ENTREGA UNICA PROYECTO 3

Diagramas de clases

1. **UML.**
2. **Diagrama de Clases de alto nivel (logica) .**
   Se encuentra en el arhivo adjunto en el repositorio y está titulado: Diagrama de Clases de alto nivel (logica).
3. **Diagrama de Clases de alto nivel (Ingerfaces Graficas) .**

Cambios Realizados, Alternativas y correcciones consideradas durante el proceso del trabajo

1. Para realizar la persistencia, consideramos usar la librería externa json.jar ya que nos facilitaba el desarrollo del trabajo al ofrecernos herramientas ya implementadas para almacenar archivos, sin embargo, tuvimos problemas al momento de instalarla en eclipse.
2. Durante el proceso de diseño, tuvimos problemas decidiendo a que clases les aplicaríamos la persistencia.
3. Se cambió la persistencia, de tenerla repartida por las diferentes clases importantes a tener su propia clase y que realice desde ahí la persistencia de las clases importantes (Atraccion, Espectaculo, Tiquete).
4. Se eliminaron las clases de Cafeteria, Taquilla y Restaurante, y se dejó solo LugarServicio que indica el lugar de servicio con un string.
5. Se eliminó también ExcLevel(nivel de exclusividad) en Tiquete, se cambió por un atributo String al tener problemas con el método comprarTiquete y con la Persistencia, solo usandosé en Atraccion y se planea eliminarlo del todo en el futuro.
6. Se crearon pruebas para los métodos más importantes como los metodos de Usuario, consolaAdministrador, Empleado y Atraccion.
7. Las interfaces no se lograron implementar debido a que cliente y empleado se diferencian por el booleano que los hace diferente y tambien administrador no existe como tal sino una clase independiente llamada consolaAdministrador.
8. En la entrega, agregamos las interfaces graficas requeridas tanto en el UML como en la implementación. A pesar de que, en la entrega pasada, no se hizo la implementación de las interfaces (las no graficas), porque no se vio necesaria, en este de la misma manera no se implementó debido a que ya tenemos las interfaces graficas. Porque al momento de contar con las gráficas no se haría uso, de las no graficas, entonces serian nada más saturación de código.
9. Las interfaces graficas se incluyeron en paquetes que se llaman loginInterfaz , administradorInterfaz, usuarioInterfaz y tiqueteInterfaz. Cada uno de estos paquetes tienes 4 clases, cada uno con ventana, panal inferior, panel superior y centro. Se agrego a su vez una carpeta con las imágenes que se pondran en las

interfaces, tenemos que en el panel superior de cada una se tiene una imagen como indicativo visual que indica la bienvenida a cada tipo de usuario, a excepción de la clase tiqueteInterfaz la cual es nada más un panel mostrando la imagen del tiquete.

10. Para la mayoria de las interfaces el panel inferior es para hacer el cierre de sesion.

Contexto y alcance:

Se está buscando diseñar un sistema integral que sea capaz de gestionar interacciones y ventas de boletería de un parque de diversiones con el uso de un catálogo de atracciones y espectáculos.

Los 4 principales objetivos que va a tener nuestro sistema son:

- Tendrá un catálogo de atracciones y espectáculos que se pueda presentar a los usuarios (Clientes, empleados y administrador).
- Gestión de empleados con relación a sus turnos, los roles y las capacitaciones que requiere cada atracción y que tiene cada empleado.
- Debe ser capaz de vender tiquetes de manera de forma virtual y en la taquilla (Debe haber una variedad de modalidades y opciones de tiquetes).
- Guardar y mostrar la información más importante de la aplicación (Persistencia)

Los principales Stakeholders del contexto son:

- Administradores (gestionan el parque y la plataforma).
- Empleados (Consultan los turnos, venden tiquetes a los clientes y operan las atracciones teniendo en cuenta los turnos consultados).
- Clientes (Compran tiquetes y consultan el catálogo de Atracciones/Espectáculos)

Requerimientos del código:

- Lenguaje Java.
- Toda la información debe ser persistente.
- La información debe almacenarse en archivos (Planos/Binarios).
- Solo la aplicación va a leer los archivos de la carpeta donde se guarde la información.
- La carpeta no puede ser la misma carpeta donde se encuentre el código fuente de la aplicación.
- La persistencia no necesariamente debe hacerse en un solo archivo: diseñe con cuidado cuántos archivos habrá y cómo van a estar estructurados.
- Todos los usuarios del sistema deben tener un login y una contraseña para poder acceder al sistema.
- Se pueden añadir funcionalidades extra que faciliten el trabajo (UML e implementación).

