Diseño del Sistema Boletamaster
Curso: DPOO - Uniandes
Autores : Mateo Zea Alzate, Johan Santanilla, Santiago Bobadilla
Proyecto: Boletamaster

Estado de la Aplicación

El diseño del sistema **Boletamaster** representa de manera completa y estructurada todo el estado relevante de la aplicación, según el UML actualizado. El modelo captura entidades, relaciones, multiplicidades y comportamientos coherentes con la gestión de eventos, usuarios y ventas de tiquetes.

Estructura principal del modelo:

- **Usuario (abstracta):** clase base con atributos compartidos entre todos los usuarios (id, nombre, email, login, password, saldo, fechaRegistro). Define operaciones genéricas como autenticar(), acreditarSaldo() y debitarSaldo().

- **Administrador:** hereda de Usuario. Controla la creación y mantenimiento de eventos y venues. Métodos: fijarTarifaServicio(), fijarCuotaEmision(), aprobarVenue(), cancelarEvento(), autorizarReembolso() y consultarFinanzas().

- **Promotor:** Hereda de Usuario. Administra los eventos que crea. Métodos: crearEvento(), configurarLocalidades(), crearOferta(), y consultarFinanzas().

- **Cliente:** Hereda de Usuario. Contiene documento y telefono. Gestiona sus tiquetes mediante GestionTiquetes.

- **Evento:** entidad central que asocia Venue, Localidad, Oferta y FinanzasEvento. Atributos incluyen tipo, idEvento, fecha, hora y estado (esActivo).

- **Venue:** representa el lugar físico con atributos como nombre, ubicación, capacidad y restricciones.

- **Localidad:** define secciones del venue (idLocalidad, precio, características, disponible, esNumerada).

- **Asiento:** representa una posición numerada dentro de una localidad (numeroAsiento, fila, disponible).

- **Tiquete / TiqueteMultiple / PaqueteDeluxe:** modelan diferentes tipos de entrada adquirida.

- **FinanzasEvento:** encapsula la lógica de ganancias, porcentaje de venta y cobros asociados.

- **GestionTiquetes:** controla operaciones sobre tiquetes (compra, abono, transferencia, reembolso).

- **Repositorios (Usuarios, Eventos, Venues):** gestionan la persistencia y almacenamiento de datos en disco.

Soporte a Funcionalidades

El diseño soporta **todas las funcionalidades del enunciado** del proyecto: creación de eventos, compra de tiquetes, gestión de venues, transferencias, ofertas, finanzas y reembolsos.

Por tipo de usuario:

- **Administrador:** aprueba venues, fija tarifas, cancela eventos, autoriza reembolsos y consulta finanzas.

- **Promotor:** crea y configura eventos, define ofertas y consulta ganancias.

- **Cliente:** compra tiquetes, transfiere tiquetes, solicita reembolsos.

Módulos complementarios:

- GestionTiquetes: orquesta todas las operaciones entre eventos, localidades y clientes.

- FinanzasEvento: encapsula el cálculo de ingresos y porcentajes de venta.

- PlainTextUserRepository: gestiona persistencia de usuarios en archivos .txt.

Documentación de la Persistencia

Los datos de usuarios y tiquetes se almacenan en archivos .txt en formato CSV delimitado por punto y coma (;), mientras que los eventos y venues se mantienen en memoria mediante estructuras HashMap, separadas entre activos e inactivos.

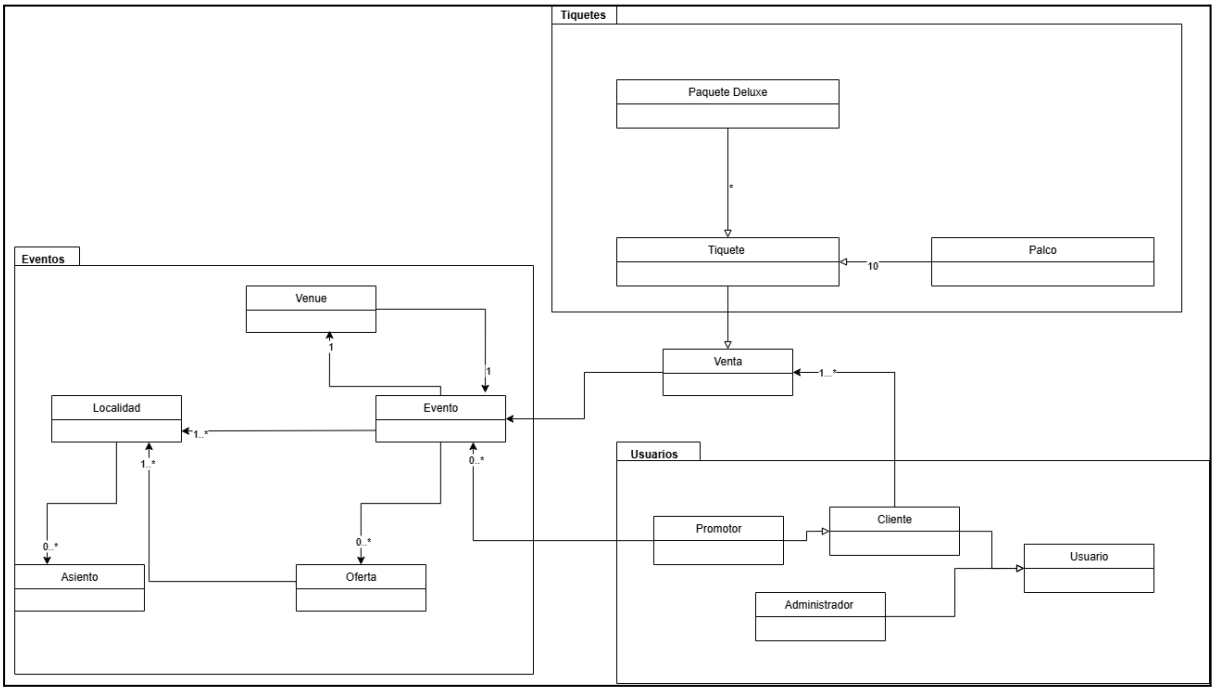
- Usuarios: gestionados por RepositorioUsuarios, con operaciones guardarUsuario() y cargarUsuarios(). Cada línea del archivo representa un usuario (Cliente, Promotor o Administrador) con sus atributos básicos y saldo.

