Universidad de los Andes

Ingeniería de Sistemas y Computación

ISIS- 1226: Diseño y Programación Orientado a Objetos

**“Proyecto #1: Boletmaster”**

**Presentado a:**

Camilo Andrés Ortiz Casas

Carlos Alberto Poveda Riaño

Sabrina Sadday Noguera Beltran

**Presentado por:**

Andrés Huertas Lobo

David Alejandro Lucuara

# **Contexto y Alcance**

El sistema *Boletamaster* busca emular una plataforma digital de venta de tiquetes para eventos de varios tipos, como musicales, culturales, deportivos, etc. De este modo, dicho programa va a ser realizado en Java, aplicando principios de la programación orientada objetos, persistencia de datos mediante archivos y diseño modular. Para entender mejor el funcionamiento, tenemos en cuenta tres roles principales: Clientes, Organizadores, Promotores y Administradores. El principio fundamental es implementar la operación de una empresa tipo boletería con las siguientes funciones: Publicación de eventos, configurar localidades, reportes de estados financieros, entre muchas más. En este proyecto se priorizará la lógica del dominio y pruebas de la persistencia.

**Gestión de eventos y *venues***

Los organizadores estarán a cargo de crear los eventos, definiendo hora, fecha y la asociación obligatoria a un *venue* con cierta capacidad. Por temas de logística, un *venue* no podrá tener más de un evento por día. El administrador determina si aprueba un *venue* o no.

**Modelación de localidades y tipos de tiquete**

Las localidades definen un precio y una característica en especifico para su tiquete. Si una localidad llega a ser numerada, es importante que cada tiquete lleve un identificador único para evitar repeticiones de los asientos que conlleven a inconsistencias en las ventas, en su propia localidad. Un evento puede tener varios tipos de tiquete con una o varias localidades disponibles.

**Venta de tiquetes**

Cada tiquete tendrá un precio base, cargo porcentual (ingreso generado por la boletería) y una cuota fija por emisión/impresión. Se necesita un identificador único para cada tiquete para evitar amenazas de seguridad como la falsificación. Se pueden comprar varios tiquetes por transacción, con un tope configurable. Se tiene en cuenta que las compras se hacen con un saldo de una plataforma ya tercerizada.

**Tiquetes múltiples y Paquete *deluxe***

Los tiquetes múltiples se venden en bloque; también existen paquetes que agrupan entradas para varios eventos. El límite por transacción en este caso aplica para la agrupación más no para los elementos internos. El precio de dicho grupo puede diferir con la suma de sus componentes. Para los paquetes *deluxe* también se tienen en cuenta mercancía y beneficios, e incluso tiquetes adicionales. Sin embargo, no se pueden transferir los tiquetes de paquete *deluxe*.

**Transferencia y rembolso**

En dado caso de que se vaya a hacer una transferencia, es importante tener el usuario y contraseña del dueño de la boleta por temas de seguridad. He de recordar que para paquetes *deluxe* no se pueden transferir tiquetes. Para un tiquete múltiple, se pueden transferir los tiquetes individuales o todo el paquete, con tal de que ninguno de los elementos haya pasado su fecha de vencimiento o ya se hayan transferido. Para los rembolsos, por cancelación del administrador (restando la cuota de emisión), por solicitud del organizador (solo tener en cuenta precio base ya que los cargos quedan en la boletería), o por calamidad del cliente, a discreción del administrador. Los rembolsos van al saldo virtual del usuario.

**Interacciones por rol y estado financiero**

El organizador tiene la capacidad de consultar las ganancias y porcentajes de ventas (globalmente, por evento o por localidad) y puede generar ofertas de descuento en ventanas de tiempo definidas por él mismo. Si el organizador compra tiquetes de su propio evento, se asume que es con cortesía (no se reflejan ingresos). **Administrador** define **porcentaje de servicio** y **cuota fija de emisión**, aprueba venues, cancela eventos, y observa **ganancias de la tiquetera** por fecha, evento u organizador

# **Objetivos**

* Diseñar una aplicación en java que controle la venta de tiquetes de conciertos, eventos culturales, religiosos y deportivos.
* Permitir la compra de tiquetes a cualquier cliente, promotor u organizador que acceda a la plataforma.
* Permitir la consulta de la información, al ser esta persistente en diferentes archivos de tipo JSON.
* Garantizar el acceso a la aplicación a los clientes, los organizadores de eventos y un administrador quién es que se encarga que todo esté en orden.
* Garantizar la seguridad de los datos del usuario en el sistema.
* Garantizar que hay un sistema para transferencia de tiquetes donde existe una restricción de la cantidad de operaciones que se pueden hacer.

# **No objetivos**

* Permitir que el administrador compre tiquetes, a pesar de que este también es usuario de la aplicación.
* Implementar un sistema de métodos de pago para la compra de tiquetes.
* Creación de eventos de cualquier tipo.

# **Justificaciones de las decisiones**

**Justificación de paquete usuario**

El paquete **usuario** contiene las clases relacionadas con la gestión y comportamiento de los diferentes tipos de usuarios dentro del sistema. Este paquete agrupa tanto la clase abstracta base Usuario, como las clases concretas Cliente, Organizador, Promotor y Administrador, además de la interfaz IDuenoTiquetes, que define el contrato común para los usuarios que poseen tiquetes.

La clase Usuario se definió como abstracta ya que representa la estructura y comportamiento general de todos los usuarios del sistema, evitando duplicación de código entre las clases hijas. Esta clase contiene los atributos básicos: *login*, *password* y *tipoUsuario*, además de métodos comunes como IsPasswordTrue, que valida credenciales, y formatear, que permite representar los datos en formato JSON, aprovechando la interoperabilidad con el paquete de Persistencia a través de la interfaz IFormateo y la clase auxiliar **TextoUtils**. De esta manera, el diseño facilita la serialización y almacenamiento de usuarios en estructuras JSON, manteniendo coherencia con el resto del sistema.

