Proyecto 1 – Entrega 2: Diseño e Implementación

Facultad de Ingeniería, Universidad de los Andes

ISIS 1226: Diseño y programación orientada a objetos

Hector Florez

19 de octubre 2025

Sistema: Boletamaster

Integrantes:

- Tomás Fernando Albarracín Malaver - 202421942
- Juan Esteban García Bonillas - 202420481
- Juan Esteban Angel Perdigon - 202420145

---

1. Estado de la Aplicación

Contexto:

Boletamaster es una plataforma digital diseñada para la gestión integral de eventos y venta de tiquetes, orientada a mejorar la experiencia tanto de los usuarios asistentes como de los organizadores.
El sistema permite administrar el ciclo completo de una transaccion: creación de eventos, gestión de localidades y venues, compra y transferencia de tiquetes, aplicación de descuentos y reembolsos, así como la administración de saldos virtuales para los usuarios registrados.

El diseño del sistema busca reflejar procesos reales de plataformas de boleteria, incorporando reglas de negocio claras y restricciones que garanticen la seguridad, integridad de las operaciones y trazabilidad de cada transaccion.
A partir del modelo de dominio construido, se desarrollo una arquitectura modular que separa el modelo logico de datos, los servicios transaccionales y la interacción con servicios externos como las pasarelas de pago.

El diseño planteado representa y mantiene de forma completa todo el estado relevante del sistema Boletamaster, cubriendo los siguientes elementos:

- Usuarios: clases Usuario, Cliente, Organizador y Administrador con relaciones de herencia y roles bien definidos.

- Eventos y Venues: clases Evento y Venue con restricción {unique(venue, fecha)} para asegurar unicidad.

- Tiquetes y Paquetes: TiqueteSimple, TiqueteMultiple y PaqueteDeLuxe con manejo de transferencia, reembolso y vencimiento.

- Transacciones: Compra, Transferencia, Reembolso y Cashout gestionan los movimientos de dinero sobre saldoVirtual.

- Persistencia: los atributos principales de cada entidad son persistibles, asegurando reconstrucción del estado.

Decisiones/Reglas de negocio:

A continuación vamos a presentar la lista de decisiones que tomamos (contando decisiones previas y nuevas):

1. **Uso obligatorio del SaldoVirtual para todas las transacciones:** Todas las compras, reembolsos y transferencias se realizan usando el saldo virtual del usuario. No se permiten transacciones directas con dinero real dentro de la plataforma.
   Impacto UML: El atributo saldoVirtual: double en la clase Usuario adquiere mayor relevancia funcional y se convierte en el origen de todas las operaciones monetarias; además, la clase Transaccion se asocia directamente con Usuario (1:N) para reflejar el uso del saldo virtual.
2. **Recargas (abonos) solo desde cuentas externas:**Los usuarios pueden aumentar su saldo virtual abonando desde un banco o billetera externa mediante la interfaz PasarelaPago.
   Impacto UML: La interfaz PasarelaPago mantiene su rol externo, pero se debe agregar un método abonar_saldo(usuario, monto) y la relación entre Usuario y PasarelaPago se vuelve explícita para representar esta interacción.
3. **Los tiquetes de una misma localidad deben tener precio base uniforme:** Todos los tiquetes pertenecientes a una misma localidad comparten el mismo precio base para evitar inconsistencias.
   Impacto UML: La clase Localidad debe incluir el atributo precioBase y la generación de tiquetes (Tiquete) dependerá de este valor fijo. No se requiere relación nueva, pero se enfatiza la dependencia de Tiquete respecto a Localidad.

4. **Un venue no puede albergar dos eventos el mismo día:** Un mismo escenario no puede tener más de un evento en una fecha determinada.
   Impacto UML: Se mantiene la relación 1:N entre Venue y Evento, pero se debe documentar la restricción de unicidad (venue-fecha) como una regla de integridad en el diseño lógico.
5. **El administrador debe aprobar ciertos venues antes de usarse:** Los organizadores no pueden asignar un venue sin la aprobación previa del administrador.
   Impacto UML: En la clase Venue se refuerza el atributo aprobado: boolean, y se añade un método aprobarVenue() en Administrador. La relación Administrador–Venue se hace explícita para reflejar esa acción.
6. **Todo reembolso se acredita al saldo virtual, nunca en efectivo o tarjeta:** Los reembolsos se devuelven siempre dentro de la plataforma al saldo virtual del usuario.
   Impacto UML: La clase Reembolso debe incluir una relación directa con Usuario para actualizar su saldoVirtual. El atributo origen puede reflejar el tipo de transacción reembolsada.
7. **El límite de tiquetes por transacción se calcula según tipo de compra:** En compras múltiples o de temporada, el tope se calcula por paquetes completos.
   Impacto UML: El atributo maximoTiquetesPorTransaccion: int en Transaccion se mantiene, pero ahora depende del tipo de tiquete (TiqueteSimple, TiqueteMultiple, etc.), lo cual refuerza la necesidad del atributo tipo en las subclases.
8. **Los paquetes Deluxe no son transferibles bajo ninguna condición:** Este tipo de tiquete es personal e intransferible.
   Impacto UML: En la clase PaqueteDeLuxe se puede incluir un método transferir() sobrescrito que lanza una excepción o retorna error, y no se establece relación de transferencia hacia otros usuarios.
9. **No es posible transferir un paquete múltiple si algún tiquete ya está vencido o transferido:** Se bloquea la transferencia del paquete completo si alguno de sus tiquetes asociados ya cambió de estado.
   Impacto UML: La clase TiqueteMultiple debe tener un método validarTransferencia() y una relación interna con una lista de TiqueteSimple o atributos que reflejen los estados individuales.
10. **El organizador solo ve reportes con ingresos sin recargos:** Los informes generados para organizadores excluyen tarifas de servicio y costos de emisión, que pertenecen a la plataforma.
    Impacto UML: En la clase Organizador se debe incluir el método generarReporte(evento) y aclarar que los datos provienen de Evento y Localidad, sin acceder a la Administracion.
11. **Autenticación obligatoria para acciones críticas:** Operaciones como compra, transferencia o retiro requieren validación mediante login y contraseña.
    Impacto UML: Se mantiene el método login() en Usuario, y se añade una referencia a autenticar() que debe invocarse desde Cliente, Administrador o Organizador antes de ejecutar acciones de riesgo.
12. **Implementación de función de "cashout" o retiro a banco:** El usuario puede retirar dinero de su saldo virtual hacia su cuenta bancaria registrada.