- Tiquetes: manejados por Repositorio_tiquetes, con soporte para Individual, Multiple y Deluxe. Se guarda el id del evento, precios, beneficios (en el caso deluxe) y características de asiento o cantidad.

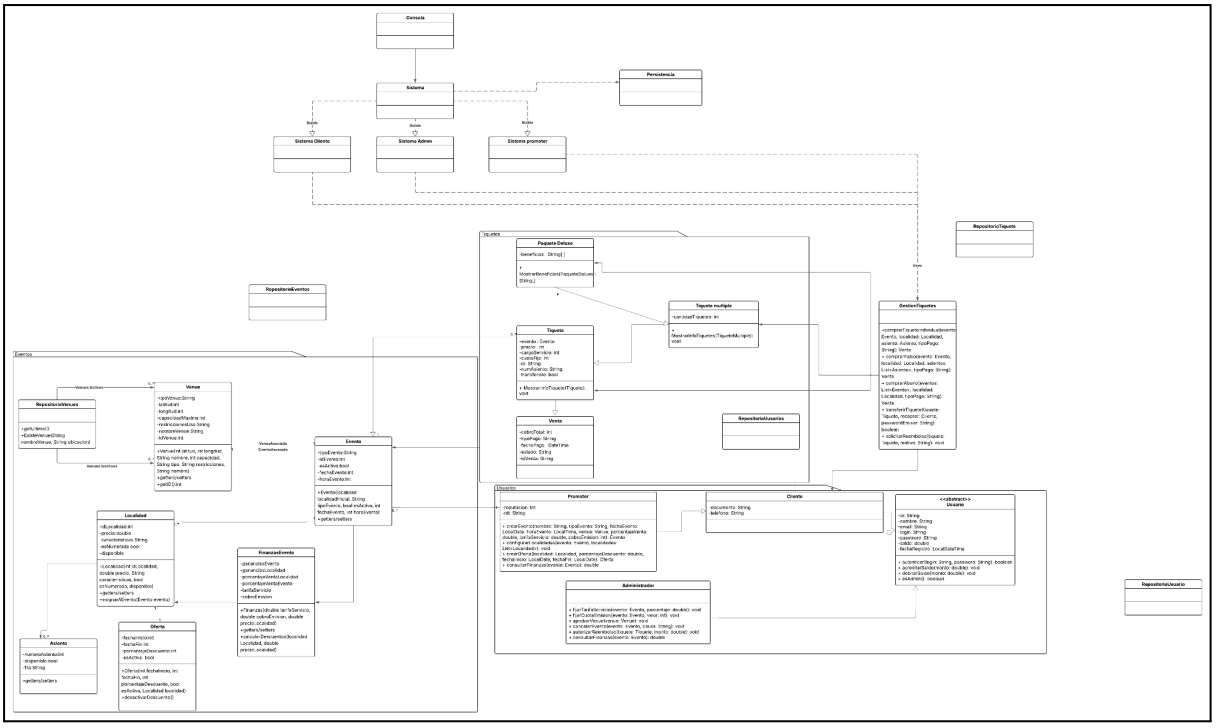
- Eventos y Venues: administrados en tiempo de ejecución por RepositorioEventos y RepositorioVenues usando mapas (Map<Integer, Evento> y Map<Integer, Venue>), lo que permite registrar, desactivar y finalizar objetos de manera eficiente

Diagramas UML

4.1. Diagrama de Alto Nivel:



4.2. Diagrama Detallado:



5. Asignación de Responsabilidades

Clase	Responsabilidad principal
Usuario	Gestión genérica de usuarios y saldo
Administrador	Control global del sistema (venues, eventos, finanzas)
Promotor	Configuración de eventos/ofertas
Cliente	Compra, transferencia y reembolso de tiquetes
Evento	Representar información, crear evento y estado de eventos
Venue	Representar el lugar físico de los eventos