Las clases **Cliente**, **Organizador** y **Promotor** heredan de **Usuario** e implementan la interfaz **IDuenoTiquetes**, que define los métodos necesarios para administrar los tiquetes de los que son propietarios, tales como getTiquetes, agregarTiquete, eliminarTiquete y actualizarSaldo.

* La clase Cliente gestiona un *saldo* y una colección de *tiquetes*, permitiendo agregar o eliminar entradas adquiridas, además de reflejar los cambios en su saldo tras una compra o reembolso.
* La clase Organizador extiende la funcionalidad de un usuario al permitir la creación de eventos y venues, así como la generación de ofertas asociadas a estos. También incluye métodos para modificar fechas y horarios de los eventos, mostrando una clara relación con el paquete Evento.
* La clase Promotor, por su parte, se centra en la administración de ingresos y porcentajes de venta relacionados con la promoción de eventos, incorporando métodos como getGanancias y getPorcentajeVenta, que facilitan el seguimiento económico del usuario.

La clase Administrador también hereda de Usuario, pero a diferencia de las anteriores, no implementa la interfaz IDuenoTiquetes, ya que su función principal no es poseer entradas, sino gestionar el sistema. Posee atributos específicos como *porcentajeAdicional* y *cobroEmision*, que le permiten aplicar recargos o tarifas administrativas, y el método cancelarEvento, que interactúa directamente con la clase Evento, anulando tanto el evento como sus tiquetes asociados.

El diseño de este paquete promueve la herencia, la modularidad y el principio de responsabilidad única, al separar claramente las funciones de cada tipo de usuario dentro del sistema. Además, su integración con la capa de persistencia en formato JSON garantiza una representación consistente de los datos, facilitando su almacenamiento, recuperación y extensión futura, como la incorporación de nuevos tipos de usuarios o funcionalidades sin alterar la estructura base del sistema.

**Justificación paquete de evento**

El paquete Evento concentra las clases encargadas de modelar y administrar los distintos aspectos relacionados con la organización y gestión de eventos dentro del sistema. Su propósito principal es representar los escenarios donde se lleva a cabo la venta y asignación de tiquetes, estableciendo una conexión directa entre los organizadores, los lugares (venues) y las localidades disponibles para cada evento.

La clase Evento constituye el núcleo funcional del paquete. Contiene atributos como la entrada, la fecha, la hora, el conjunto de tiquetes disponibles y el venue asociado. Esta clase permite controlar el estado del evento, determinar si está cancelado, así como gestionar la oferta de tiquetes disponibles y su relación con las ventas y promociones. Su diseño facilita la interacción con otros paquetes como usuario, tiquete y finanzas, ya que los organizadores y promotores pueden crear, modificar o cancelar eventos según las políticas definidas.

Por su parte, la clase Venue representa el lugar físico o digital donde se desarrolla un evento. Contiene información relevante como la ubicación, la capacidad máxima y el estado de aprobación del recinto. Esta estructura es clave para validar la disponibilidad de espacios antes de la creación de un nuevo evento, garantizando que el sistema mantenga coherencia entre los aforos y las ventas de tiquetes.

Además, la clase Localidad describe las secciones o divisiones dentro de un venue, como platea, palco o general, asignando un precio base y un número específico de cupos para cada una. Gracias a esto, el sistema puede diferenciar precios y administrar la ocupación de forma más detallada. Este diseño modular permite que cada localidad se relacione directamente con los tiquetes y que, a su vez, pueda integrarse con las ofertas del paquete Finanzas para aplicar descuentos o ajustes dinámicos de precio.

El paquete Evento se integra de manera estrecha con la capa de persistencia en formato JSON, asegurando que los datos de cada evento, localidad y venue puedan ser almacenados y recuperados de forma estructurada. Esto facilita la gestión de información tanto para los administradores como para los usuarios finales, quienes pueden consultar la disponibilidad de eventos activos o cancelados.

En conjunto, este paquete presenta una alta cohesión al encapsular toda la lógica relacionada con la creación, configuración y control de eventos, manteniendo independencia respecto a las funciones de usuario y finanzas. De este modo, el diseño garantiza un modelo escalable, fácil de mantener y adaptable a futuras ampliaciones del sistema, como integración con calendarios, estadísticas de asistencia o gestión avanzada de reservas.

**Justificación paquete de Tiquete**

El paquete de tiquete contiene 4 clases principales: tiquete, tiquete simple, tiquete múltiple y paquete deluxe. La clase Tiquete se definió como abstracta debido a que existen diferentes tipos de tiquetes que comparten los mismos atributos y comportamientos básicos, por lo que esta estructura permite evitar duplicación de código y favorece la reutilización y extensibilidad. La clase contiene atributos como: tipo Tiquete, identificador, fecha Expiración, precio, nombre, y transferido (indica si el tiquete ha sido transferido o no). Los métodos que se implementan igual en las clases que heredan de tiquetes son: esVigente (verifica si la fecha del tiquete sigue siendo válida), isTransferido( consulta si el tiquete ya fue transferido), y setTransferido(actualiza el estado del tiquete a transferido). Mientras que los métodos que son abstractos son el de cacularPrecio y getTipoTiquete ya que estos se implementan de forma diferente en cada clase que hereda de tiquete.

Las clases Tiquete simple y tiquete múltiple heredan de la clase abstracta de tiquete donde se implementan de la forma requerida los métodos mencionados previamente. Además, la clase tiquete múltiple tiene dos atributos diferentes que son una lista de tiquetes internos y otra lista de eventos asociados.