Impacto UML: Se amplía la interfaz PasarelaPago con el método transferir_a_banco(usuario, cuenta, monto) y se crea una subclase TransaccionCashout que hereda de Transaccion.

13. **Registro de todas las operaciones como transacciones:** Cada compra, reembolso, transferencia o retiro se registra como una Transaccion.
    Impacto UML: Se crean subclases de Transaccion (TransaccionCompra, TransaccionTransferencia, TransaccionReembolso, TransaccionCashout) y la relación 1:N con Usuario se hace obligatoria.
14. **Validación previa de cuentas bancarias para retiros (cashout):** Las cuentas destino deben estar verificadas antes de permitir retiros.
    Impacto UML: En Usuario se agrega un atributo cuentaVerificada: boolean y posiblemente una clase CuentaBancaria asociada (1:1) para guardar los datos verificados.
15. **Límite dinámico de tiquetes por transacción:** El sistema ajusta el límite según el tipo de compra (por ejemplo, 10 simples o 2 paquetes múltiples).
    Impacto UML: Este comportamiento puede reflejarse mediante un método validarLimite() en Transaccion, y un atributo tipoTransaccion en cada instancia.
16. **Reportes de ingresos netos para organizadores:** Los reportes de ingresos excluyen recargos de servicio o costos fijos de la plataforma.
    Impacto UML: Se complementa el método generarReporte() de Organizador y se refuerza su asociación con Evento, sin conexión directa con Administracion.
17. **Separación de capas del sistema (Modelo / Servicio / Vista):** La arquitectura se divide en tres capas para modularidad y mantenibilidad.
    Impacto UML: Aunque no se refleja directamente en el diagrama de clases del dominio, sí debe representarse en el diseño general del sistema (por ejemplo, mostrando paquetes o módulos: modelo, servicios, vista).
18. **Política de reversión segura ante errores:** Si una transacción falla (por ejemplo, saldo insuficiente u oferta expirada), el sistema revierte automáticamente los cambios realizados.
    Impacto UML: No implica nuevas clases, pero se documenta que los métodos de Transaccion y PasarelaPago deben manejar excepciones específicas (SaldoInsuficienteException, OfertaExpiradaException, etc.) y realizar rollbacks en su implementación.
19. **Cada transacción tiene un ID único:** Cada transacción tiene un ID único para búsqueda y acceso fácil.

---

2. Soporte a Funcionalidades

El diseño contempla todas las funcionalidades del sistema, organizadas por tipo de usuario:

- Cliente: compra, transferencia y reembolso de tiquetes.
- Organizador: creación y administración de eventos, generación de reportes.
- Administrador: aprobación de venues y supervisión de transacciones.

El sistema sigue una arquitectura modular con tres capas:

1. Modelo (Dominio): entidades y reglas de negocio.
2. Servicios: transacciones, pagos, reportes.
3. Vista/Control: interacción de usuarios.

---

3. Documentación de la Persistencia

La persistencia del sistema se implementa mediante archivos JSON ubicados en la carpeta data/, garantizando que la información del sistema se conserve entre ejecuciones. El archivo principal db.json almacena todos los objetos del dominio (usuarios, eventos, localidades, tiquetes y venues) en formato estructurado y legible.

La clase JsonStore se encarga de guardar y cargar los datos.

- El método save(Database db) convierte los objetos en texto JSON y los escribe en data/db.json.
- El método load() lee ese archivo y reconstruye los objetos originales en memoria.

La clase Database actúa como contenedor de la información, manteniendo listas de usuarios, eventos y venues, que luego son serializadas o cargadas por JsonStore.

El proceso fue validado mediante el test PersistenciaJsonTest, que crea una base de datos en memoria, la guarda, la vuelve a cargar y confirma que los datos se conservan sin pérdida.

El formato JSON fue elegido por su claridad, portabilidad y facilidad de manejo en Java, cumpliendo con el requisito de que los datos sean persistentes fuera del código fuente del proyecto.

---

4. Diagrama UML de Alto Nivel

El diagrama de clases de alto nivel permite entender la estructura general del sistema, mostrando:

- Relaciones de herencia y dependencias entre Usuario, Transaccion, Evento y Venue.

- Interfaz externa «interface» PasarelaPago.

