Pablo Galvis

Jorge Sebastián Otálora

Isabella Forero

Documento de Diseño

1) Introducción

BoletaMaster es una aplicación de escritorio en Java con interfaz gráfica basada en Swing que permite gestionar la venta de tiquetes para eventos musicales, deportivos, culturales y religiosos.
El sistema modela tres roles principales (Administrador, Organizador y Cliente) que interactúan con las entidades del dominio: Evento, Localidad, Tiquete, Oferta, Pago, Venue y PaqueteTiquetes.

Este documento describe el diseño completo del sistema, incluyendo la estructura de clases, la arquitectura multicapa y los diagramas de clases, secuencia y estado.

2) Estado de la aplicación

El sistema mantiene el estado completo en memoria a través de estructuras estáticas en Main:

static Administrador admin;

static ArrayList<Usuario> usuarios;

static ArrayList<Evento> eventos;

static ArrayList<Localidad> localidades;

static ArrayList<Tiquete> inventario;

Estos representan el estado total de la aplicación.

El objeto DataStore garantiza la persistencia de este estado en archivos CSV (en carpeta data/), conservando usuarios, eventos, localidades, ofertas, tiquetes, paquetes y pagos entre ejecuciones. Además, Main mantiene un objeto estático DataStore DS que centraliza la persistencia del estado completo en archivos CSV.

3) Soporte a funcionalidades

El diseño cubre todas las funcionalidades del enunciado del proyecto:

Rol	Funcionalidades implementadas
Administrador	Configurar tarifas (porcentaje + cuota fija), cancelar eventos, procesar reembolsos.
Organizador	Crear eventos, localidades y ofertas; gestionar precios y aforos.
Cliente	Abonar saldo, comprar tiquetes, transferir tiquetes a otros usuarios.
General	Persistencia completa, cálculo de precios, estados de tiquetes y aforos.

Funciones planificadas para una futura entrega:

- Aprobación de venues propuestos.
- Límite máximo de tiquetes por transacción.
- Reembolsos parciales o por calamidad.

4) Documentación de la persistencia

La persistencia del sistema se maneja con archivos CSV dentro de la carpeta data/, sin usar bases de datos.
Toda la lectura y escritura se controla desde la clase DataStore, que utiliza el utilitario Csv para leer y guardar los datos de manera estructurada.
Así se asegura que la información de usuarios, eventos y tiquetes se conserve entre ejecuciones.

Al iniciar el programa, Main llama a DataStore.loadAll(). Este método carga los archivos y reconstruye todos los objetos (Administrador, Clientes, Organizador, Eventos, Localidades, Tiquetes, Ofertas, etc.), vinculándolos entre sí a través de sus IDs.
Cuando el usuario realiza una acción que cambia el estado del sistema (como crear eventos, comprar tiquetes o configurar tarifas), se llama a DataStore.saveAll() para guardar nuevamente todos los datos en los CSV.

Cada tipo de entidad tiene su archivo:

- administradores.csv, clientes.csv, organizadores.csv -> información de usuarios.
- eventos.csv, localidades.csv, ofertas.csv, venues.csv -> estructura de los eventos.
- tiquetes.csv -> inventario de tiquetes (con su estado y propietario).

- pagos.csv y paquete_items.csv están definidos para mantener la trazabilidad de pagos y paquetes.

El diseño de persistencia mantiene bajo acoplamiento con el resto del sistema:
Main y las clases del dominio nunca manejan directamente archivos, sino que delegan todo a DataStore.
Esto facilita reemplazar la persistencia (por ejemplo, cambiar CSV a base de datos) sin modificar las reglas del dominio.

Una pequeña limitación actual es que los archivos de pagos se crean vacíos si no se mantiene una lista central de pagos, pero DataStore ya está preparado para leerlos en futuras versiones.

5. Arquitectura del sistema

La arquitectura del sistema sigue un enfoque multicapa. En el Proyecto 3, la aplicación incorpora una **capa de presentación gráfica**, reemplazando la consola usada en versiones anteriores. La nueva arquitectura queda organizada así:

Capa de Presentación (GUI – Swing)

Incluye las clases encargadas de la interacción con el usuario:

- VentanaLogin
- VentanaPrincipal
- VentanaCliente
- VentanaOrganizador
- VentanaAdmin
- DialogoImprimirTiquete
- UIUtils
- QRUtil

Responsabilidades:

- Mostrar componentes visuales (formularios, listas, botones).
- Recibir acciones del usuario y comunicarlas al dominio.

- Mostrar mensajes de error y confirmación.
- Renderizar la vista previa de tiquetes y generar códigos QR.

Capa de Dominio

Incluye las clases con la lógica del negocio:

Usuario, Cliente, Organizador, Administrador,
Evento, Localidad, Oferta, Venue,
Tiquete, TiqueteSimple, TiqueteNumerado,
PaqueteTiquetes, PaqueteDeluxe, Pago.

Responsabilidades:

- Modelar las reglas del negocio de la plataforma.
- Validar transacciones como compras, transferencias, cancelaciones, aforo, precios y reembolsos.
- Mantener estados válidos (tiquete impreso, tiquete transferido, evento cancelado).

Capa de Persistencia

Clases:

- DataStore
- Csv

Responsabilidades:

- Leer y escribir archivos CSV.
- Reconstruir objetos a partir de los datos persistidos.
- Guardar modificaciones cuando se compran, transfieren o imprimen tiquetes.

Relación entre capas

- La GUI **usa** el dominio.
- El dominio **delegue** la persistencia en DataStore.
- DataStore utiliza Csv como utilitario de lectura/escritura.

- El Main original se conserva solo como punto de entrada alternativo, pero la interacción principal se realiza vía GUI.

6) Contexto y alcance

Contexto:
Los organizadores crean eventos, los clientes compran y transfieren tiquetes, y el administrador fija tarifas y controla cancelaciones.

Alcance de la entrega:

- Crear eventos, localidades y ofertas.
- Publicar y vender tiquetes simples o numerados.
- Transferir tiquetes y procesar reembolsos.

Fuera de alcance:

- Pasarelas de pago externas.
- Persistencia en bases de datos.

7) Objetivos del diseño

- Implementar la lógica del dominio descrita en el análisis.
- Aplicar principios de diseño orientado a objetos: encapsulamiento, herencia, cohesión y bajo acoplamiento.
- Mantener una estructura modular y extensible
- Asegurar correspondencia directa entre los diagramas y las clases Java.

8) Requerimientos

Requerimientos funcionales

1. **Autenticación de usuarios**
    - Iniciar sesión como Cliente, Organizador o Administrador.
    - Validar contraseña y rol.

2. **Gestión de eventos (Organizador)**
    - Crear un evento con nombre, fecha y venue.
    - Crear localidades dentro del evento, con aforo, precio base y numeración.
    - Crear ofertas (descuentos) asociadas a una localidad.
3. **Gestión de tiquetes**
    - Generar tiquetes simples o numerados según la localidad.
    - Mostrar inventario de tiquetes disponibles para venta.
4. **Compra de tiquetes (Cliente)**
    - Validar saldo del cliente.
    - Registrar los tiquetes vendidos.
    - Actualizar aforo y estado del tiquete.
5. **Transferencia de tiquetes (Cliente)**
    - Transferir tiquetes vendidos a otro cliente.
    - Validar contraseña y que el tiquete sea transferible.
    - **Bloquear transferencia si el tiquete ya fue impreso.**
6. **Configuración de tarifas (Administrador)**
    - Definir tarifa porcentual y cuota fija aplicadas a las ventas.
7. **Cancelación de eventos (Administrador)**
    - Cambiar el evento a estado CANCELADO.
    - Procesar reembolsos automáticos a todos los clientes afectados.
8. **Persistencia del sistema**
    - Guardar en CSV los datos de usuarios, eventos, localidades, ofertas, tiquetes, paquetes y pagos.
    - Restaurar el estado completo del sistema al iniciar la aplicación.

