**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](#_Toc184670025)

[**1. Single Responsibility Principle (SRP):** Cada clase en el sistema tiene una única responsabilidad. Por ejemplo: 3](#_Toc184670026)

[**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](#_Toc184670027)

[**3. Liskov Substitution Principle (LSP):** Las subclases deben poder reemplazar a sus clases base sin que el sistema falle. Por ejemplo: 4](#_Toc184670028)

[**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](#_Toc184670029)

[**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](#_Toc184670030)

[**Sección B: Diagrama de Casos de Uso** 5](#_Toc184670031)

[**Diagrama de casos de Uso** 5](#_Toc184670032)

[**Detalle de los 4 Casos de Uso Principales** 5](#_Toc184670033)

[1. **Adquirir Boletos** 5](#_Toc184670034)

[**2. Visualizar Disponibilidad de Asientos** 6](#_Toc184670035)

[3. **Configurar Precios y Políticas** 6](#_Toc184670036)

[**4. Atender Incidentes de Cliente** 7](#_Toc184670037)

[**Sección C: Diagrama de Clases con Patrones de Diseño** 7](#_Toc184670038)

[**Diagrama de Clases** 7](#_Toc184670039)

[**Principios SOLID en el Sistema EnVivoTickets:** 8](#_Toc184670040)

[**Sección D: Diagramas de Secuencia** 8](#_Toc184670041)

[**Diagrama de Secuencias de Caso de Uso 1. Adquirir Boletos** 8](#_Toc184670042)

[**Diagrama de Secuencias de Caso de Uso 2. Visualizar Disponibilidad de Asientos** 9](#_Toc184670043)

[**Diagrama de Secuencias de Caso de Uso 3. Configurar Precios y Políticas** 9](#_Toc184670044)

[**Diagrama de Secuencias de Caso de Uso 4. Atender Incidentes de Cliente** 10](#_Toc184670045)

[**Sección E: Generación de Código en Java** 10](#_Toc184670046)

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

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

La clase User se encarga solo de los datos del usuario y sus métodos relacionados (como iniciar sesión).

La clase Evento gestiona toda la información y detalles de un evento en específico, como su nombre, tipo y descripción, y no se encarga de la disponibilidad de boletos o de la compra.

Compra es responsable únicamente del proceso de compra de boletos y se conecta con Pago para la transacción.

Esto facilita el mantenimiento, ya que si necesitamos cambiar la lógica de compra, solo modificamos Compra sin afectar otras partes del sistema.

### **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:

Evento y Funcion pueden extenderse para nuevos tipos de eventos sin cambiar el código actual, ya que se pueden crear subclases específicas de eventos (como teatro, stand-up, etc.).

Paquete permite incluir más opciones adicionales como estacionamiento o bebidas, manteniendo la estructura actual de la clase sin modificarla directamente.

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

Si creamos diferentes tipos de usuarios (como UsuarioRegular y UsuarioVIP que extienden de User), ambos deben cumplir con las mismas interfaces de usuario (como login y verEventos).

Igualmente, Funcion debe ser intercambiable en contextos donde se espera un Evento, ya que es un tipo de evento específico que hereda de la clase base.

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

Notificacion tiene una interfaz específica de envío de mensajes para distintos tipos de notificaciones, y las clases relacionadas (como User o Administrador) solo implementan lo necesario de esta interfaz.

SoporteTecnico implementa una interfaz enfocada en la gestión de incidencias, pero no es responsable de la compra ni el pago.

### **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.

Sistema\_EnVivoTickets depende de interfaces de Compra, Pago, y Notificacion, permitiendo que se integren diferentes servicios de pago o de notificaciones sin cambiar la lógica principal del sistema.