Localidad	Modelar sectores del venue y precios
Asiento	Controlar posiciones disponibles dentro de localidades
Tiquete	Asociar cliente-evento y precios finales
Oferta	Modelar descuentos por fechas o localidades
FinanzasEvento	Calcular cobros y descuentos aplicados a eventos
GestionTiquetes	Coordinar operaciones entre usuarios y eventos
Repositorios	Manejar la persistencia de usuarios eventos y tiquetes

6. Colaboración entre Clases

. Usuario ↔ RepositorioUsuarios

- Los objetos Usuario (y sus subclases) se crean y persisten mediante el RepositorioUsuarios.
- RepositorioUsuarios.guardarUsuario() serializa los datos de Usuario al archivo .txt, y cargarUsuarios() los reconstruye.
- Esta colaboración garantiza que las operaciones de autenticación y registro funcionen de manera persistente entre ejecuciones.

2. Promotor ↔ Evento ↔ Venue ↔ Localidad

- Promotor crea nuevos eventos (crearEvento()), configurando Localidad y asociándolos a un Venue aprobado por el Administrador.
- La relación es de composición: si se elimina un Evento, se eliminan sus Localidades y Ofertas.

- Esta colaboración mantiene la coherencia del dominio de negocio.

3. Cliente ↔ GestionTiquetes ↔ Evento

- Cliente delega las operaciones de compra, transferencia y reembolso a GestionTiquetes.
- GestionTiquetes consulta el estado de Evento y Localidad para validar disponibilidad y precios antes de emitir un Tiquete.
- En caso de transferencia, se actualizan las referencias del Cliente propietario en el Tiquete.

4. Administrador ↔ Evento / Venue / FinanzasEvento

- El Administrador actúa como supervisor global:
    - Usa aprobarVenue() para permitir eventos.
    - cancelarEvento() para anular un evento activo.
    - consultarFinanzas() accede a FinanzasEvento para obtener estadísticas de venta.

5. Oferta ↔ Localidad / Evento

- Oferta define descuentos que afectan una Localidad o conjunto de Tiquetes durante un rango de fechas.
- Se utiliza dependencia unidireccional, donde Evento consulta Oferta antes de calcular el precio final

6. Relaciones de asociación clave (UML)

- Usuario (1) ↔ (N) Tiquete
- Promotor (1) ↔ (N) Evento
- Evento (1) ↔ (N) Localidad
- Localidad (1) ↔ (N) Tiquete
- Evento (1) ↔ (1) FinanzasEvento
- Administrador (1) ↔ (N) Venue

8. Requerimientos Funcionales

#	Requerimiento Funcional	Descripción	Clases / Métodos Relacionados
1	Registro de usuario	Permitir crear nuevos usuarios del tipo Cliente, Promotor o Administrador.	RepositorioUsuarios.guardarUsuario(), constructores de Cliente, Promotor, Administrador
2	Autenticación de usuario	Validar credenciales para ingresar al sistema.	Usuario.autenticar(login, password)
3	Creación de evento	El Promotor puede crear un evento con su venue, tipo y fecha.	Promotor.crearEvento(), Evento
4	Configurar localidades	Asociar localidades con precios y capacidades a un evento.	Promotor.configurarLocalidades(), Evento.configurarLocalidades()
5	Crear oferta / descuento	El Promotor puede crear ofertas temporales con porcentaje de descuento.	Promotor.crearOferta(), Oferta
6	Comprar tiquete	El Cliente puede comprar un tiquete individual, palco o abono.	Cliente.comprarTiquete(), GestionTiquetes.venderTiquete()

7	Transferir tiquete	El Cliente puede transferir un tiquete a otro usuario.	Cliente.transferirTiquete(), GestionTiquetes.transferir()
8	Solicitar reembolso	El Cliente puede pedir un reembolso de su compra.	Cliente.solicitarReembolso(), GestionTiquetes.reembolsar()

8.Justificaciones y Decisiones de Diseño

- **Herencia estructural:** evita duplicación de código entre los tres tipos de usuario.
- **Composición en Evento:** garantiza que cada evento tenga su propio conjunto de localidades, tiquetes y finanzas asociadas.
- **Repositorio independiente:** separa la lógica de negocio de la persistencia.
- **UUID e IDs únicos:** aseguran integridad en la creación de entidades sin colisiones.
- **Validación con Objects.requireNonNull:** previene errores de integridad en constructores.

9. Diagramas de Secuencia

Caso 1: Compra de tiquete individual (Cliente)

- Cliente → GestionTiquetes → Evento → Tiquete → Venta
  Flujo: validación de disponibilidad, cálculo de precio, creación del tiquete y confirmación de venta.

Caso 2: Transferencia de tiquete (Cliente)

- Cliente → GestionTiquetes → Tiquete → Cliente receptor
  Flujo: verificación de propiedad y contraseña, reasignación de dueño y actualización en colecciones.

Caso 3: Cancelación de evento (Administrador)

- Administrador → Evento → Tiquete → Cliente
  Flujo: cancelación, notificación y reembolso de tiquetes afectados.