9. **Impresión de tiquetes (Cliente – Proyecto 3)**
    - Mostrar un diálogo con los datos del tiquete.
    - Generar un código QR dinámico con:
        - Nombre del evento
        - ID del tiquete
        - Fecha del evento
        - Fecha de impresión
    - Registrar la fecha de impresión.
    - **Bloquear futuras impresiones del mismo tiquete.**
10. **Restricciones del QR y seguridad**
    - El QR debe ser legible con la cámara de un celular.
    - Cada QR debe ser único para cada tiquete.
    - Los datos deben estar en texto plano dentro del QR.

Requerimientos no funcionales

1. **Interfaz gráfica en Swing**
   La aplicación debe funcionar como programa de escritorio con interfaz gráfica construida en Java Swing.

2. **Persistencia en CSV**
   No se usan bases de datos; todos los datos se almacenan y leen desde archivos CSV.

3. **Tiempo de respuesta**
    - Los cambios de ventana deben ser inmediatos.
    - La generación del QR debe tardar menos de 1 segundo.

4. **Portabilidad**
    - Debe ejecutarse en cualquier computador con Java 17 o superior.

5. **Usabilidad**
    - La navegación entre ventanas debe ser clara, con botones visibles según el rol del usuario.
    - Los mensajes de error deben ser comprensibles (contraseña incorrecta, tiquete ya impreso, etc.).
6. **Legibilidad del QR**
    - El código QR generado debe ser decodificable por aplicaciones comunes de escaneo en celulares.
7. **Mantenibilidad**
    - La lógica del dominio debe estar desacoplada de la interfaz gráfica.
    - DataStore debe ser reemplazable por una base de datos en el futuro.

9) Modelo de dominio

El sistema Boletamaster se estructura alrededor de entidades que representan los distintos roles de usuario, los eventos y la gestión de tiquetes.
A continuación se describe el modelo de dominio en formato textual, indicando las relaciones y multiplicidades principales entre las clases:

- Usuario es una clase abstracta con los atributos básicos de identificación (`id`, `login`, `password`).
  A partir de esta clase se derivan tres tipos de usuarios:
    - Cliente, que puede comprar tiquetes, abonar saldo y transferir tiquetes.
    - Organizador, que hereda de Cliente y puede crear eventos, localidades y ofertas.
    - Administrador, que hereda directamente de Usuario y gestiona las tarifas, cancelaciones y reembolsos.
- Evento representa cada espectáculo o actividad disponible.
  Está asociado a un Venue (lugar donde se realiza) en una relación de *uno a uno* (cada evento se lleva a cabo en un solo lugar).
  Un evento contiene *una o varias* Localidades, que definen zonas o secciones del lugar.

- Localidad pertenece a un evento y tiene información sobre precio base, aforo, numeración y ofertas.
  Cada localidad puede tener *cero o más* Ofertas asociadas (descuentos activos) y *uno o varios* Tiquetes disponibles para la venta.

- Tiquete corresponde a una entrada específica que puede ser simple o numerada, según si tiene o no asiento asignado.
  Cada tiquete pertenece a una localidad y puede tener como propietario *cero o un* Cliente (cuando ha sido vendido o transferido).
  El tiquete puede formar parte de *cero o más* Paquetes de tiquetes.

- PaqueteTiquetes agrupa varios tiquetes bajo un mismo beneficio o precio total.
  Existen dos tipos de paquetes: el PaqueteTiquetes básico y el PaqueteDeluxe, que incluye beneficios adicionales.

- Pago registra las compras de tiquetes realizadas por los clientes.
  Un pago puede incluir *cero o varios* tiquetes dentro de la misma transacción.

- Venue describe el lugar donde se realizan los eventos, con su capacidad, ubicación y fechas reservadas.

Herencias y jerarquías

- Cliente hereda de Usuario.

- Organizador hereda de Cliente.

- Administrador hereda directamente de Usuario.

- TiqueteNumerado y TiqueteSimple heredan de Tiquete.

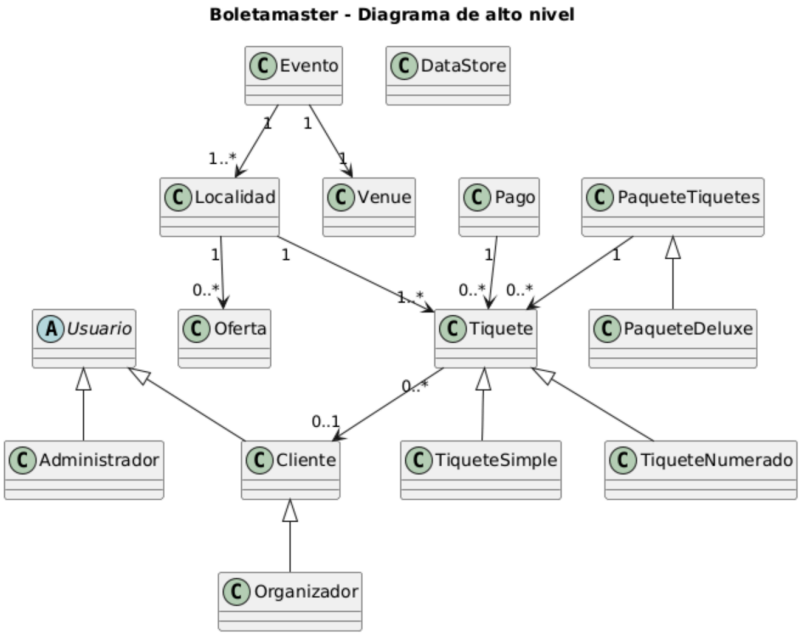
- PaqueteDeluxe hereda de PaqueteTiquetes.