Por otro lado, se implementa la clase de paquete deluxe que, aunque no hereda de la clase abstracta de tiquete, contiene de atributos una lista de tiquetes y otra de tiquetes adicionales. Esta relación se refleja en la multiplicidad del diagrama UML, indicando que un paquete deluxe puede contener múltiples tiquetes, pero cada tiquete pertenece únicamente a un paquete determinado.

El diseño de este paquete promueve el modularidad, la extensibilidad y el principio de reutilización de código, facilitando futuras ampliaciones, como nuevos tipos de tiquetes o características adicionales sin alterar la estructura existente.

**Justificación de la persistencia**

Para la persistencia se diseñó una interfaz principal con los tres métodos comunes a todas las clases de persistencia: crearArchivo, cargar y salvar. Adicionalmente, se creó una interfaz llamada IFormateable , la cual deben implementar todas las clases cuyos objetos serán persistentes, esto para facilitar la implementación en el método de cargar, permitiendo convertir los objetos a formato JSON de manera uniforme.

La interfaz principal de persistencia utiliza *generics* y extiende de IFormateable, lo que permite manejar distintos tipos de objetos sin duplicar código.

El método cargar archivo, como su nombre lo indica crea el archivo de la respectiva clase a la que se hará persistente su información. El segundo método carga la información al archivo creado previamente aplicando el método formateo con su respectivo tipo del objeto. Por último, el método salvar registra la información en el respectivo archivo para que sea visible en el sistema.

Por otra parte, se creó una clase estática que solo contiene el método escape que se usa en las diferentes implementaciones del método formateo.

**Justificación paquete finanzas**

El paquete Finanzas agrupa las clases encargadas de manejar todos los procesos económicos y transaccionales del sistema, incluyendo la gestión de ofertas, el control de ganancias, los registros de transacciones y la persistencia de operaciones financieras. Este paquete permite mantener la trazabilidad y coherencia entre las acciones de compra, transferencia y cálculo de beneficios económicos dentro de la aplicación.

Dentro del paquete, la clase Transaccion representa la estructura principal para la gestión de operaciones económicas relacionadas con tiquetes y usuarios. Esta clase establece la lógica para la compra, transferencia y adquisición de paquetes de tiquetes, garantizando que cada acción quede registrada y validada conforme a las reglas del negocio, como límites máximos de compra y verificación de saldo. Además, interactúa directamente con otras clases como Registro, Evento y PersistenciaTransacciones, permitiendo que cada transacción sea almacenada en formato JSON mediante la interfaz IFormateo, asegurando la persistencia y trazabilidad de las operaciones.

La clase Registro cumple la función de documentar las transferencias entre usuarios, asociando al propietario original del tiquete, el nuevo receptor y el tiquete involucrado. Este diseño favorece la auditoría de las operaciones y mantiene un control sobre la propiedad de los activos digitales dentro del sistema. Su método de formateo también utiliza la estructura JSON para garantizar la compatibilidad con el módulo de persistencia.

Por su parte, la clase EstadosFinancieros centraliza los cálculos relacionados con las ganancias, costos de producción y precios sin recargos. Esta clase proporciona herramientas para determinar el balance financiero de los eventos, ofreciendo información sobre las utilidades netas o márgenes de beneficio. Su implementación también está orientada a la persistencia en formato JSON, lo que permite almacenar y consultar fácilmente los resultados financieros generados.

La clase Oferta representa los descuentos o promociones aplicables a las ventas de tiquetes. Define atributos como el identificador, el porcentaje de descuento y las fechas de vigencia, junto con métodos que permiten validar si una oferta está activa según la fecha actual. De esta manera, el sistema puede aplicar condiciones promocionales dinámicas en las compras, fortaleciendo la flexibilidad del modelo económico.

El diseño del paquete Finanzas promueve la cohesión entre las distintas entidades del sistema al integrar la lógica financiera con los usuarios, eventos y tiquetes. Además, su estructura modular facilita la extensión del sistema hacia nuevas funcionalidades, como reportes contables o gestión de impuestos, sin alterar las clases existentes. Su enfoque en la persistencia mediante formato JSON garantiza un almacenamiento estructurado y compatible con otros componentes del sistema, fortaleciendo la integridad de los datos y la transparencia de las operaciones económicas.

# **Información sobre persistencia**

Teniendo en cuenta que la información de la aplicación debe ser persistente, para esto la información de cada paquete propuesto en la aplicación tendrá su respectivo archivo en el formato de JSON. La aplicación va a tener la capacidad de crear un documento donde se guarde la información manipulada por el usuario, de tal modo que, en el transcurso de su uso, se pueda volver a acceder a ella de forma facil.

# **Alternativa que se hayan considerado**

Al principio consideramos hacer la aplicación sin la clase de transacciones porque no lo vimos tan claro en ese momento, sin embargo, avanzando en el proyecto identificamos que la clase de transacciones era clave para el desarrollo de este. También, para la persistencia pensamos inicialmente en hacerla únicamente de los tiquetes y usuarios, pero después determinamos que la mejor opción era que cada paquete establecido en la aplicación fuera persistente, con el fin de que esta funcionara de la mejor manera.

