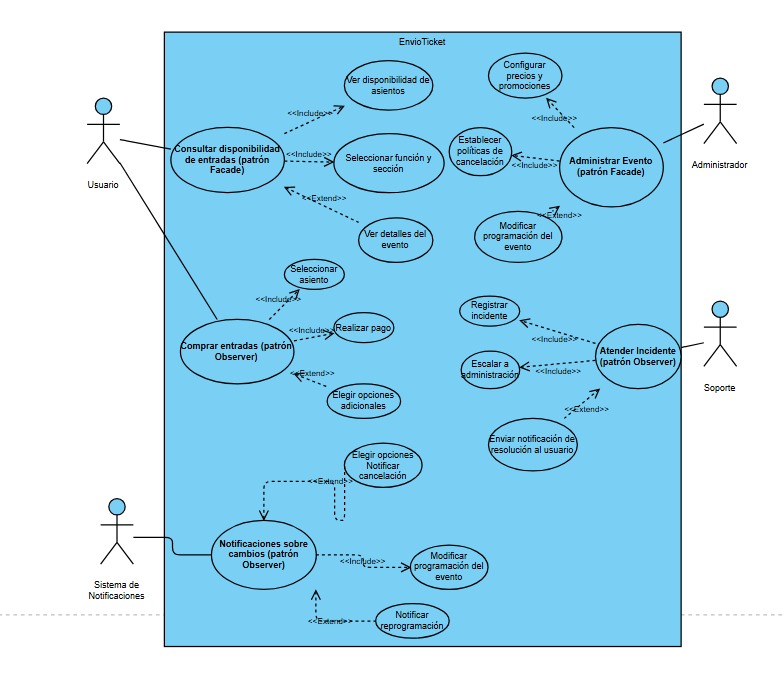
* + La clase SistemaEnVivoTickets actúa como la fachada, exponiendo métodos simples como comprarBoleto() o verDisponibilidad().
  + Internamente, esta clase coordina múltiples subsistemas como Compra, Pago y Notificacion.
  + Los desarrolladores no necesitan interactuar directamente con cada subsistema,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



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

### 1. **Adquirir Boletos**

* Actores: Usuario
* Precondiciones: El usuario debe estar registrado; los asientos seleccionados deben estar disponibles.
* Flujo de Eventos Principal:

1. El usuario selecciona el evento y la función deseada.
2. El sistema muestra la disponibilidad de asientos.
3. El usuario elige su asiento en el mapa interactivo.
4. El sistema retiene temporalmente los asientos.
5. El usuario realiza el pago y confirma la compra.
6. El sistema confirma la transacción y envía el boleto al usuario.

Flujos Alternativos:

* A1: El pago falla -> El sistema ofrece intentarlo nuevamente o cambiar de método de pago.
* A2: Tiempo de espera excedido -> Los boletos se liberan automáticamente.
* Postcondiciones: El boleto queda en estado "reservado"; el usuario recibe la

confirmación.

### **2. Visualizar Disponibilidad de Asientos**

* Actores: Usuario
* Precondiciones: Debe existir una función para el evento con asientos

disponibles.

* Flujo de Eventos Principal:
  + - 1. El usuario selecciona el evento y la función.
      2. El sistema muestra el mapa de asientos con el estado en tiempo real

(disponible, reservado, agotado).

* Flujos Alternativos:
* A1: Si se agotan los boletos durante la visualización, el sistema actualiza

el mapa.

* Postcondiciones: El usuario visualiza la disponibilidad de asientos en tiempo

real.

### 3. **Configurar Precios y Políticas**

* Actores: Administrador
* Precondiciones: El evento debe estar registrado en el sistema.
* Flujo de Eventos Principal:

El administrador accede a la configuración del evento.

* + - 1. El sistema muestra las opciones de precios, promociones y políticas.
      2. El administrador define los precios para cada sección y las promociones.
      3. El administrador establece políticas de cancelación y cambios.
      4. El sistema guarda y publica las configuraciones.
* Flujos Alternativos:
* A1: Si hay restricciones en las promociones, el sistema alerta al

administrador.

* Postcondiciones: Los precios, promociones y políticas quedan actualizados y

visibles para los usuarios.

# **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)