Estados relevantes del dominio

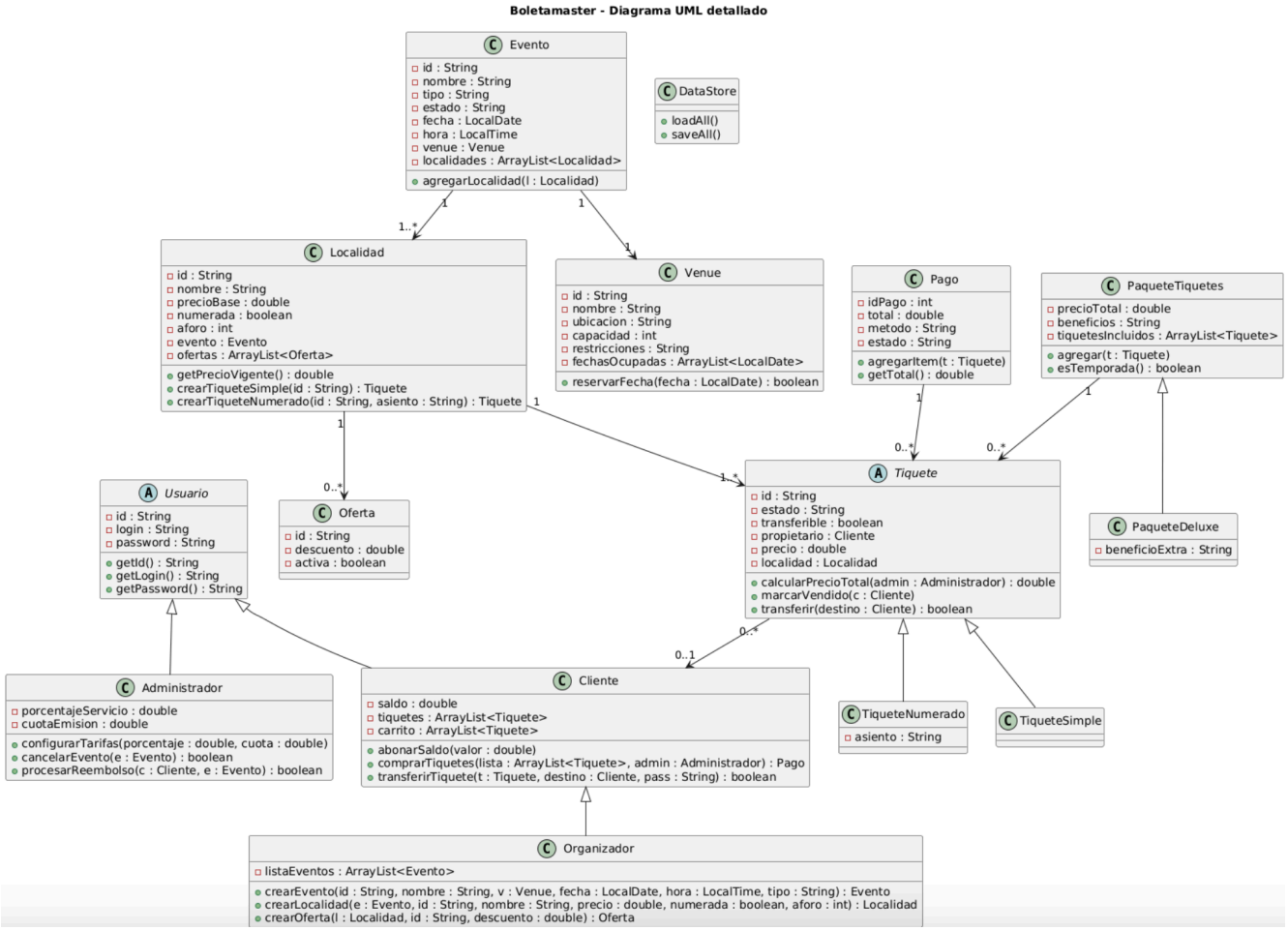
- Evento: puede estar en los estados `{PROGRAMADO, CANCELADO}`.