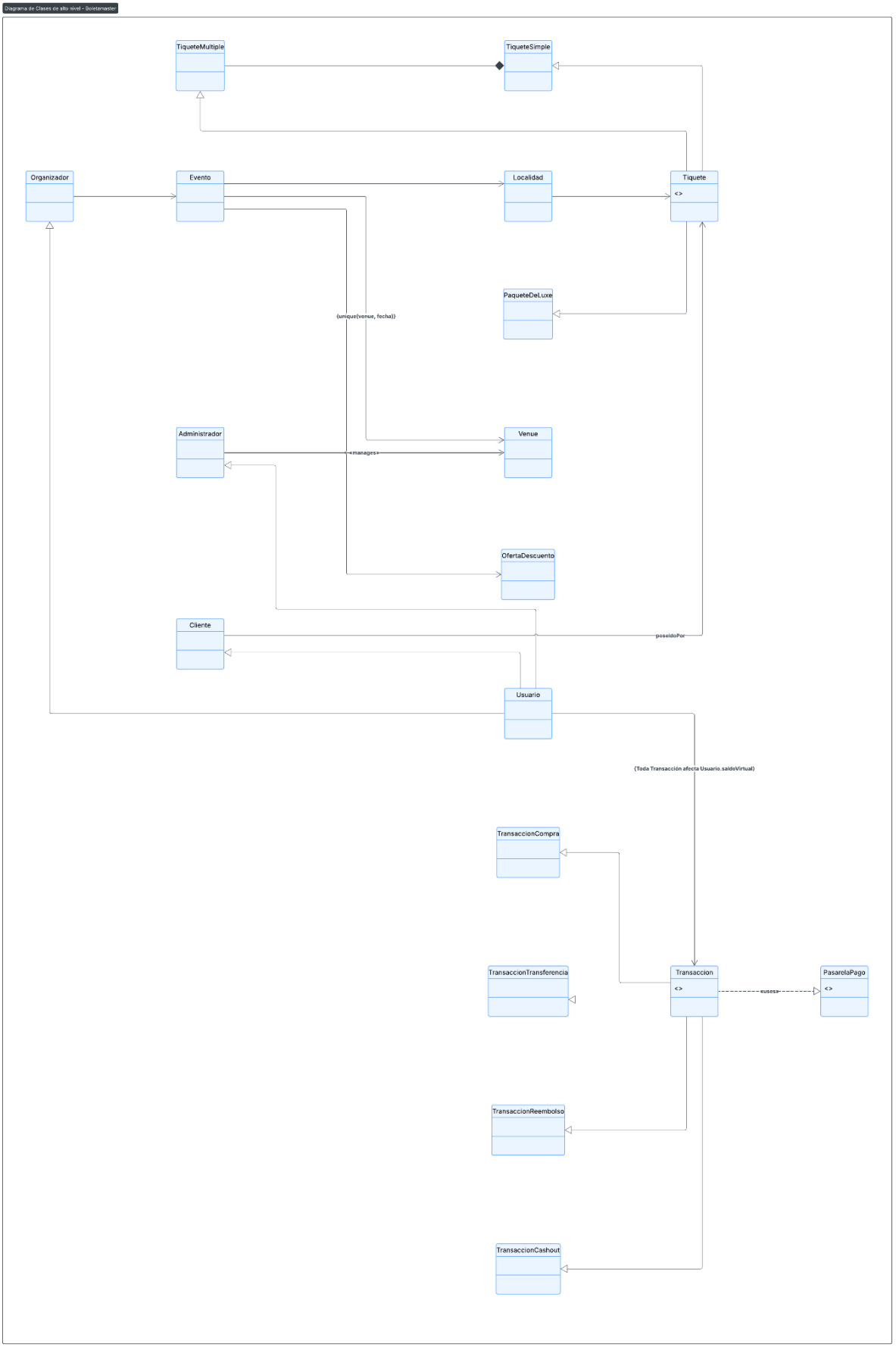


Diagrama de Clases de alto nivel - Boletamaster
TiqueteMultiple
TiqueteSimple
Organizador
Evento
Localidad
Tiquete
<>
PaqueteDeLuxe
{unique(venue, fecha)}
Administrador
«manages»
Venue
OfertaDescuento
Cliente
poseídoPor
Usuario
{Toda Transacción afecta Usuario.saldoVirtual}
TransaccionCompra
TransaccionTransferencia
Transaccion
<>
«usesa»
PasarelaPago
<>
TransaccionReembolso
TransaccionCashout

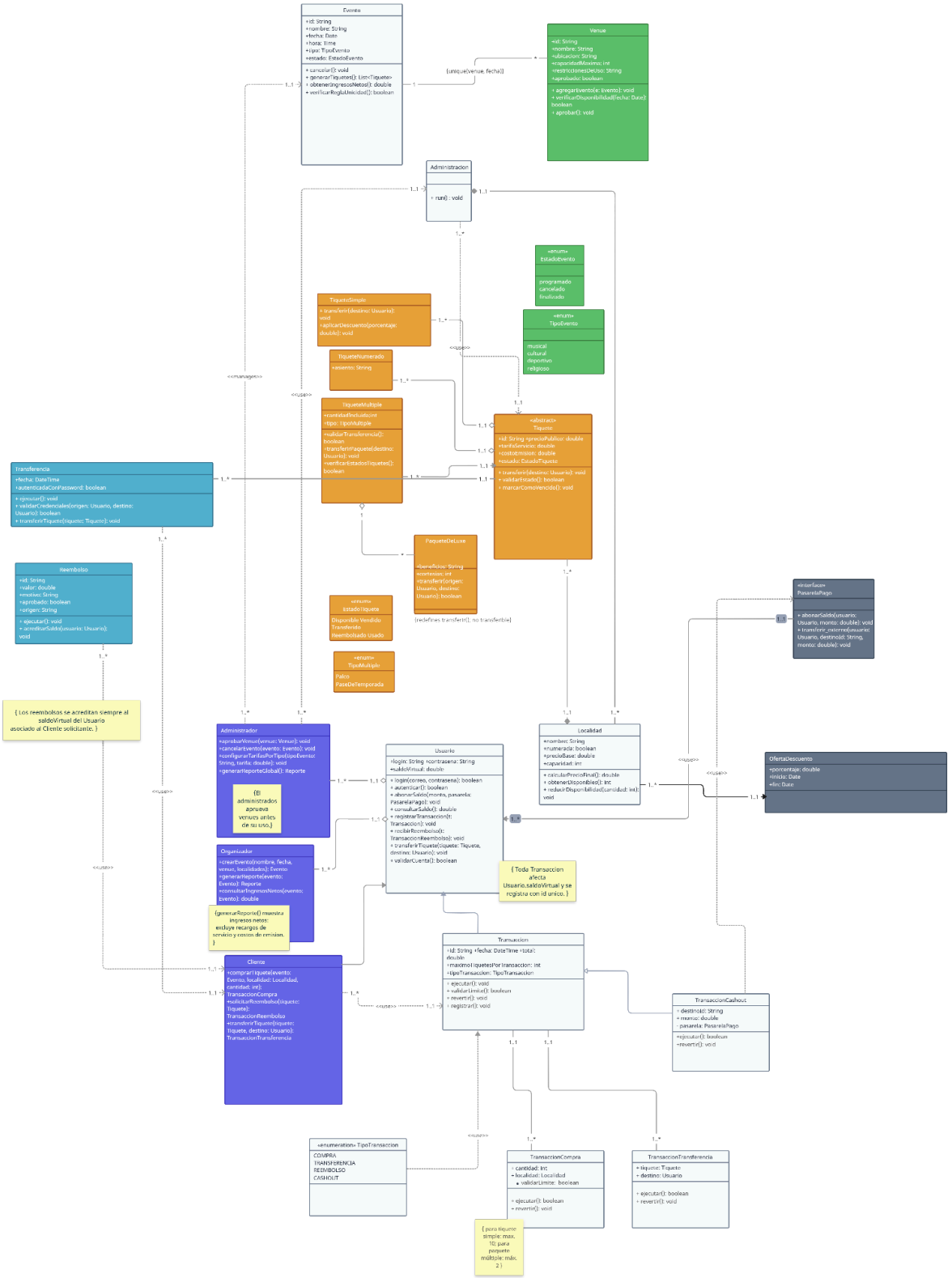
---

5. Diagrama UML Detallado

Incluye todas las clases, atributos, métodos y multiplicidades.