# **Codigo de algoritmos criticos**

package Finanzas;

import usuario.IDuenoTiquetes;

import usuario.Usuario;

import java.time.LocalDateTime;

import java.util.ArrayList;

import java.util.Collection;

import java.util.Scanner;

import Evento.Evento;

import Persistencia.IFormateo;

import Persistencia.PersistenciaTransacciones;

import Persistencia.TextoUtils;

import tiquete.PaqueteDeluxe;

import tiquete.Tiquete;

import tiquete.TiqueteMultiple;

import tiquete.TiqueteSimple;

public class Transaccion implements IFormateo {

Tiquete tiquete;

Usuario dueno;

LocalDateTime fecha;

private static final double ***NUMERO\_MAX\_TRANSACCION*** = 10;

String tipoTransaccion;

double valorTransaccion;

Registro registro;

public Transaccion(String tipoTransaccion, Tiquete tiquete, Usuario dueno, LocalDateTime localDateTime, Registro registro,

double valorTransaccion) {

this.tiquete = tiquete;

this.dueno = dueno;

this.fecha = localDateTime;

this.registro = registro;

this.tipoTransaccion = tipoTransaccion;

this.valorTransaccion = valorTransaccion;

}

public Usuario getDueno() {

return this.dueno;

}

public void transferirTiquete(Tiquete tiquete, Usuario dueno, Usuario usuarioDestino, String fechaActual) {

if (dueno.getTipoUsuario().equalsIgnoreCase("ADMINISTRADOR")) {

try (Scanner sc = new Scanner(System.***in***)) {

System.***out***.print("Ingresa el login del destinatario: ");

String login = sc.nextLine();

System.***out***.print("Ingresa tu password: ");

String password = sc.nextLine();

if (usuarioDestino.getLogin() == login && dueno.IsPasswordTrue(password)) {

if (dueno instanceof IDuenoTiquetes && usuarioDestino instanceof IDuenoTiquetes) {

Collection<Tiquete> tiquetes = ((IDuenoTiquetes) dueno).getTiquetes();

TiqueteMultiple tm = (TiqueteMultiple) tiquete;

System.***out***.print("¿Desea transferir todo el paquete? (s/n): ");

String opcion = sc.nextLine();

if (opcion.equalsIgnoreCase("s")) {

boolean puedeTransferirse = true;

for (TiqueteSimple t : tm.getTiquetes()) {

if (!t.esVigente(fechaActual) || t.isTransferido()) {

puedeTransferirse = false;

break;

}

}

if(puedeTransferirse) {

((IDuenoTiquetes) dueno).eliminarTiquete(tm);

((IDuenoTiquetes) usuarioDestino).agregarTiquete(tm);

tm.setTransferido(true);

for (Tiquete t : tm.getTiquetes()) {

t.setTransferido(true);

}

System.***out***.println("Paquete completo transferido con éxito.");

} else {

System.***out***.println("No se puede transferir el paquete: contiene tiquetes vencidos o ya transferidos.");

}

}else {

System.***out***.print("Ingrese el ID del tiquete individual dentro del paquete: ");

String idTiquete = sc.nextLine();

boolean encontrado = false;

for (TiqueteSimple t : tm.getTiquetes()) {

if (t.getId().equals(idTiquete)) {

encontrado = true;

if (!t.esVigente(fechaActual)) {

System.***out***.println("ERROR: el tiquete está vencido.");

} else if (t.isTransferido()) {

System.***out***.println("ERROR: el tiquete ya fue transferido anteriormente.");

} else {

t.setTransferido(true);

((IDuenoTiquetes) usuarioDestino).agregarTiquete(t);

((IDuenoTiquetes) dueno).eliminarTiquete(t);

System.***out***.println("Tiquete individual transferido con éxito.");

}

break;

}

}

if (!encontrado) {

System.***out***.println("ERROR: el ID no corresponde a ningún tiquete dentro del paquete.");

}

}

System.***out***.print("Ingrese el id del tiquete a transferir: ");

String idTiquete = sc.nextLine();

for(Tiquete t: tiquetes) {

if(idTiquete.equals(t.getId())) {

((IDuenoTiquetes) dueno).eliminarTiquete(tiquete);

((IDuenoTiquetes) usuarioDestino).agregarTiquete(tiquete);

tiquete.setTransferido(true);

}else {

System.***out***.print("ERROR: el id escrito no es válido");

}

}

}

}else {

System.***out***.print("ERROR: login o password incorrecto");

}

}

}

Registro registro = new Registro(dueno, tiquete, usuarioDestino);

Transaccion trans = new Transaccion("TRANSFERENCIA", tiquete, dueno, LocalDateTime.*now*(), registro, 0);

PersistenciaTransacciones persist = new PersistenciaTransacciones();

persist.cargar(trans);

}

public ArrayList<Tiquete> comprarTiquete(Tiquete tiqueteComprar, Usuario comprador,int cantidad, Evento eventoAsociado) {

double precioTotal = 0;

ArrayList<Tiquete> tiquetesComprados = new ArrayList<>();

Collection<Tiquete> tiquetes = eventoAsociado.getTiquetesDisponibles();

if (tiquetes.isEmpty()) {

System.***out***.println("ERROR: No hay tiquetes disponibles.");

return tiquetesComprados;

}

if (cantidad > ***NUMERO\_MAX\_TRANSACCION***) {

System.***out***.println("ERROR: Supera el número máximo de tiquetes por transacción.");

return tiquetesComprados;

}

if(comprador instanceof IDuenoTiquetes) {

double saldo =((IDuenoTiquetes) comprador).getSaldo();

if (tiqueteComprar instanceof TiqueteMultiple) {

precioTotal = tiqueteComprar.calcularPrecio();

} else {

precioTotal = tiqueteComprar.calcularPrecio() \* cantidad;

}

if(precioTotal<= saldo ) {

saldo -= precioTotal;

((IDuenoTiquetes) comprador).actualizarSaldo(saldo);

}else {

System.***out***.print("ERROR: Saldo insuficiente");

return tiquetesComprados;

}

if (tiqueteComprar instanceof TiqueteMultiple) {

TiqueteMultiple tm = (TiqueteMultiple) tiqueteComprar;

for (Tiquete t : tm.getTiquetes()) {

((IDuenoTiquetes) comprador).agregarTiquete(t);

eventoAsociado.venderTiquete(t.getId());

tiquetesComprados.add(t);