- Tiquete: puede estar en los estados `{DISPONIBLE, VENDIDO, TRANSFERIDO}`

10) Diagrama UML de Alto Nivel

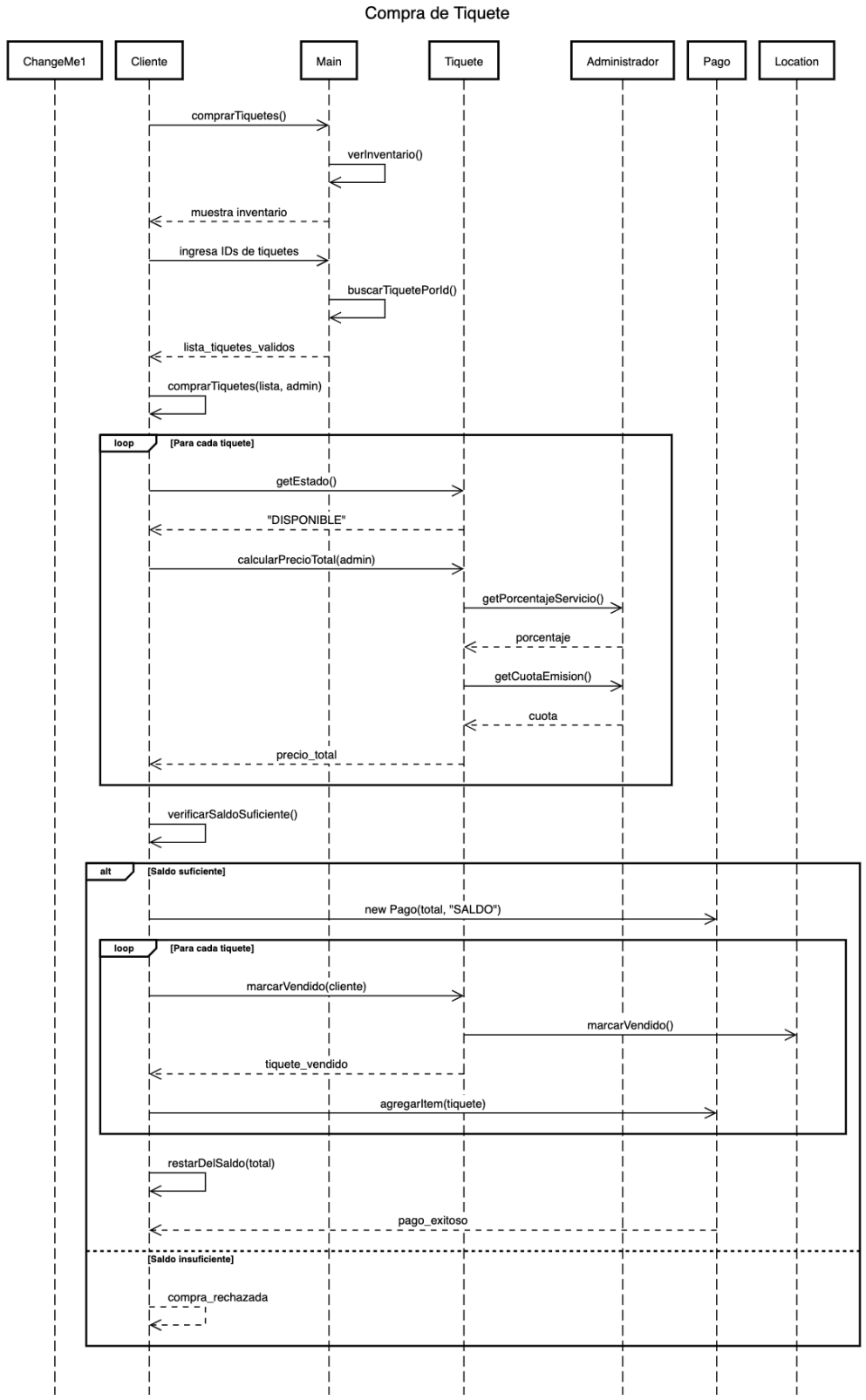


11) Diagrama UML detallado

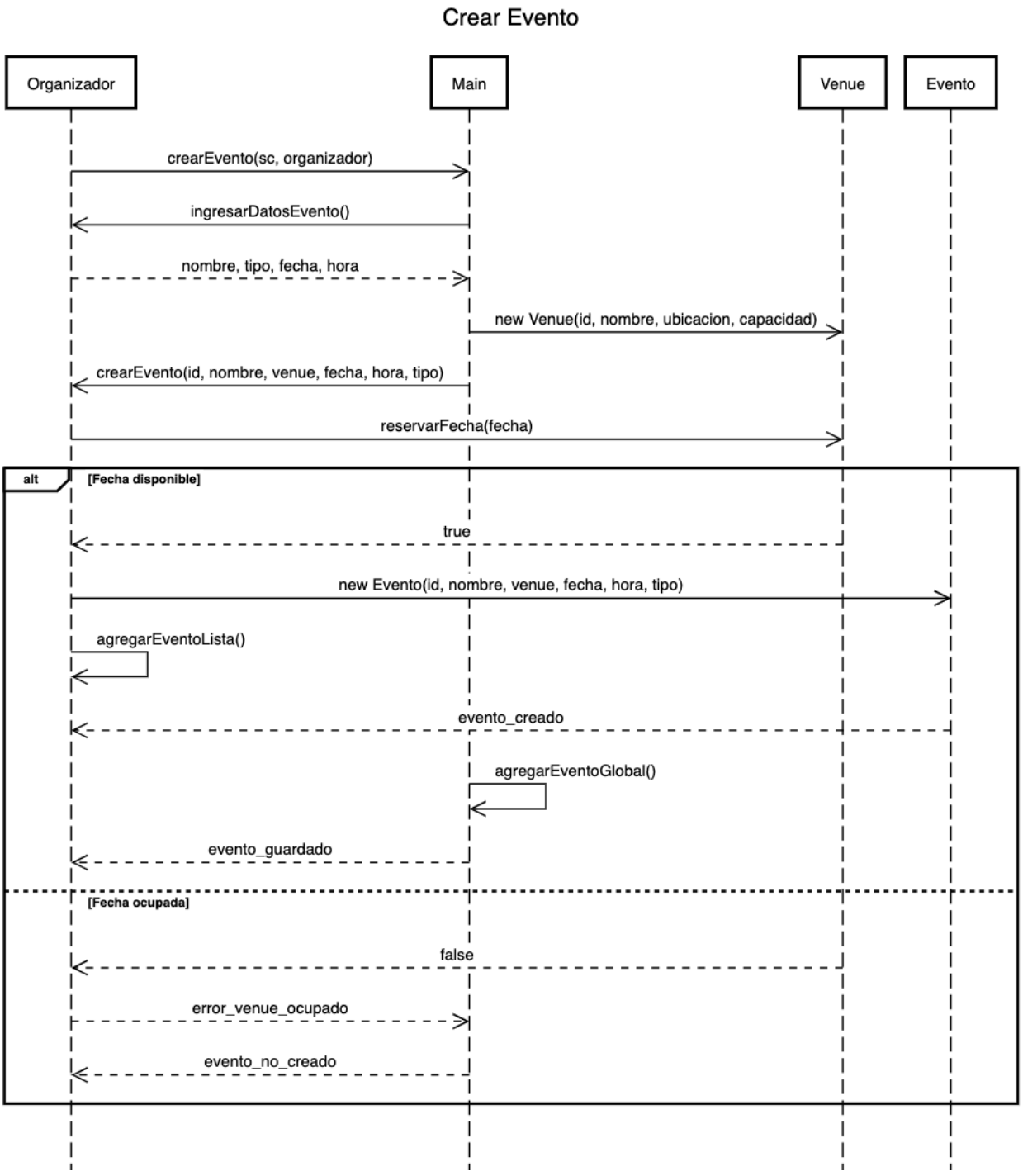


12) Diagramas de secuencia

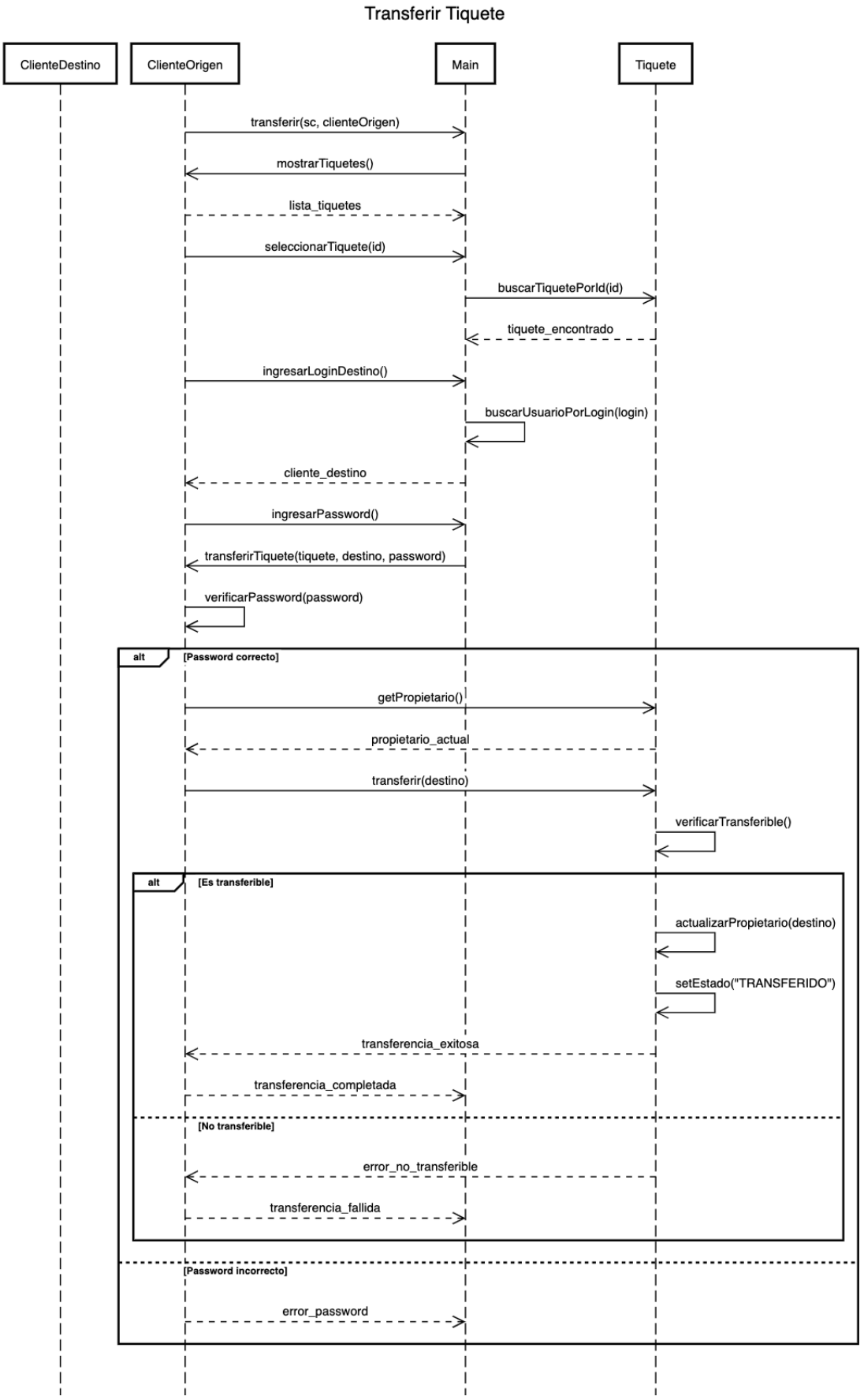
a) Comprar tiquete



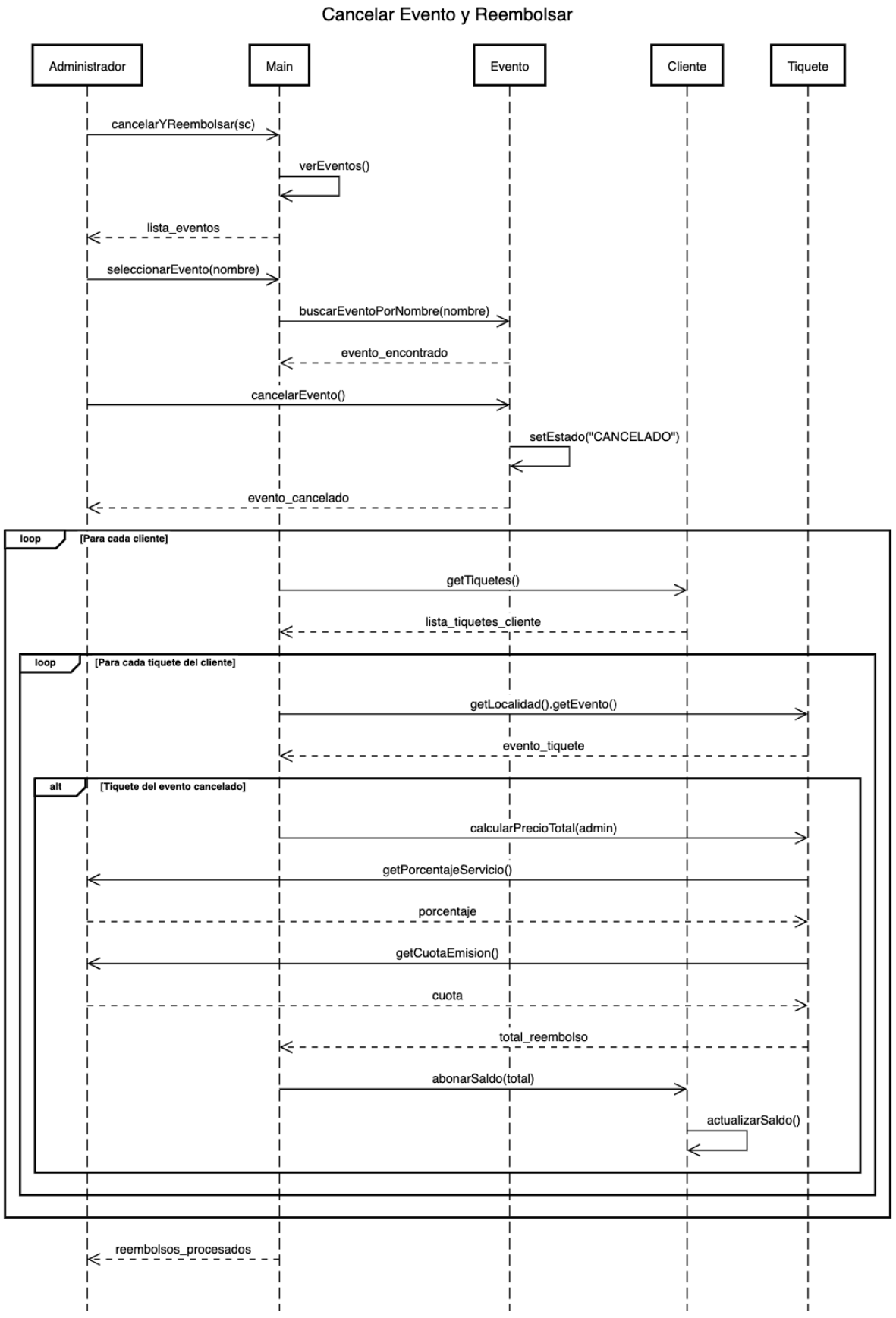
b) Crear Evento



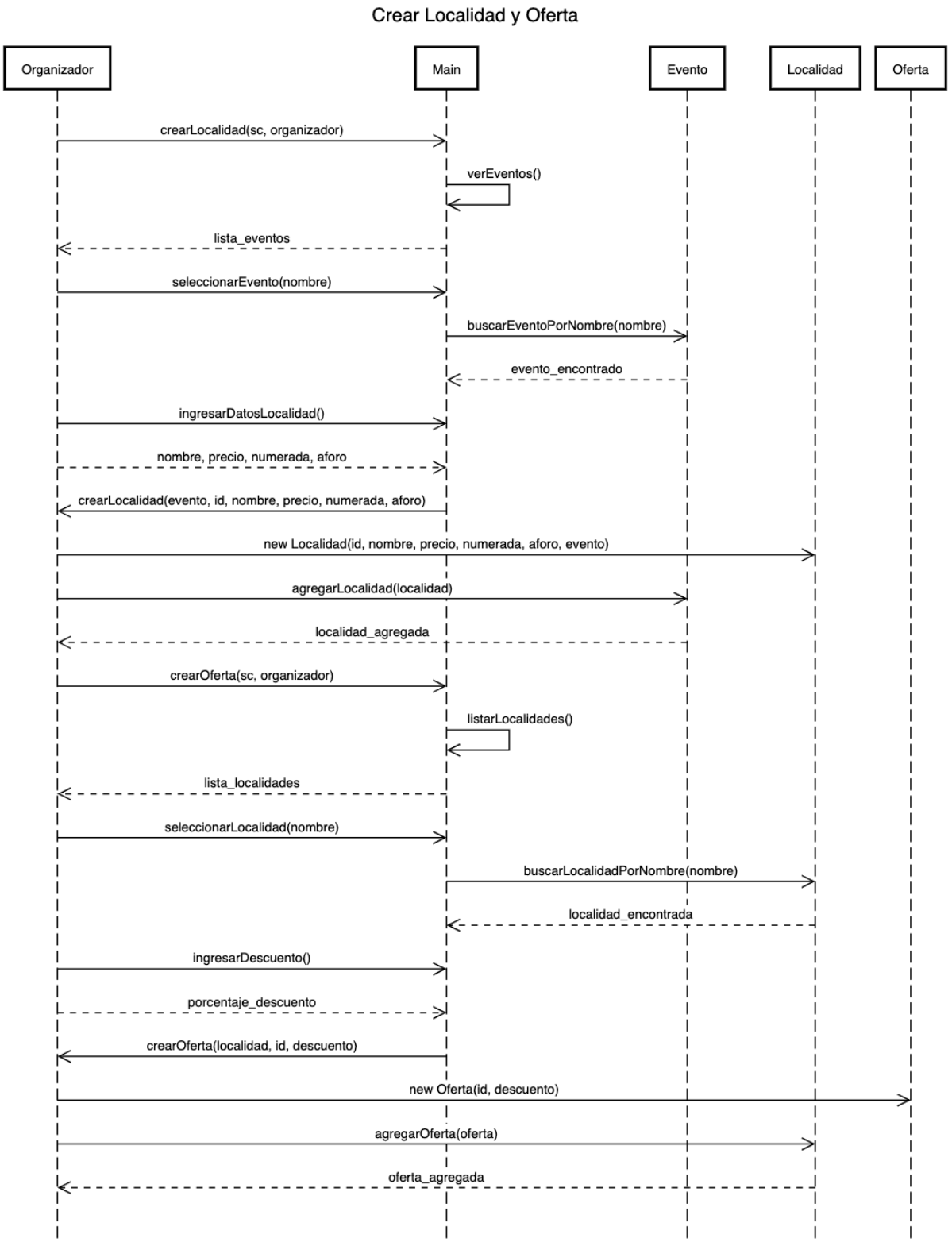
c) Transferir tiquete



d) Cancelar Evento y Reembolsar