Se evidencian las reglas de negocio mediante notas OCL y estereotipos UML («uses», «manages», «enumeration», «interface»).

Cumple las 20 decisiones de diseño del dominio Boleta Master.


Evento
+id: String
+nombre: String
+fecha: Date
+hora: Time
+tipo: TipoEvento
+estado: EstadoEvento
+ cancelar(): void
+ generarTiquetes(): List<Tiquete>
+ obtenerIngresosNetos(): double
+ verificarReglaUnicidad(): boolean
Venue
+id: String
+nombre: String
+ubicacion: String
+capacidadMaxima: int
+restriccionesDeUso: String
+aprobado: boolean
+ agregarEvento(e: evento): void
+ verificarDisponibilidad(fecha: Date): boolean
+ aprobar(): void
{unique(Venue, fecha)}
Administracion
+ run() : void
«enum»
EstadoEvento
programado
cancelado
finalizado
«enum»
TipoEvento
musical
cultural
deportivo
religioso
TiqueteSimple
+ transferir(destino: Usuario): void
+aplicarDescuento(porcentaje: double): void
TiqueteNumerado
+asiento: String
TiqueteMultiple
+cantidadEnIncluidos:int
+tipo: TipoMultiple
+validarTransferencia(): boolean
+transferirPaquete(destino: Usuario): void
-verificarEstadosTiquetes(): boolean
«abstract»
Tiquete
+id: String +precioPublico: double
+tarifaServicio: double
+costoEmision: double
+estado: EstadoTiquete
+ transferir(destino: Usuario): void
+ validarEstado(): boolean
+ marcarComoVencido(): void
PaqueteDeLuxe
+beneficios: String
+comision: int
+transferir(origen: Usuario, destino: Usuario): boolean
{redefines transferir(); no transferible}
«enum»
EstadoTiquete
Disponible Vendido
Transferido
Reembolsado Usado
«enum»
TipoMultiple
Palco
PaseDeTemporada
Transferencia
+Fecha: DateTime
+autenticadaConPassword: boolean
+ ejecutar(): void
+ validarCredenciales(origen: Usuario, destino: Usuario): boolean
+ transferirTiquete(tiquete: Tiquete): void
Reembolso
+id: String
+valor: double
+motivo: String
+aprobado: boolean
+origen: String
+ ejecutar(): void
+ acreditarSaldo(usuario: Usuario): void
{ Los reembolsos se acreditan siempre al saldoVirtual del Usuario asociado al Cliente solicitante. }
«interface»
PasarelaPago
+ abonarSaldo(usuario: Usuario, monto: double): void
+ transferir_reembolso(usuario: Usuario, destinoId: String, monto: double): void
Administrador
+aprobarVenue(venue: Venue): void
+cancelarEvento(evento: Evento): void
+configurarTarifaPorTipoEvento(: String, tarifa: double): void
+generarReporteGlobal(): Reporte
{El administrador aprueva venues antes de su uso.}
Usuario
+login: String +contrasena: String
+saldoVirtual: double
+ login(correo, contrasena): boolean
+ autenticar(): boolean
+ abonarSaldo(monto, pasarela: PasarelaPago): void
+ consultarSaldo(): double
+ registrarTransaccion(t: Transaccion): void
+ recibirReembolso(t: TransaccionReembolso): void
+ transferirTiquete(tiquete: Tiquete, destino: Usuario): void
+ validarCuenta(): boolean
Localidad
+nombre: String
+numerada: boolean
+precioBase: double
+capacidad: int
+ calcularPrecioFinal(): double
+ obtenerDisponibles(): int
+ reducirDisponibilidad(cantidad: int): void
OfertaDescuento
+porcentaje: double
+inicio: Date
+fin: Date
Organizador
+crearEvento(nombre, fecha, venue, localidades): Evento
+generarReporte(evento: Evento): Reporte
+consultarIngresosNetos(evento: Evento): double
{generarReporte() muestra ingresos netos: excluye recargos de servicio y costos de emision. }
{ Toda Transaccion afecta Usuario.saldoVirtual y se registra con id unico. }
Cliente
+comprarTiquete(evento: Evento, localidad: Localidad, cantidad: int): TransaccionCompra
+solicitarReembolso(tiquete: Tiquete): TransaccionReembolso
+transferirTiquete(tiquete: Tiquete, destino: Usuario): TransaccionTransferencia
Transaccion
-id: String +Fecha: DateTime +total: double
+maximoTiquetesPorTransaccion: int
-tipoTransaccion: TipoTransaccion
+ ejecutar(): void
+ validarLimite(): boolean
+ revertir(): void
+ registrar(): void
TransaccionCashout
+ destinoId: String
+ monto: double
- pasarela: PasarelaPago
+ejecutar(): boolean
+revertir(): void
«enumeration» TipoTransaccion
COMPRA
TRANSFERENCIA
REEMBOLSO
CASHOUT
TransaccionCompra
+ cantidad: Int
+ localidad: Localidad
+ validarLimite: boolean
+ ejecutar(): boolean
+ revertir(): void
TransaccionTransferencia
+ tiquete: Tiquete
+ destino: Usuario
+ ejecutar(): boolean
+ revertir(): void
{ para tiquete simple: max. 10 para paquete múltiple: máx. 2 }

6. Asignación de Responsabilidades

1. USUARIOS Y ROLES

Usuario (superclase)
Responsabilidades: Gestionar autenticacion y sesion, mantener y exponer el saldo virtual, ser dueño de todas sus transacciones, recibir reembolsos.
Colabora con: Transacciones (Compra, Transferencia, Reembolso, Cashout), Pasarela de Pago.