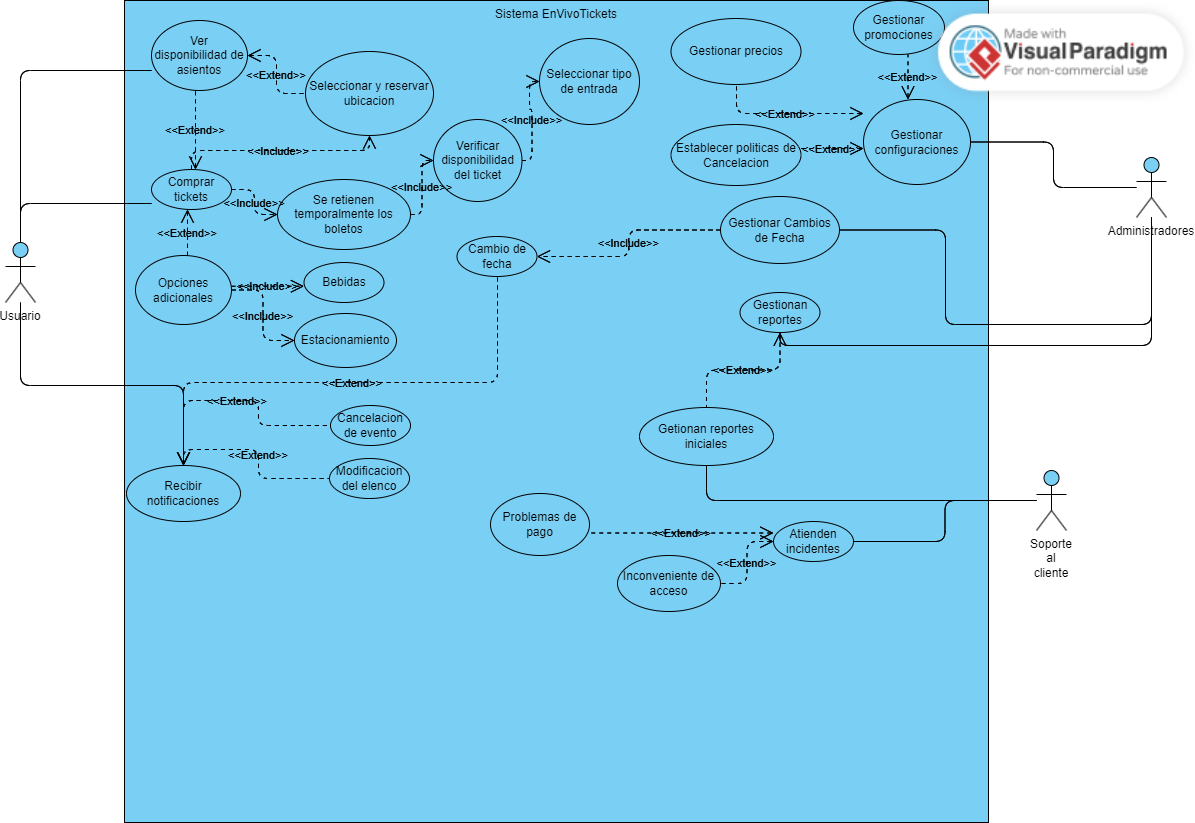
Esto facilita la integración de nuevos métodos de pago, por ejemplo, ya que la implementación de Pago puede cambiar sin afectar al sistema en general.

# 

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

## **Diagrama de casos de Uso**

LINK: https://online.visual-paradigm.com/share/book/tarea-1-ds-1yfmkf9ozs



## **Detalle de los 4 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.

### **4. Atender Incidentes de Cliente**

* Actores: Soporte al Cliente
* Precondiciones: Debe existir un incidente reportado por el usuario.
* Flujo de Eventos Principal:

El soporte recibe el incidente reportado.

El sistema verifica el tipo de problema (pago, acceso, etc.).

El soporte revisa y resuelve el problema o lo escala si es necesario.

El sistema envía una notificación al usuario con la actualización.

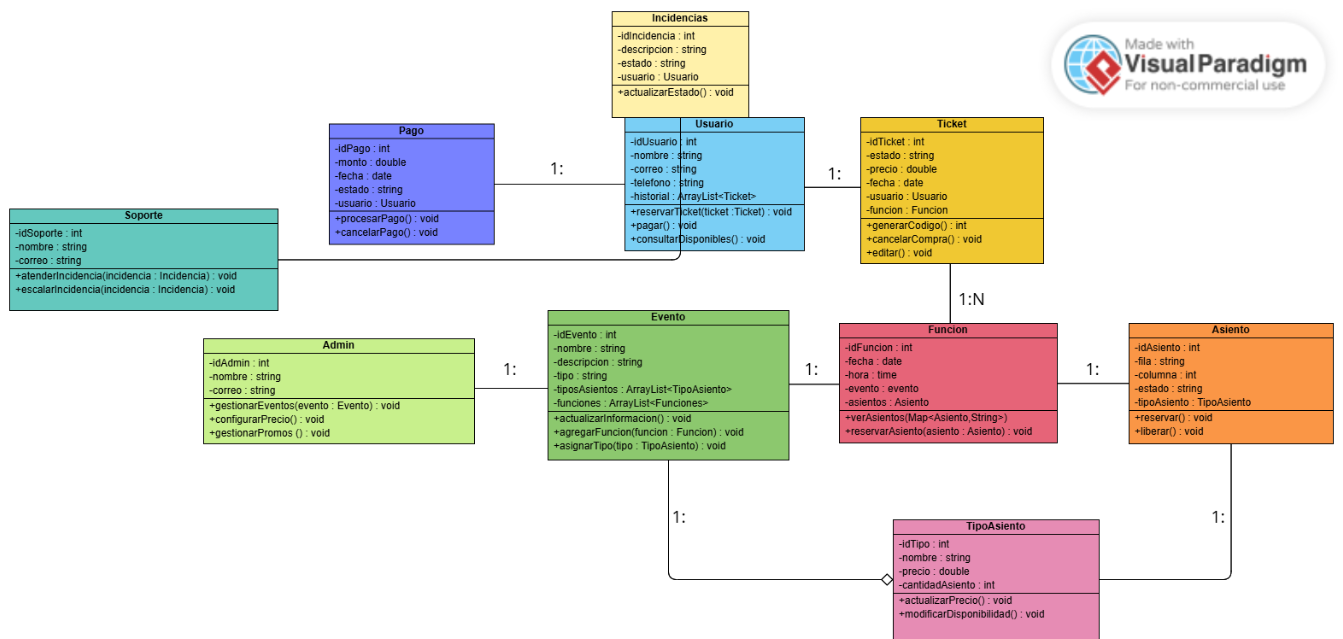
* Flujos Alternativos:
* A1: Si el problema requiere intervención administrativa, se escala.
* Postcondiciones: El incidente queda resuelto o escalado, y el usuario es