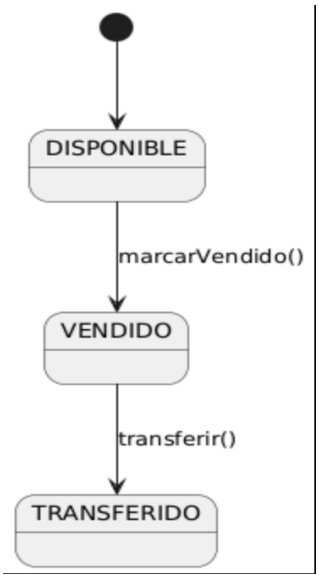


e) Crear Localidad y Oferta

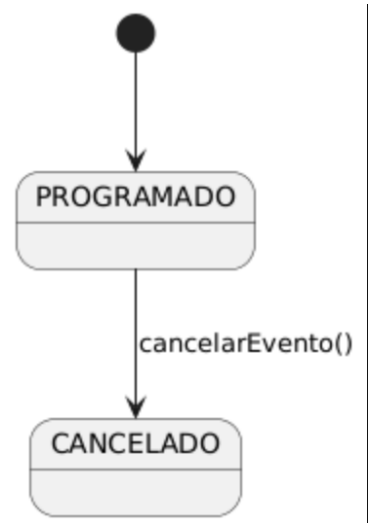


13) Diagramas de estado

a) Tiquete



b) Evento



14) Asignación de responsabilidades

Clase	Responsabilidad
`Administrador`	Configurar tarifas, cancelar eventos, reembolsar clientes.
`Cliente`	Comprar, abonar saldo, transferir tiquetes.
`Organizador`	Crear eventos, localidades y ofertas.
`Evento`	Mantener fecha, hora, estado y localidades.