Cliente (subclase de Usuario)
Responsabilidades: Comprar tiquetes, solicitar reembolsos, transferir tiquetes, ejecutar cashout.
Colabora con: Evento, Localidad, Tiquete, Transaccion, PasarelaPago.

Organizador (subclase de Usuario)
Responsabilidades: Crear eventos, definir localidades, consultar ingresos netos, generar reportes.
Colabora con: Evento, Localidad, Administrador.

Administrador (subclase de Usuario)
Responsabilidades: Aprobar venues, cancelar eventos, configurar tarifas globales, supervisar persistencia.
Colabora con: Venue, Evento.

2. DOMINIO DE EVENTOS

Evento
Responsabilidades: Representar un evento con fecha, venue y localidades, generar tiquetes, calcular ingresos netos, verificar unicidad (venue, fecha).
Colabora con: Venue, Localidad, Tiquete, OfertaDescuento.

Venue
Responsabilidades: Representar el lugar fisico, validar disponibilidad, permitir asignacion de eventos aprobados.
Colabora con: Administrador, Evento.

Localidad
Responsabilidades: Representar una zona con precio base uniforme, administrar disponibilidad, calcular precio final con descuentos.
Colabora con: Evento, Tiquete, OfertaDescuento.

OfertaDescuento
Responsabilidades: Definir vigencia y porcentaje de descuento, indicar si aplica a una fecha dada.
Colabora con: Evento, Localidad.

3. TIQUETES

Tiquete (abstracta)
Responsabilidades: Representar un derecho de acceso con estado, transferir titularidad, validar estado.
Colabora con: Usuario (titular), TransaccionTransferencia.

TiqueteSimple
Responsabilidades: Instancia basica de tiquete.
Colabora con: Tiquete (superclase).

TiqueteMultiple
Responsabilidades: Agrupar multiples tiquetes simples, validar transferencia del paquete, ejecutar transferencia de paquete.
Colabora con: TiqueteSimple, Usuario.

PaqueteDeLuxe
Responsabilidades: Representar paquete premium no transferible, gestionar beneficios adicionales.
Colabora con: Tiquete (superclase).

4. TRANSACCIONES Y PAGOS

Transaccion (abstracta)
Responsabilidades: Unidad de trabajo para operaciones monetarias, ejecutar, registrar y revertir transacciones.
Colabora con: Usuario, PasarelaPago.

TransaccionCompra
Responsabilidades: Ejecutar la compra de tiquetes, verificar stock y limites, debitar saldo y registrar tiquetes.
Colabora con: Usuario (Cliente), Evento, Localidad, Tiquete.

TransaccionTransferencia
Responsabilidades: Transferir titularidad de un tiquete, validar autenticacion y estado.
Colabora con: Usuario (origen y destino), Tiquete.

TransaccionReembolso
Responsabilidades: Ejecutar devolucion de valor al saldo virtual, cambiar estado del tiquete.
Colabora con: Usuario, Tiquete.

TransaccionCashout
Responsabilidades: Retirar dinero del saldo virtual hacia destino externo mediante pasarela.
Colabora con: Usuario, PasarelaPago.

PasarelaPago (interfaz)
Responsabilidades: Abstraer el proveedor externo de pagos, realizar abonos y transferencias externas.
Colabora con: Usuario, TransaccionCashout.

Clase	Responsabilidad Principal
Usuario	Autenticación y manejo del saldo virtual
Cliente	Compra, transferencia y solicitud de reembolso
Organizador	Creación de eventos y reportes
Administrador	Aprobación de venues y control de eventos
Evento	Gestión de tiquetes y cálculo de ingresos
Transaccion	Registro y ejecución de operaciones
PasarelaPago	Procesamiento de pagos y retiros externos

---

7. Justificaciones y Decisiones Críticas

- Se elimina CuentaBancaria, reemplazada por la interfaz PasarelaPago.
- El atributo saldoVirtual centraliza todas las operaciones financieras.
- Las subclases de Transacción permiten trazabilidad individual.
- PaqueteDeLuxe es intransferible por regla de negocio.
- Se documentan restricciones con notas {} (OCL) en el UML.