Notificado

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

## **Diagrama de Clases**

**LINK:** [**https://online.visual-paradigm.com/share.jsp?id=333639383633352d31**](https://online.visual-paradigm.com/share.jsp?id=333639383633352d31)

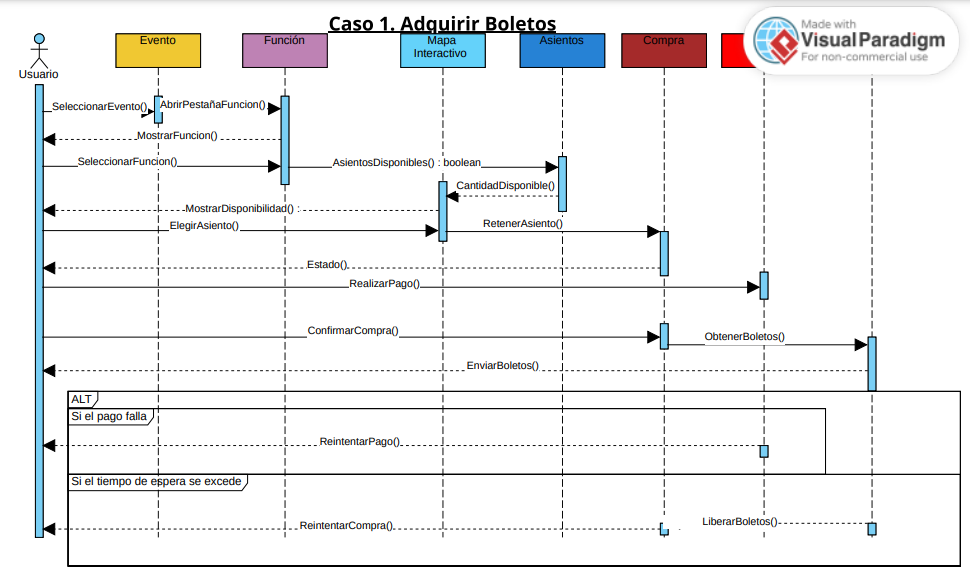
****

## **Principios SOLID en el Sistema EnVivoTickets:**

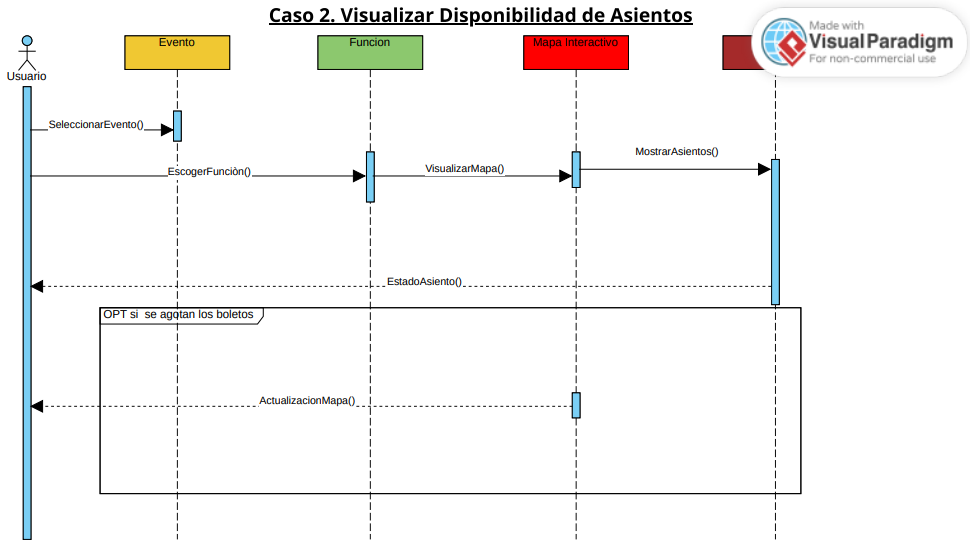