`Localidad`	Calcular precios, manejar aforo, asociar ofertas.
`Oferta`	Aplicar descuento activo a localidad.
`Tiquete`	Representar entrada, manejar estado, calcular precio total.
`PaqueteTiquetes`	Agrupar tiquetes, manejar beneficios y tipo (deluxe o normal).
`Pago`	Registrar transacción de compra.
`Venue`	Reservar fechas y almacenar capacidad.
`DataStore`	Persistir y reconstruir datos desde CSV.
`Main`	Consola de interacción, orquestar flujos de uso.

15) Justificaciones de diseño

1. **CSV sobre base de datos:** simplicidad, portabilidad y trazabilidad de datos.
2. **Herencia en Usuario/Tiquete:** reutilización y polimorfismo.
3. **Control centralizado en Main:** facilita pruebas y separación futura de interfaz.
4. **Tarifas globales en Administrador:** una fuente de verdad para recargos.
5. **Reglas de negocio localizadas:** cada clase gestiona su propia lógica.
6. **Cálculo de precios:** implementado en `tiquete` para consistencia entre compra y reembolso.

16) Reglas de dominio

- `Localidad.vendidos ≤ aforo`.
- `TiqueteDeluxe` no es transferible.
- `Venue.reservarFecha()` evita eventos simultáneos en la misma fecha.
- Reembolso = suma de `calcularPrecioTotal(admin)` de tiquetes asociados.

- `Evento.cancelarEvento()` cambia estado a CANCELADO y activa reembolsos.

17) Reglas de Negocio

- Capacidad: Vendidos debe ser menor o igual al aforo.
- Venue no puede tener dos eventos al día
- Solo se venden tiquetes disponibles.
- Solo se transfieren tiquetes vendidos y transferibles.
- Transferencia requiere contraseña del cliente
- Reembolso = suma del precio total calculado con tarifas del admin.
- Reembolsos se abonan al saldo.
- Un paquete es “de temporada” si incluye eventos distintos.
- Paquete deluxe no transferible

18) Principios de Diseño (SOLID y GRASP)

SOLID

- **S (Responsabilidad Única):** cada clase tiene una función clara (`Cliente` compra, `Administrador` reembolsa, `DataStore` persiste).
- **O (Abierto/Cerrado):** `Tiquete` se extiende en `TiqueteSimple` y `TiqueteNumerado` sin modificar la clase base.
- **L (Sustitución de Liskov):** `Organizador` y `Cliente` pueden sustituir a `Usuario`.
- **I (Segregación de Interfaces):** previsto para próxima entrega (`IPersistente`, `ITransaccion`).
- **D (Inversión de Dependencias):** `Main` depende de `DataStore`, no de archivos.

GRASP

- **Information Expert:** `Tiquete` calcula precios, `Localidad` maneja aforo.
- **Creator:** `Organizador` crea `Evento`, `Localidad`, `Oferta`.

- **Controller:** `Main` coordina la consola.
- **Low Coupling / High Cohesion:** clases bien delimitadas y autónomas.
- **Indirection / Pure Fabrication:** `DataStore` como intermediario de persistencia.
- **Polymorphism:** herencias de `Usuario` y `Tiquete`.
- **Protected Variations:** la estructura CSV puede cambiar sin afectar la lógica.

19) Correspondencia con el código

Caso de uso	Método en código
Configurar tarifas	`Administrador.configurarTarifas()`
Cancelar evento	`Administrador.cancelarEvento()`
Procesar reembolso	`Administrador.procesarReembolso()`
Crear evento	`Organizador.crearEvento()`

Crear localidad	`Organizador.crearLocalidad()`
Crear oferta	`Organizador.crearOferta()`
Comprar tiquete	`Cliente.comprarTiquetes()`
Transferir tiquete	`Cliente.transferirTiquete()`
Abonar saldo	`Cliente.abonarSaldo()`

19) Pruebas Unitarias

Alcance y estrategia

Objetivo: verificar la lógica del dominio y la persistencia de BoletaMaster con pruebas rápidas, aisladas y repetibles.

Herramienta: JUnit 5 (Jupiter).

Enfoque (pirámide de pruebas):

Unidad (foco principal): métodos de clases del dominio (sin I/O).

Integración ligera: DataStore + Csv con archivos en directorio temporal.

Sin testear Main.main: el punto de entrada no contiene reglas; la lógica se cubre indirectamente.

Estructura y convenciones

Nombre de clases de test: ClaseBajoPruebaTest (p. ej. EventoTest, LocalidadTest).

Nombre de métodos: metodo_condicion_resultado()
Ej.: transferirTiquete_passwordIncorrecta_falla().

Patrón AAA: Arrange – Act – Assert en cada prueba.

Datos de prueba:

IDs cortos: E1, L1, C1, V1.

Fechas/hora determinísticas (2026-05-20, 19:30).

Cobertura por componente

Administrador

configurarTarifas(%) y cuotaEmision: asigna valores válidos; rechaza negativos.

cancelarEvento(e): estado pasa a CANCELADO; dispara regla de reembolso cuando aplique.

Organizador

crearEvento(...): retorna evento con Venue asociado y reserva fecha.

crearLocalidad(...): crea localidad, setea aforo y asociación a evento.

crearOferta(...): agrega oferta activa a la localidad.

Cliente

abonarSaldo(monto): aumenta saldo; rechaza montos negativos.

comprarTiquetes(...): sólo si hay saldo y DISPONIBLE; cambia a VENDIDO; actualiza vendidos ≤ aforo.

transferirTiquete(t, password, destino): requiere propiedad + password válida + transferible.

Tiquete / TiqueteSimple / TiqueteNumerado

Estados: DISPONIBLE → VENDIDO → (opcional) USADO/TRANSFERIDO.

isTransferible() respeta deluxe no transferible (si aplica).

En numerados, conserva asiento.

Localidad

Invariante: vendidos ≤ aforo.

Cálculo de precio base + ofertas activas.

Oferta

descuento ∈ [0,1]; estado activa/inactiva.

Evento

Estado inicial PROGRAMADO; cancelar() → CANCELADO.

Mantiene fecha, hora, venue.

Venue

reservarFecha(fecha): retorna true la primera vez, false si ya estaba ocupada.

Lleva fechasOcupadas coherentes.

20) Diseño preliminar de la interfaz gráfica

Antes de implementar la interfaz gráfica en Java Swing, realizamos un diseño preliminar para definir cómo se verían las ventanas y qué elementos debía contener cada una. Esto nos permitió planear correctamente la navegación y ubicar los componentes necesarios de acuerdo con las funcionalidades del sistema.