---

8. Diagramas de Secuencia

Diagrama de secuencia de compra de tiquete simple

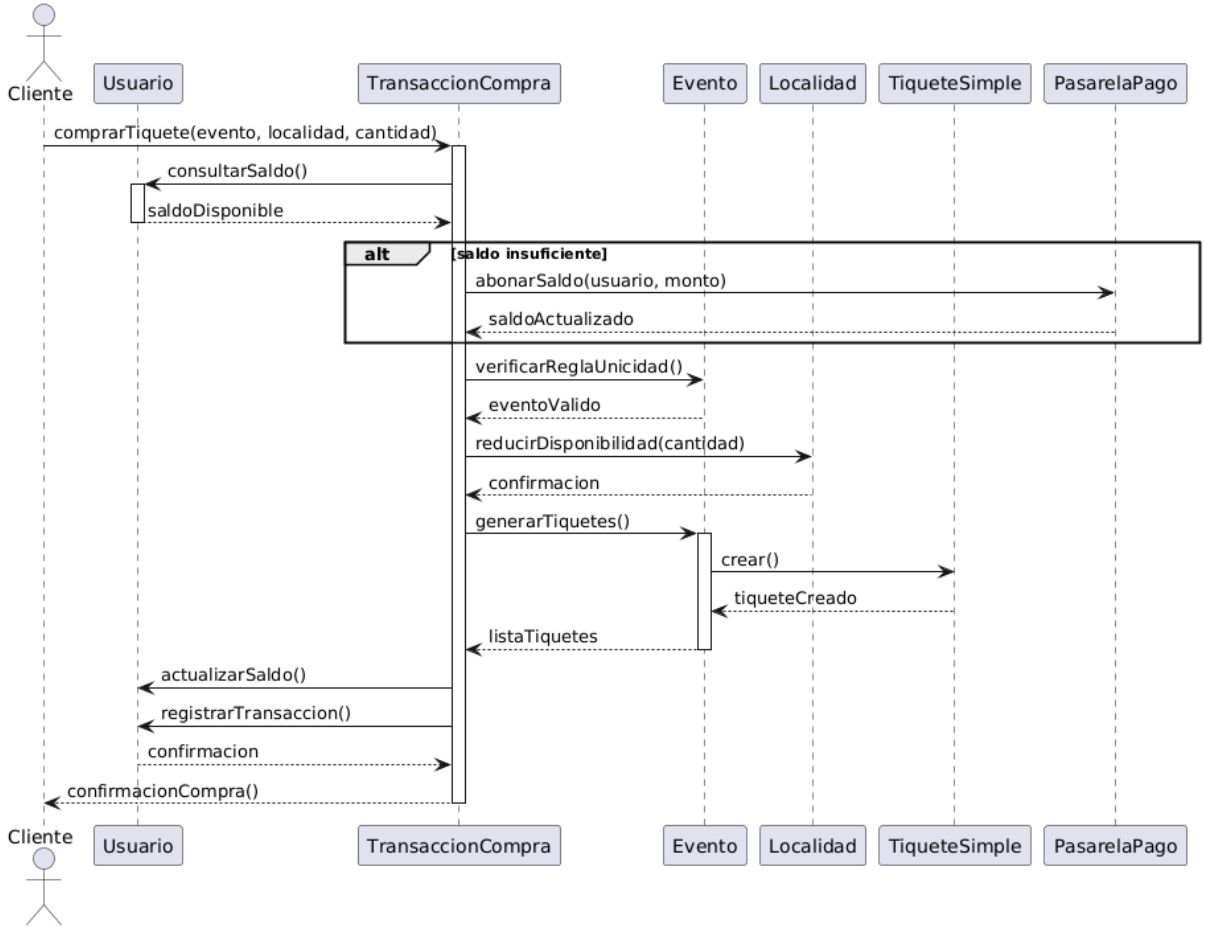


Diagrama de secuencia de creación de evento

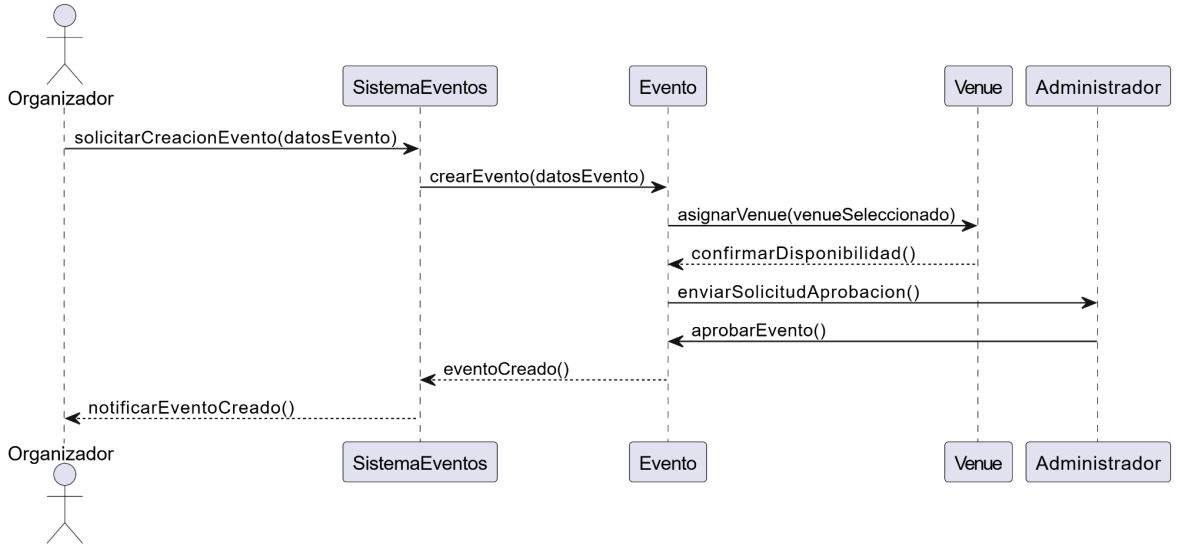