}

}else {

for (int i = 0; i < cantidad; i++) {

((IDuenoTiquetes) comprador).agregarTiquete(tiqueteComprar);

eventoAsociado.venderTiquete(tiqueteComprar.getId());

tiquetesComprados.add(tiqueteComprar);

}

}

}

Registro registro = new Registro(comprador, tiqueteComprar, null);

Transaccion trans = new Transaccion("TRANSFERENCIA", tiquete, dueno, LocalDateTime.*now*(), registro, precioTotal);

PersistenciaTransacciones persist = new PersistenciaTransacciones();

persist.cargar(trans);

return tiquetesComprados;

}

public ArrayList<Tiquete> comprarPaqueteDeluxe(PaqueteDeluxe paquete,

Usuario comprador, int cantidad, Evento eventoAsociado ) {

ArrayList<Tiquete> tiquetesComprados = new ArrayList<>();

if (paquete == null || comprador == null || eventoAsociado == null) {

System.***out***.println("ERROR: Datos incompletos para comprar paquete Deluxe.");

return tiquetesComprados;

}

if (cantidad <= 0) {

System.***out***.println("ERROR: La cantidad debe ser mayor que cero.");

return tiquetesComprados;

}

if (cantidad > ***NUMERO\_MAX\_TRANSACCION***) {

System.***out***.println("ERROR: Supera el número máximo de paquetes por transacción.");

return tiquetesComprados;

}

if (cantidad != 1) {

System.***out***.println("ERROR: Por ahora solo se soporta comprar 1 Paquete Deluxe por transacción con el paquete recibido.");

return tiquetesComprados;

}

ArrayList<Tiquete> tiqsPaquete = new ArrayList<>();

if (paquete.getTiquetes() != null) {

for (Tiquete t : paquete.getTiquetes()) {

if (t != null) {

tiqsPaquete.add(t);

}

}

}

if (paquete.getTiquetesAdicionales() != null) {

for (Tiquete t : paquete.getTiquetesAdicionales()) {

if (t != null) {

tiqsPaquete.add(t);

}

}

}

if (tiqsPaquete.isEmpty()) {

System.***out***.println("ERROR: El paquete Deluxe no contiene tiquetes.");

return tiquetesComprados;

}

for (Tiquete t : tiqsPaquete) {

Tiquete enEvento = eventoAsociado.getTiquetePorId(t.getId());

if (enEvento == null) {

System.***out***.println("ERROR: Tiquete no disponible en el evento: " + t.getId());

return tiquetesComprados;

}

}

double total = paquete.getPrecioPaquete();

boolean esCortesiaOrganizador = "ORGANIZADOR".equalsIgnoreCase(comprador.getTipoUsuario());

if (esCortesiaOrganizador) {

total = 0.0;

}

if (comprador instanceof IDuenoTiquetes) {

IDuenoTiquetes dueno = (IDuenoTiquetes) comprador;

double saldo = dueno.getSaldo();

if (total > saldo) {

System.***out***.println("ERROR: Saldo insuficiente para comprar el paquete Deluxe.");

return tiquetesComprados;

}

if (total > 0) {

saldo -= total;

dueno.actualizarSaldo(saldo);

}

} else {

System.***out***.println("ERROR: El comprador no puede poseer tiquetes (no implementa IDuenoTiquetes).");

return tiquetesComprados;

}

for (Tiquete t : tiqsPaquete) {

t.setTransferido(true);

}

IDuenoTiquetes dueno = (IDuenoTiquetes) comprador;

for (Tiquete t : tiqsPaquete) {

dueno.agregarTiquete(t);

eventoAsociado.venderTiquete(t.getId());

tiquetesComprados.add(t);

}

return tiquetesComprados;

}

public Tiquete getTiquete() {

return this.tiquete;

}

public Usuario getReceptor(Usuario UsuarioReceptor) {

return UsuarioReceptor;

}

*@Override*

public String formatear() {

return String.*format*(

"{\n \"tipoTransaccion\": \"%s\",\n \"valor\": %.2f,\n \"fecha\": \"%s\",\n \"registro\": %s\n}",

TextoUtils.*escape*(this.tipoTransaccion),

TextoUtils.*escape*(this.valorTransaccion),

TextoUtils.*escape*(this.fecha),

this.registro.formatear()

);

}

}

* La persistencia

**Persistencia estados financieros**

package Persistencia;

import java.io.BufferedWriter;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import Finanzas.EstadosFinancieros;

public class PersistenciaEstadosFinan implements IPersistencia<EstadosFinancieros> {

private static final String ***RUTA*** = "data/estadosFinancieros.json";

*@Override*

public void crearArchivo() {

try {

Files.*createDirectories*(Path.*of*("data"));

} catch (IOException e) {

System.***out***.println("Error al crear carpeta data: " + e.getMessage());

}

}

*@Override*

public void cargar(EstadosFinancieros newObjeto) {

String format = newObjeto.formatear();

String info = "";

try {

if (Files.*exists*(Path.*of*(***RUTA***))) {

info = Files.*readString*(Path.*of*(***RUTA***));

}

} catch (IOException e) {

System.***out***.println("ERROR: no se puede leer el archivo");

}

String newJson;

if (info.isBlank()) {

newJson = "[\n" + format + "\n]";

} else {

info = info.trim();

if (info.endsWith("]")) {

info = info.substring(0, info.length() - 1);

if (info.contains("{")) {

newJson = info +",\n" + format + "\n]";

} else {

newJson = "[\n" + format + "\n]";

}

} else {

newJson = "[\n" + format + "\n]";

}

}

salvar(newJson);