A continuación describimos los mockups utilizados:

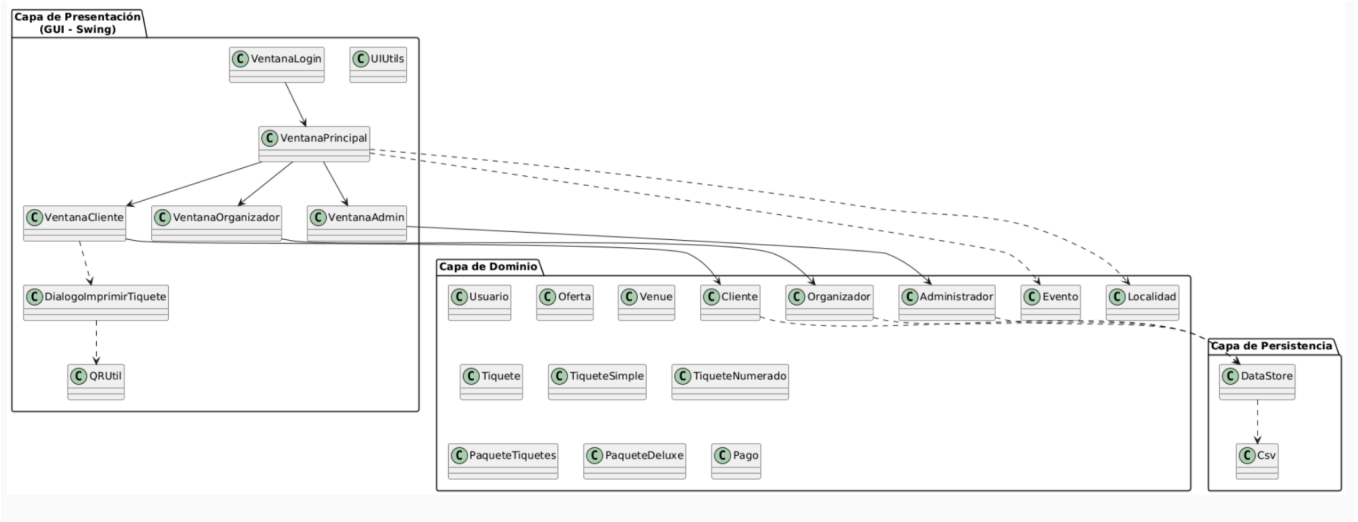
Mockup 1: Ventana de Inicio de Sesión

Mockup 2: Ventana Principal (según rol)

Mockup 3: Ventana de impresión de tiquete

21) Arquitectura actualizada incluyendo la interfaz gráfica

El sistema ahora utiliza una arquitectura **multicapa**, adaptada al Proyecto 3:



Capa 1 — Presentación (GUI – Java Swing)

Compuesta por:

- `VentanaPrincipal`
- `VentanaLogin`
- `VentanaCliente`
- `VentanaOrganizador`
- `VentanaAdmin`
- `DialogoImprimirTiquete`
- `UIUtils`
- `QRUtil`

Responsabilidades:

- Interactuar con el usuario final.

- Mostrar listas, formularios, botones y cuadros de diálogo.
- Enviar las acciones del usuario al dominio.
- Mostrar errores o mensajes de confirmación.

Capa 2 — Dominio

Incluye:
`Usuario`, `Cliente`, `Organizador`, `Administrador`,
`Evento`, `Localidad`, `Oferta`, `Venue`,
`Tiquete`, `TiqueteSimple`, `TiqueteNumerado`,
`PaqueteTiquetes`, `PaqueteDeluxe`, `Pago`

Responsabilidades:

- Contener la lógica del negocio.
- Aplicar reglas de dominio (transacciones, compra, transferencia, impresión, etc.).

Capa 3 — Persistencia

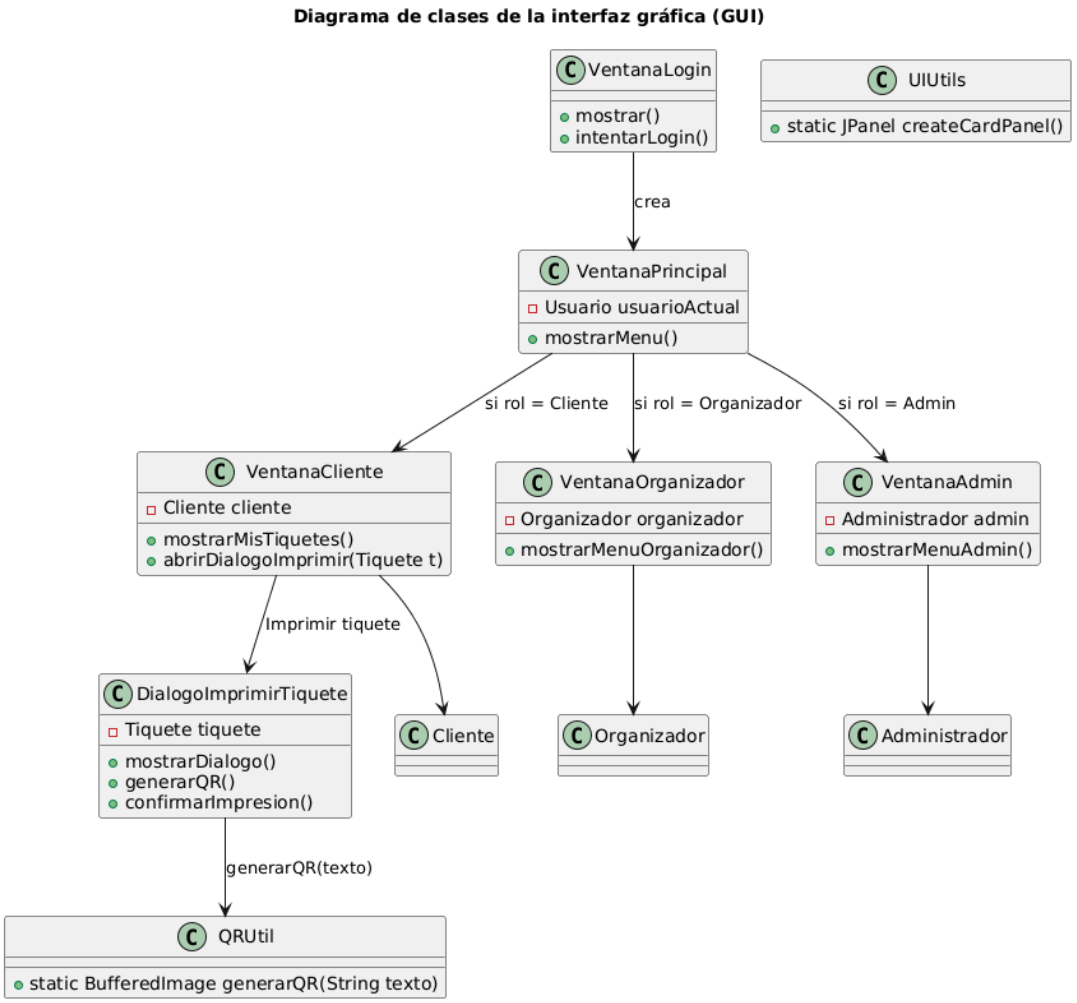
Clases:

- `DataStore`
- `Csv`

Responsabilidades:

- Leer y escribir archivos CSV.
- Reconstruir relaciones entre entidades.
- Persistir cambios después de compras, transferencias o impresiones.