Diagrama de secuencia de cancelación de evento

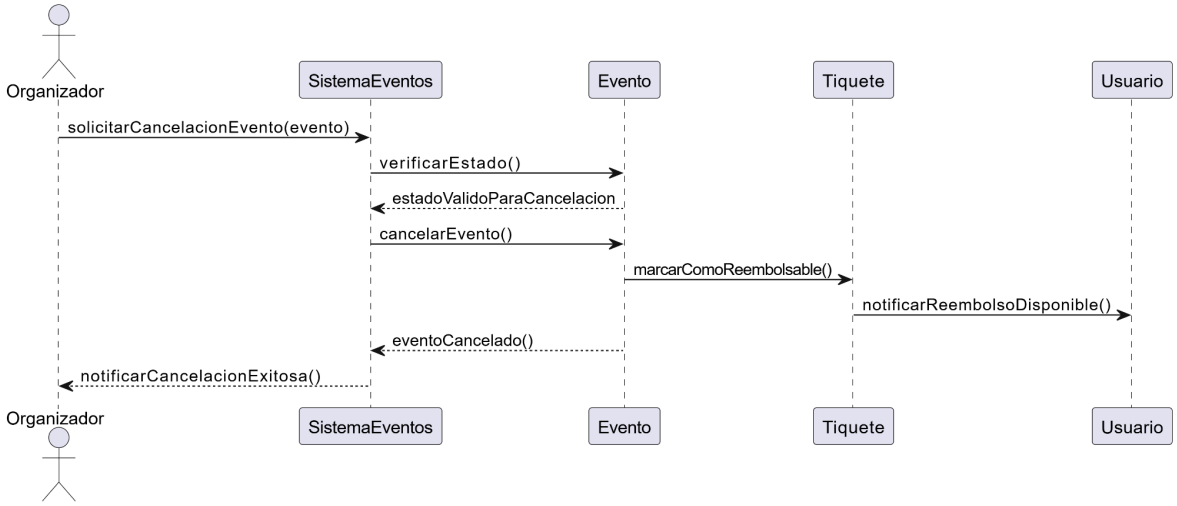
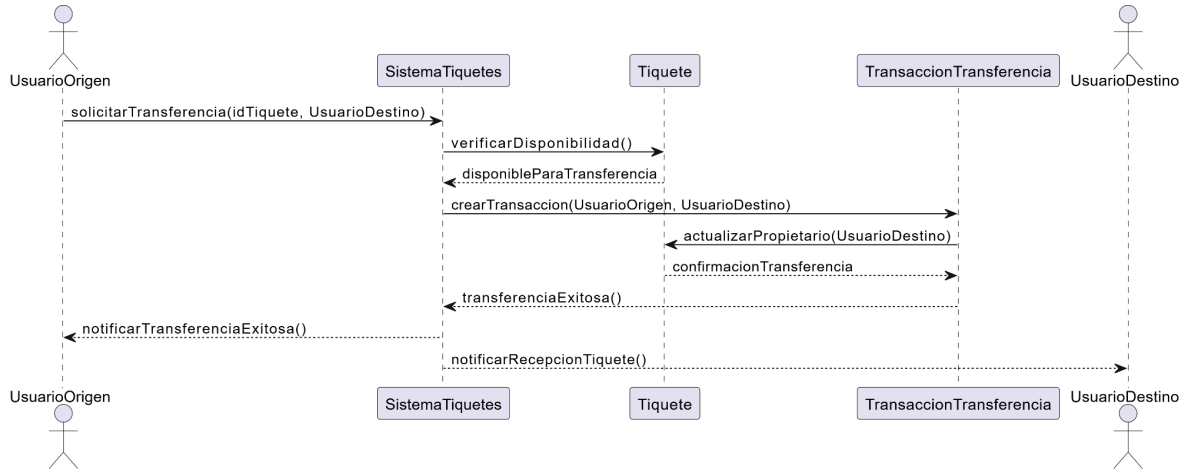
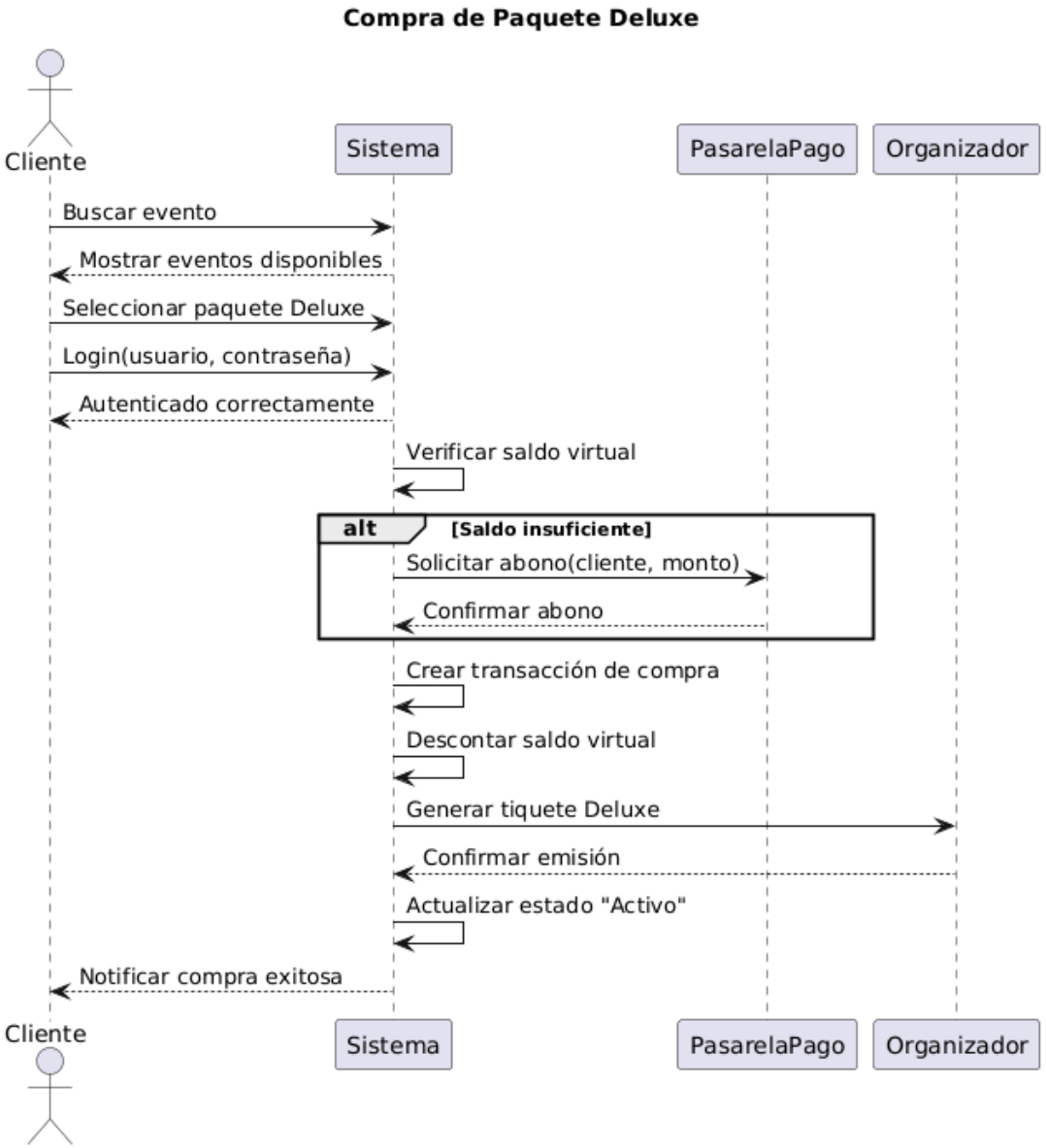
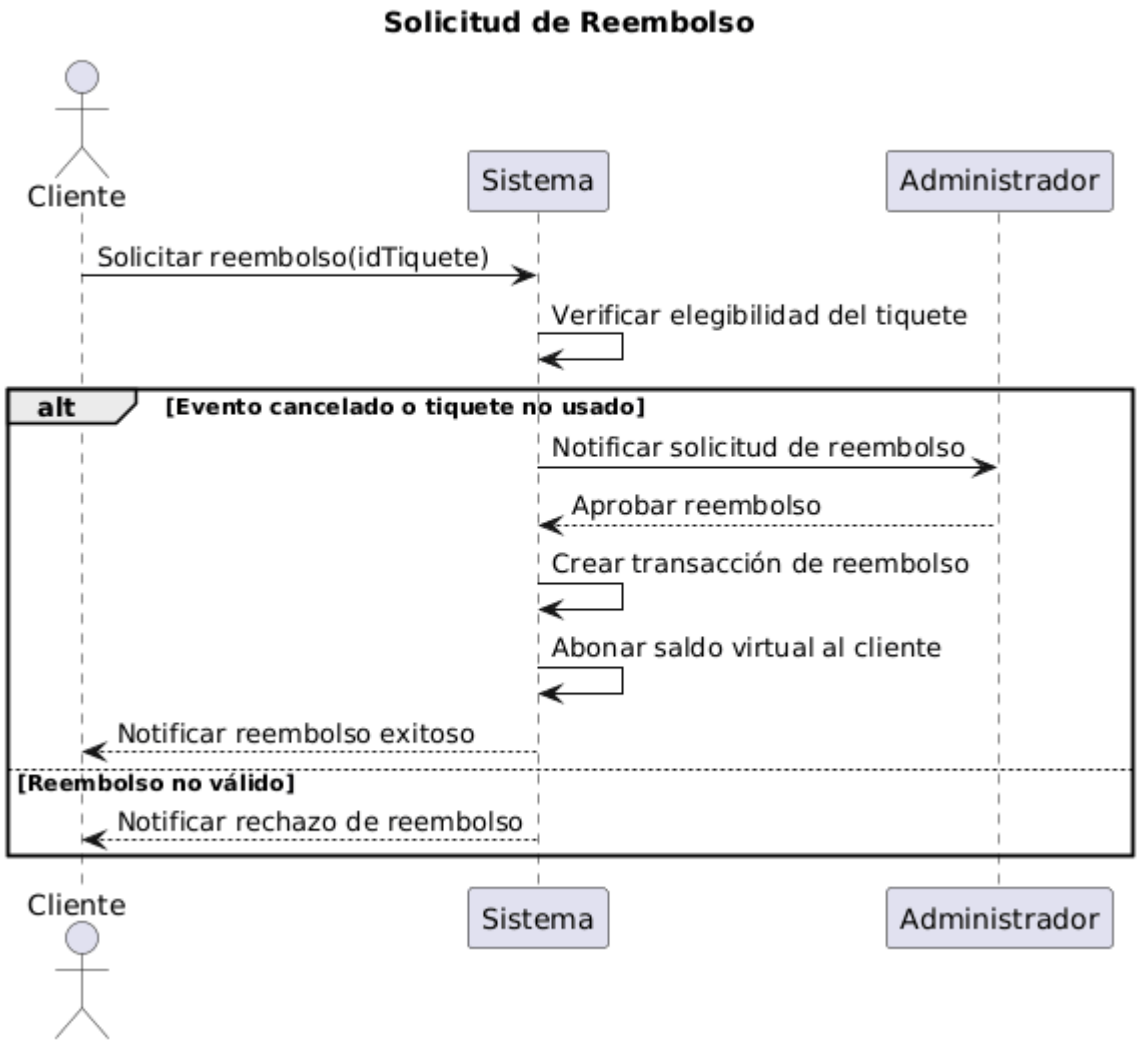