}

*@Override*

public void salvar(String jsonFormatted) {

try (BufferedWriter writer = Files.*newBufferedWriter*(Path.*of*(***RUTA***))) {

writer.write(jsonFormatted);

System.***out***.println("\n Tiquete registrado correctamente en " + ***RUTA***);

} catch (IOException e) {

System.***out***.println(" Error al guardar el archivo: " + e.getMessage());

}

}

}

**Persistencia Tiquete**

package Persistencia;

import java.io.BufferedWriter;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import tiquete.Tiquete;

public class PersistenciaTiquetesJson implements IPersistencia<Tiquete> {

private static final String ***RUTA*** = "data/Tiquetes.json";

*@Override*

public void crearArchivo() {

try {

Files.*createDirectories*(Path.*of*("data"));

} catch (IOException e) {

System.***out***.println("Error al crear carpeta data: " + e.getMessage());

}

}

*@Override*

public void cargar(Tiquete newObjeto) {

String format = newObjeto.formatear();

String info = "";

try {

if (Files.*exists*(Path.*of*(***RUTA***))) {

info = Files.*readString*(Path.*of*(***RUTA***));

}

} catch (IOException e) {

System.***out***.println("ERROR: no se puede leer el archivo");

}

String newJson;

if (info.isBlank()) {

newJson = "[\n" + format + "\n]";

} else {

info = info.trim();

if (info.endsWith("]")) {

info = info.substring(0, info.length() - 1);

if (info.contains("{")) {

newJson = info +",\n" + format + "\n]";

} else {

newJson = "[\n" + format + "\n]";

}

} else {

newJson = "[\n" + format + "\n]";

}

}

salvar(newJson);

}

*@Override*

public void salvar(String jsonFormatted) {

try (BufferedWriter writer = Files.*newBufferedWriter*(Path.*of*(***RUTA***))) {

writer.write(jsonFormatted);

System.***out***.println("\n Tiquete registrado correctamente en " + ***RUTA***);

} catch (IOException e) {

System.***out***.println(" Error al guardar el archivo: " + e.getMessage());

}

}

}

**Persistencia Eventos**

package Persistencia;

import java.io.BufferedWriter;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import Evento.Evento;

public class PersistenciaEventos implements IPersistencia<Evento> {

private static final String ***RUTA*** = "data/Tiquetes.json";

*@Override*

public void crearArchivo() {

try {

Files.*createDirectories*(Path.*of*("data"));

} catch (IOException e) {

System.***out***.println("Error al crear carpeta data: " + e.getMessage());

}

}

*@Override*

public void cargar(Evento newObjeto) {

String format = newObjeto.formatear();

String info = "";

try {

if (Files.*exists*(Path.*of*(***RUTA***))) {

info = Files.*readString*(Path.*of*(***RUTA***));

}

} catch (IOException e) {

System.***out***.println("ERROR: no se puede leer el archivo");

}

String newJson;

if (info.isBlank()) {

newJson = "[\n" + format + "\n]";

} else {

info = info.trim();

if (info.endsWith("]")) {

info = info.substring(0, info.length() - 1);

if (info.contains("{")) {

newJson = info +",\n" + format + "\n]";

} else {

newJson = "[\n" + format + "\n]";

}

} else {

newJson = "[\n" + format + "\n]";

}

}

salvar(newJson);

}

*@Override*

public void salvar(String jsonFormatted) {

try (BufferedWriter writer = Files.*newBufferedWriter*(Path.*of*(***RUTA***))) {

writer.write(jsonFormatted);

System.***out***.println("\n Tiquete registrado correctamente en " + ***RUTA***);

} catch (IOException e) {

System.***out***.println(" Error al guardar el archivo: " + e.getMessage());

}

}

}

**Persistencia Transacciones**

package Persistencia;

import java.io.BufferedWriter;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import Finanzas.Transaccion;

public class PersistenciaTransacciones implements IPersistencia<Transaccion> {

private static final String ***RUTA*** = "data/transferencias.json";

*@Override*

public void crearArchivo() {

// **TODO** Auto-generated method stub

try {

Files.*createDirectories*(Path.*of*("data"));

} catch (IOException e) {

System.***out***.println("Error al crear carpeta data: " + e.getMessage());

}

}

*@Override*

public void cargar(Transaccion newTransaccion) {

// **TODO** Auto-generated method stub

String format = newTransaccion.formatear();

String info = "";

try {

if (Files.*exists*(Path.*of*(***RUTA***))) {

info = Files.*readString*(Path.*of*(***RUTA***));

}

} catch (IOException e) {

System.***out***.println("ERROR: no se puede leer el archivo");

}

String newJson;

if (info.isBlank()) {

newJson = "[\n" + format + "\n]";

} else {

info = info.trim();

if (info.endsWith("]")) {

info = info.substring(0, info.length() - 1);

if (info.contains("{")) {

newJson = info +",\n" + format + "\n]";

} else {

newJson = "[\n" + format + "\n]";

}

} else {

newJson = "[\n" + format + "\n]";

}

}

salvar(newJson);

}

*@Override*

public void salvar(String jsonFormatted) {

try (BufferedWriter writer = Files.*newBufferedWriter*(Path.*of*(***RUTA***))) {

writer.write(jsonFormatted);

System.***out***.println("\n Usuario registrado correctamente en " + ***RUTA***);

} catch (IOException e) {

System.***out***.println(" Error al guardar el archivo: " + e.getMessage());

}

}

}

**Persistencia usuarios**

package Persistencia;

import java.io.\*;

import java.nio.file.\*;

import usuario.Usuario;

public class PersistenciaUsuariosJson implements IPersistencia<Usuario> {

private static final String ***RUTA*** = "data/usuarios.json";

*@Override*

public void crearArchivo() {

try {

Files.*createDirectories*(Path.*of*("data"));