22) Diagrama de clases de la interfaz gráfica



El diagrama de clases de la interfaz incluye las siguientes clases:

- `VentanaPrincipal`
- `VentanaLogin`
- `VentanaCliente`
- `VentanaOrganizador`
- `VeteranaAdmin`
- `DialogoImprimirTiquete`
- `QRUtil`

- `UIUtils`

Relaciones importantes:

- `VentanaLogin` crea `VentanaPrincipal`.
- `VentanaPrincipal` determina el rol y abre la ventana correspondiente.
- `VentanaCliente` invoca a `DialogoImprimirTiquete`.
- `DialogoImprimirTiquete` usa `QRUtil` para generar el código QR.
- Todas las ventanas leen y modifican datos del dominio usando `Main` y `DataStore`.

23) Sistema de impresión de tiquetes

Objetivo

Permitir que cada cliente pueda imprimir un tiquete comprado, generando un código QR dinámico que contenga los datos esenciales del tiquete, cumpliendo los requisitos del Proyecto 3.

Flujo del sistema

1. El cliente abre “Mis tiquetes” desde la interfaz.
2. Selecciona un tiquete y elige “Imprimir”.
3. Se abre `DialogoImprimirTiquete`.
4. El sistema genera un código QR usando `QRUtil`.
5. Se registra la fecha de impresión usando `LocalDate.now()`.
6. Se marca el atributo `impreso = true` en el objeto `Tiquete`.
7. Se llama a `DataStore.saveAll()` para guardar cambios.
8. Un tiquete impreso queda bloqueado para:
    - Volver a imprimirse
    - Transferirse a otro usuario

Contenido del código QR

El QR generado contiene:

Evento: <nombre evento>

ID: <id del tiquete>

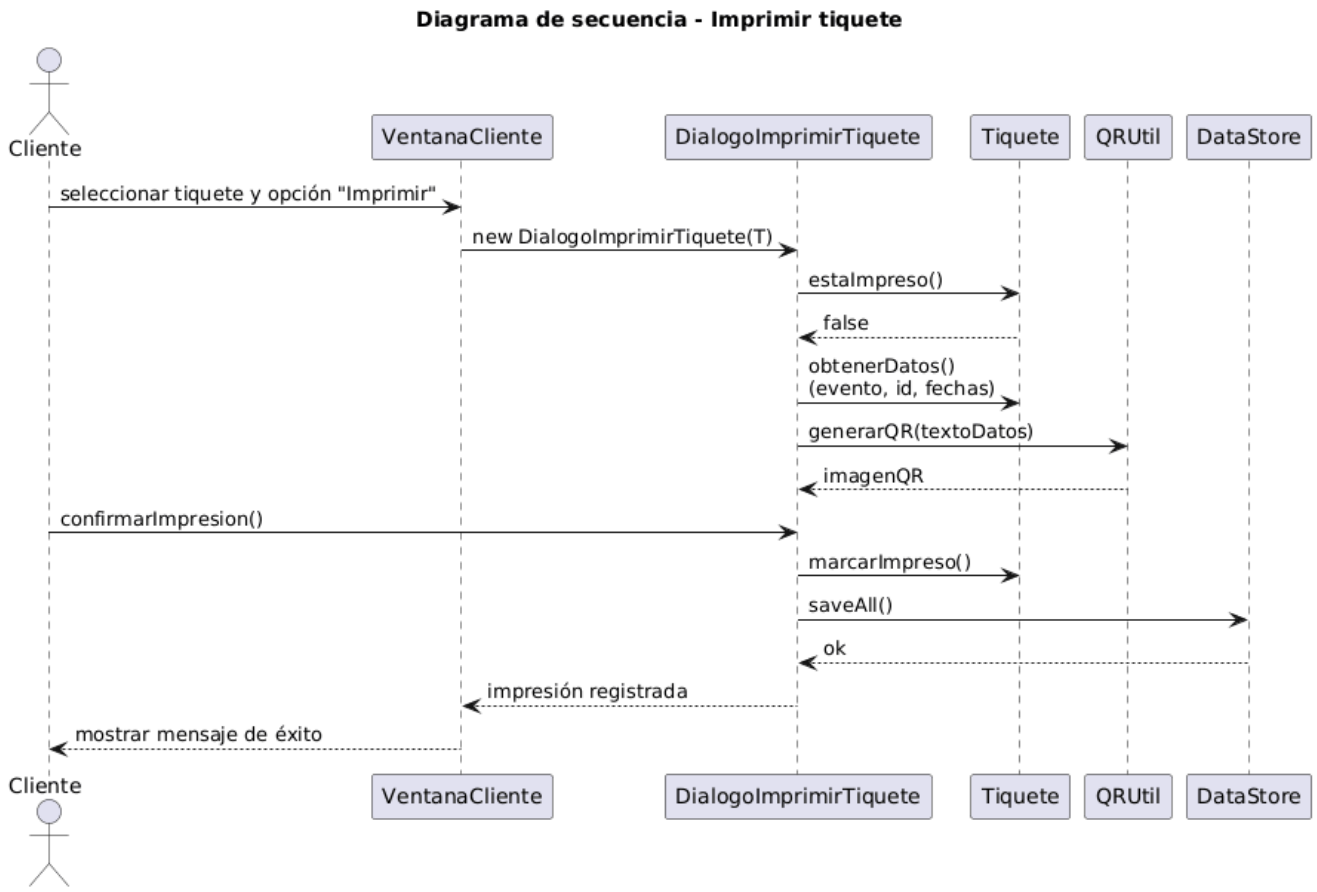
FechaEvento: <fecha del evento>

FechaImpresion: <fecha actual>

Componentes del sistema

- **DialogoImprimirTiquete**
    - Renderiza la vista previa del tiquete.
    - Muestra el QR.
    - Botón de “Imprimir”.
    - Cambia el estado del tiquete.
- **QRUtil**
    - Construye el QR usando BufferedImage.
    - Codifica la información del tiquete como texto plano.
    - Devuelve la imagen para mostrarla.
- **Tiquete**
    - Posee el atributo `boolean impreso`.
- **DataStore**
    - Guarda el estado del tiquete actualizado en `tiquetes.csv`.

25) Diagrama de secuencia: Imprimir Tiquete



26) Pruebas añadidas para impresión

Se agregaron pruebas manuales:

Prueba 1 — QR legible

- Se genera un QR desde DialogoImprimirTiquete.
- Se escanea con la cámara de un celular.
- Se valida que aparezcan los datos del tiquete.

Prueba 2 — Bloqueo de reimpresión

- Se imprime un tiquete.
- Se intenta abrir nuevamente el diálogo.
- El sistema muestra error: “El tiquete ya fue impreso”.

Prueba 3 — Bloqueo de transferencia

- Se imprime un tiquete.
- Se intenta transferir.
- El sistema niega la acción.

27) Conclusiones

En el Proyecto 3 se completó la evolución de BoletaMaster desde una aplicación de consola hacia una aplicación de escritorio con interfaz gráfica en Java Swing. Esto permitió que la experiencia del usuario fuera más intuitiva y cercana a un sistema real de boletería.

La arquitectura se reorganizó en capas (GUI, dominio y persistencia), manteniendo cohesión interna y bajo acoplamiento entre componentes. También se amplió el modelo del dominio para incluir la funcionalidad de impresión de tiquetes con códigos QR dinámicos, cumpliendo las nuevas reglas del proyecto.

Se documentaron los diagramas principales, el diseño preliminar de la interfaz gráfica, las reglas de negocio y los flujos del sistema. En conjunto, el diseño final demuestra una implementación clara, modular y coherente con los requisitos establecidos para BoletaMaster.