Diagrama de secuencia de transferencia de evento


UsuarioOrigen
SistemaTiquetes
Tiquete
TransaccionTransferencia
UsuarioDestino
solicitarTransferencia(idTiquete, UsuarioDestino)
verificarDisponibilidad()
disponibleParaTransferencia
crearTransaccion(UsuarioOrigen, UsuarioDestino)
actualizarPropietario(UsuarioDestino)
confirmacionTransferencia
transferenciaExitosa()
notificarTransferenciaExitosa()
notificarRecepcionTiquete()
UsuarioOrigen
SistemaTiquetes
Tiquete
TransaccionTransferencia
UsuarioDestino


Compra de Paquete Deluxe
Cliente
Sistema
PasarelaPago
Organizador
Buscar evento
Mostrar eventos disponibles
Seleccionar paquete Deluxe
Login(usuario, contraseña)
Autenticado correctamente
Verificar saldo virtual
alt
[Saldo insuficiente]
Solicitar abono(cliente, monto)
Confirmar abono
Crear transacción de compra
Descontar saldo virtual
Generar tiquete Deluxe
Confirmar emisión
Actualizar estado "Activo"
Notificar compra exitosa
Cliente
Sistema
PasarelaPago
Organizador



---

9. Tests unitarios

Se implementaron tests unitarios con JUnit 5 para validar el correcto funcionamiento del sistema, especialmente la capa de persistencia y las operaciones principales del dominio.

El test PersistenciaJsonTest comprueba que la información del sistema se guarde correctamente en formato JSON y pueda ser leída nuevamente sin pérdida de datos. Evalúa los métodos save() y load() de la clase JsonStore, verificando la integridad de usuarios, eventos y tiquetes antes y después del guardado.

También se incluyeron pruebas para las transacciones (TransaccionCompra y TransaccionReembolso), confirmando que las operaciones de compra, reembolso y actualización de saldo virtual funcionen según las reglas de negocio.

En conjunto, los tests garantizan que las clases centrales del sistema (usuarios, tiquetes, transacciones y persistencia) se comporten de forma coherente, segura y consistente con el modelo definido en el UML.

---

10. Conclusiones

En esta segunda etapa de diseño e implementación, consideramos que el sistema logró una estructura sólida, modular y coherente con el modelo planteado en el UML. Se desarrollaron las clases principales del dominio, las relaciones entre ellas y una capa de persistencia funcional basada en JSON, que garantiza la conservación de los datos entre ejecuciones. Además, se implementaron pruebas unitarias que validan el correcto funcionamiento de las operaciones clave, confirmando que el sistema cumple con los requerimientos funcionales establecidos y está preparado para futuras etapas de integración o despliegue.