} catch (IOException e) {

System.***out***.println("Error al crear carpeta data: " + e.getMessage());

}

}

public void cargar(Usuario newUsuario) {

String format = newUsuario.formatear();

String info = "";

try {

if (Files.*exists*(Path.*of*(***RUTA***))) {

info = Files.*readString*(Path.*of*(***RUTA***));

}

} catch (IOException e) {

System.***out***.println("ERROR: no se puede leer el archivo");

}

String newJson;

if (info.isBlank()) {

newJson = "[\n" + format + "\n]";

} else {

info = info.trim();

if (info.endsWith("]")) {

info = info.substring(0, info.length() - 1);

if (info.contains("{")) {

newJson = info +",\n" + format + "\n]";

} else {

newJson = "[\n" + format + "\n]";

}

} else {

newJson = "[\n" + format + "\n]";

}

}

salvar(newJson);

}

*@Override*

public void salvar(String jsonFormatted) {

try (BufferedWriter writer = Files.*newBufferedWriter*(Path.*of*(***RUTA***))) {

writer.write(jsonFormatted);

System.***out***.println("\n Usuario registrado correctamente en " + ***RUTA***);

} catch (IOException e) {

System.***out***.println(" Error al guardar el archivo: " + e.getMessage());

}

}

}

* La clase Tiquete

package tiquete;

import java.time.LocalDate;

import Persistencia.IFormateo;

import Persistencia.TextoUtils;

public abstract class Tiquete implements IFormateo {

private String tipoTiquete;

protected double cargoPorcentual;

protected double cuotaAdicional;

private String identificador;

private String fechaExpiracion;

protected double precio;

private String nombre;

private boolean transferido;

private boolean anulado;

public Tiquete(String tipoTiquete, double cargoPorcentual, double cuotaAdicional, String identificador, String fechaExpiracion,

int precio, String nombre, boolean transferido, boolean anulado){

this.tipoTiquete = tipoTiquete;

this.cargoPorcentual = cargoPorcentual;

this.cuotaAdicional = cuotaAdicional;

this.identificador = identificador;

this.fechaExpiracion = fechaExpiracion;

this.precio = precio;

this.nombre = nombre;

this.transferido = true;

this.anulado = false;

}

public boolean isAnulado() {

return anulado;

}

public void setAnulado(boolean anulado) {

this.anulado = anulado;

}

public abstract String getTipoTiquete();

public String getId() {

return this.identificador;

}

public String getNombre() {

return this.nombre;

}

public String getFechaExpiracion() {

return this.fechaExpiracion;

}

public boolean esVigente(String fechaActual) {

LocalDate fechaExp = LocalDate.*parse*(getFechaExpiracion());

LocalDate fechaAct = LocalDate.*parse*(fechaActual);

return!fechaExp.isBefore(fechaAct);

}

public abstract double calcularPrecio();

public boolean isTransferido() {

return transferido;

}

public void setTransferido(boolean isTransferido) {

this.transferido = isTransferido;

}

public String formatear() {

String formatJson = String.*format*(" {\n \"nombre\": \"%s\",\n \"Id\": \"%s\",\n \"FechaExpiracion\": \"%s\"\n "

+ "\\\"precio\\\": \\\"%s\\\",\\n \\\"TipoTiquete\\\": \\\"%s\\\",\\n \\\"Transferido\\\": \\\"%s\\\",\\n }"

, TextoUtils.*escape*(this.nombre), TextoUtils.*escape*(this.identificador), TextoUtils.*escape*(this.fechaExpiracion),

TextoUtils.*escape*(this.precio), TextoUtils.*escape*(this.tipoTiquete),TextoUtils.*escape*(this.transferido));

return formatJson;

}

}

**Clase administrador**

package usuario;

import java.util.ArrayList;

import Evento.Evento;

import tiquete.Tiquete;

public class Administrador extends Usuario {

private double porcentajeAdicional;

private double cobroEmision;

public Administrador(String login, String password, String tipoUsuario) {

super(login, password, tipoUsuario);

this.porcentajeAdicional = 0.0;

this.cobroEmision = 0.0;

}

public void cobrarPorcentajeAdicional(double porcentaje) {

if (porcentaje < 0) {

System.***out***.println("El porcentaje adicional no puede ser negativo.");

} else {

this.porcentajeAdicional = porcentaje;

System.***out***.println("Se ha establecido un porcentaje adicional de " + porcentaje + "%.");

}

}

public void fijarCobroEmisionImpresion(double cobroEmision) {

if (cobroEmision < 0) {

System.***out***.println("El cobro de emisión no puede ser negativo.");

} else {

this.cobroEmision = cobroEmision;

System.***out***.println("Cobro de emisión/impresión fijado en $" + cobroEmision);

}

}

public void cancelarEvento(Evento evento) {

if (evento == null) {

System.***out***.println("El evento no existe o es nulo.");

return;

}

evento.setCancelado(true);

ArrayList<Tiquete> tiquetes = new ArrayList<>(evento.getTiquetesDisponibles());

for (Tiquete t : tiquetes) {

t.setAnulado(true);

}

System.***out***.println("El evento '" + evento.getEntrada() + "' ha sido cancelado junto con sus tiquetes.");

}

public double getPorcentajeAdicional() {

return porcentajeAdicional;

}

public double getCobroEmision() {

return cobroEmision;

}

*@Override*

public String getTipoUsuario() {

return "ADMINISTRADOR";

}

}

**Clase Cliente**

package usuario;

import java.util.ArrayList;

import java.util.Collection;

import tiquete.Tiquete;

public class Cliente extends Usuario implements IDuenoTiquetes {

private double saldo;

private ArrayList<Tiquete> tiquetes;

public Cliente(String login, String password, double saldo, String tipoUsuario) {

super(login, password, tipoUsuario);

this.saldo = saldo;

this.tiquetes = new ArrayList<Tiquete>();