# **Sección D: Diagramas de Secuencia**

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

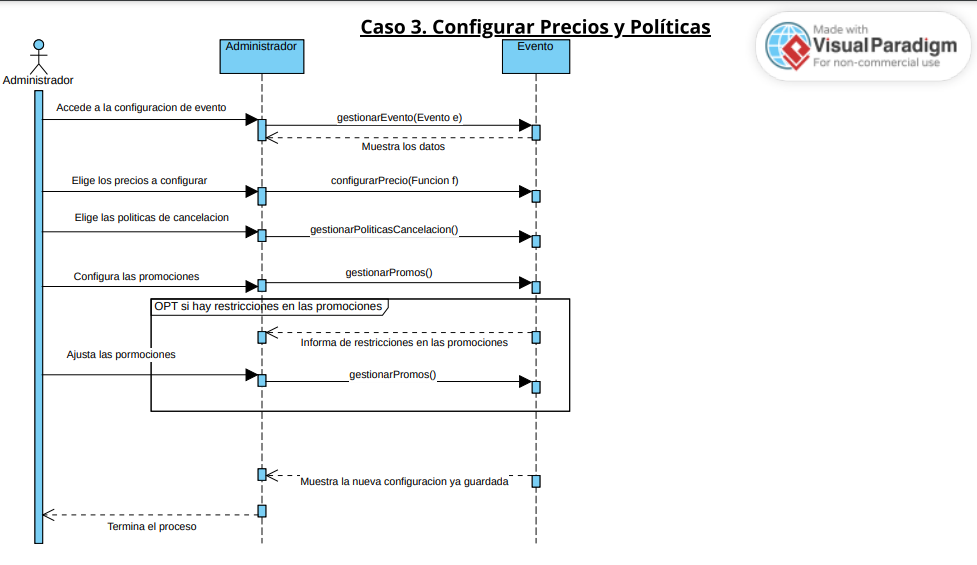
## **Diagrama de Secuencias de Caso de Uso 1. Adquirir Boletos**



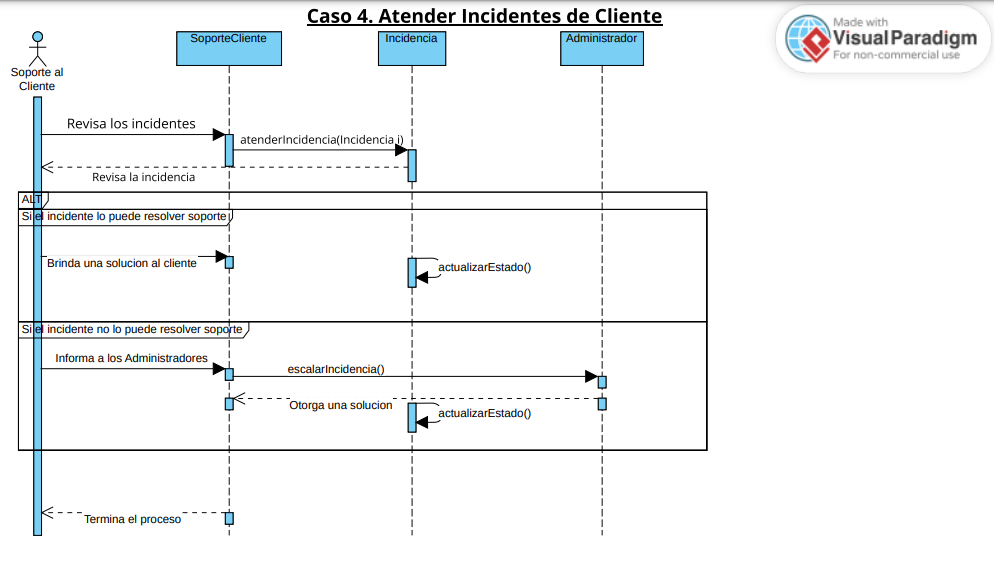
## **Diagrama de Secuencias de Caso de Uso 2. Visualizar Disponibilidad de Asientos**



## **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)