}

public double getSaldo() {

return this.saldo;

}

public Collection<Tiquete> getTiquetes() {

return this.tiquetes;

}

public void agregarTiquete(Tiquete tiquete) {

tiquetes.add(tiquete);

}

public void eliminarTiquete(Tiquete tiquete) {

tiquetes.remove(tiquete);

}

*@Override*

public String getTipoUsuario() {

return "CLIENTE";

}

*@Override*

public void actualizarSaldo(double precioNuevo) {

this.saldo = precioNuevo;

}

}

**Clase organizador**

package usuario;

import java.util.ArrayList;

import java.util.Collection;

import java.util.HashMap;

import Evento.Evento;

import Evento.Venue;

import Finanzas.Oferta;

import tiquete.Tiquete;

public class Organizador extends Usuario implements IDuenoTiquetes {

private double saldo;

private ArrayList<Tiquete> tiquetes;

public Organizador(String login, String password, double saldo, String tipoUsuario) {

super(login, password, tipoUsuario);

this.saldo = saldo;

}

public Collection<Tiquete> getTiquetes() {

return this.tiquetes;

}

public void agregarTiquete(Tiquete tiquete) {

tiquetes.add(tiquete);

}

public void eliminarTiquete(Tiquete tiquete) {

tiquetes.remove(tiquete);

}

public double getSaldo() {

return this.saldo;

}

public Evento crearEvento(String Entrada, String fecha, String hora,

HashMap<String, Tiquete> tiquetesDisponibles, Venue venueAsociado) {

Evento newEvento = new Evento(Entrada, fecha, hora, tiquetesDisponibles,venueAsociado);

return newEvento;

}

public Venue crearVenue(String ubicacion, int capacidadMax, boolean aprobado) {

return new Venue(ubicacion, capacidadMax, aprobado);

}

public void definirHoraEvento(Evento evento, String nuevaHora) {

evento.setHora(nuevaHora);

}

public void definirFechaEvento(Evento evento, String nuevaFecha) {

evento.setFecha(nuevaFecha);

}

public Oferta generarOferta(int id, double porcentaje, String fechaInicio,

String fechaFinal) {

Oferta newOferta = new Oferta(id, porcentaje, fechaInicio,

fechaFinal);

return newOferta;

}

*@Override*

public String getTipoUsuario() {

// **TODO** Auto-generated method stub

return "ORGANIZADOR";

}

*@Override*

public void actualizarSaldo(double precioNuevo) {

this.saldo = precioNuevo;

}

}

**Clase promotor**

package usuario;

import java.util.ArrayList;

import java.util.Collection;

import tiquete.Tiquete;

public class Promotor extends Usuario implements IDuenoTiquetes {

private double saldo;

private ArrayList<Tiquete> tiquetes;

public Promotor(String login, String password, double saldo, String tipoUsuario) {

super(login, password, tipoUsuario);

this.saldo = saldo;

}

public double getSaldo() {

return this.saldo;

}

public double getGanancias() {

return saldo;

}

public double getPrecio() {

return saldo;

}

public double getPorcentajeVenta() {

return saldo;

}

public double getPorcentajeVentaEvento() {

return saldo;

}

public double getPorcentajeVentaLocalidad(String identificadorEvento, String nombreLocalidad) {

return saldo;

}

public Collection<Tiquete> getTiquetes() {

return this.tiquetes;

}

public void agregarTiquete(Tiquete tiquete) {

tiquetes.add(tiquete);

}

public void eliminarTiquete(Tiquete tiquete) {

tiquetes.remove(tiquete);

}

*@Override*

public String getTipoUsuario() {

return "PROMOTOR";

}

*@Override*

public void actualizarSaldo(double precioNuevo) {

this.saldo = precioNuevo;

}

}

**Clase Usuario**

package usuario;

import Persistencia.IFormateo;

import Persistencia.TextoUtils;

import tiquete.Tiquete;

public abstract class Usuario implements IFormateo {

private String login;

private String password;

private String tipoUsuario;

public Usuario (String login, String password, String tipoUsuario) {

this.login = login;

this.password = password;

this.tipoUsuario = tipoUsuario;

}

public String getLogin() {

return this.login;

}

public boolean IsPasswordTrue(String passwordTry) {

if (passwordTry == this.password) {

return true;

}

else {

return false;

}

}

public void solicitarRembolso(Tiquete tiquete) {

}

public abstract String getTipoUsuario();

public String formatear() {

String formatJson = String.*format*(" {\n \"login\": \"%s\",\n \"password\": \"%s\",\n \"tipo\": \"%s\"\n }"

, TextoUtils.*escape*(this.login), TextoUtils.*escape*(this.password), TextoUtils.*escape*(this.tipoUsuario));

return formatJson;

}

}

# **Preocupaciones transversales**

A pesar de que queremos garantizar una funcionalidad eficiente para la aplicación de *Boletamaster*, hay ciertas preocupaciones que se generan a la hora de pasar a la fase de implementación. Primeramente, está el aspecto de la seguridad. Como aún no se han tenido en cuenta algoritmos para encriptar información sensible del usuario, esto es un aspecto importante a tener en cuenta para tener un programa que sea sostenible. Otro elemento indispensable es la transaccionalidad. Dicha característica genera intranquilidad sobre el comportamiento que puede generarse a la hora de comprar tiquetes. Adicionalmente a esto, la base de este programa es la persistencia, y una gran inquietud es que tanto la carga de datos, como guardarlos en un archivo, ya que, si se cargan incorrectamente, la información se puede ver comprometida y alterada para los diferentes tipos de usuarios que puedan necesitar diferentes funcionalidades.