Alcance

Se planteó construir 10 paquetes importantes para la solución del problema**:** Empleados, ParqueDeAtracciones, Persistencias, Tiquetes, tests, UsuarioInterfaz, AdministradorInterfaz, EmpleadoInterfaz, LoginInterfaz y TiqueteInterfaz.

Cada paquete cuenta con clases y responsabilidades únicas:

1. ParqueDeAtracciones:

   Cuenta con las siguientes responsabilidades:

- Registrar y editar atracciones o espectáculos con sus respectivas características y restricciones de uso.
- Definir para cada atracción que restricciones tiene y que modalidad de tiquete permite su uso.
- Disponibilidad según época del año, clima y mantenimiento.

   Y las clases que tiene son:

- Atraccion (cultural o mecanica)
- ConsolaAdministrador
- Espectaculo
- Parque
- Usuario
- riskLevel
- ExcLevel

Como se muestra en la figura 1:

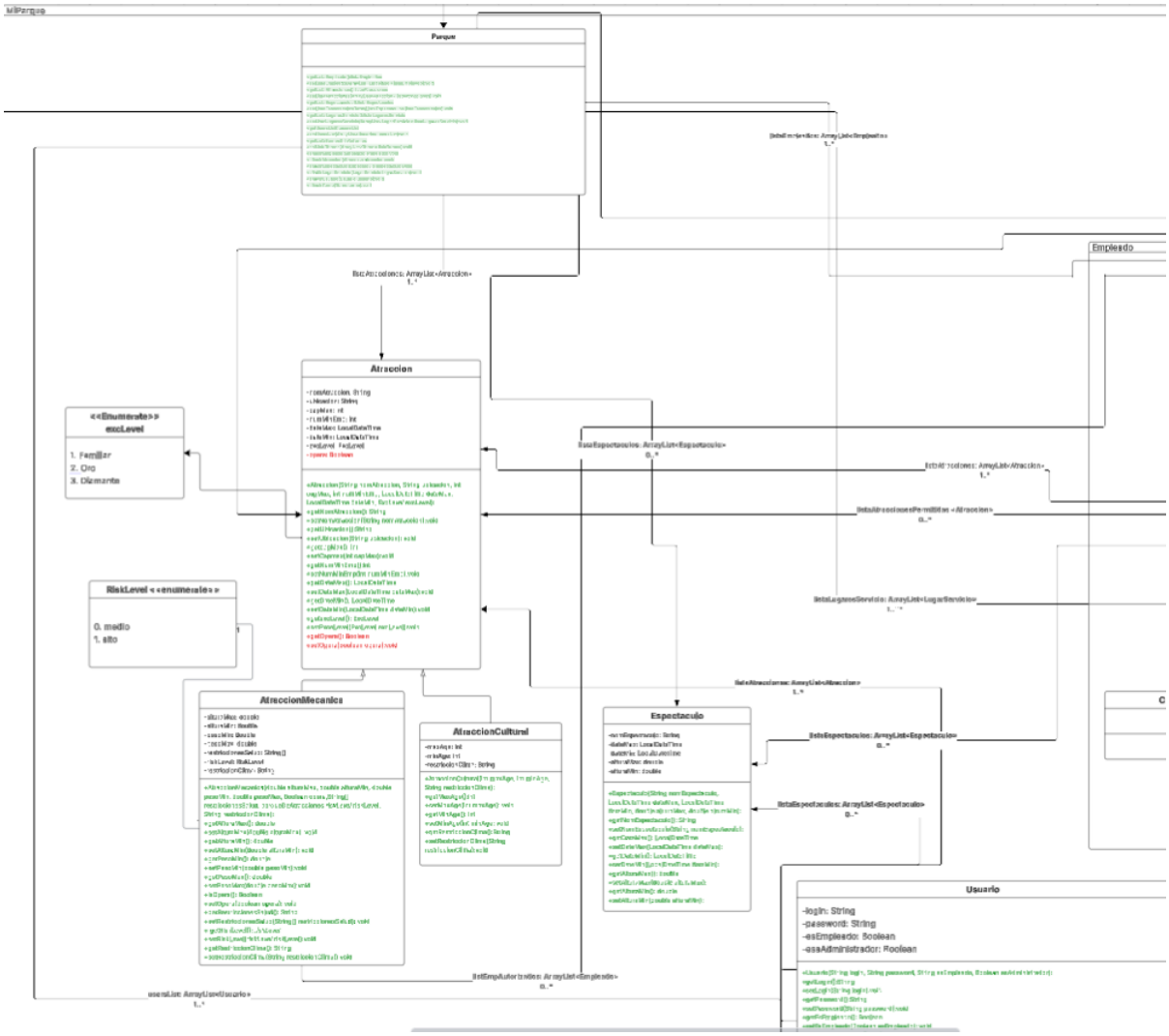


Figura 1

2. Empleados

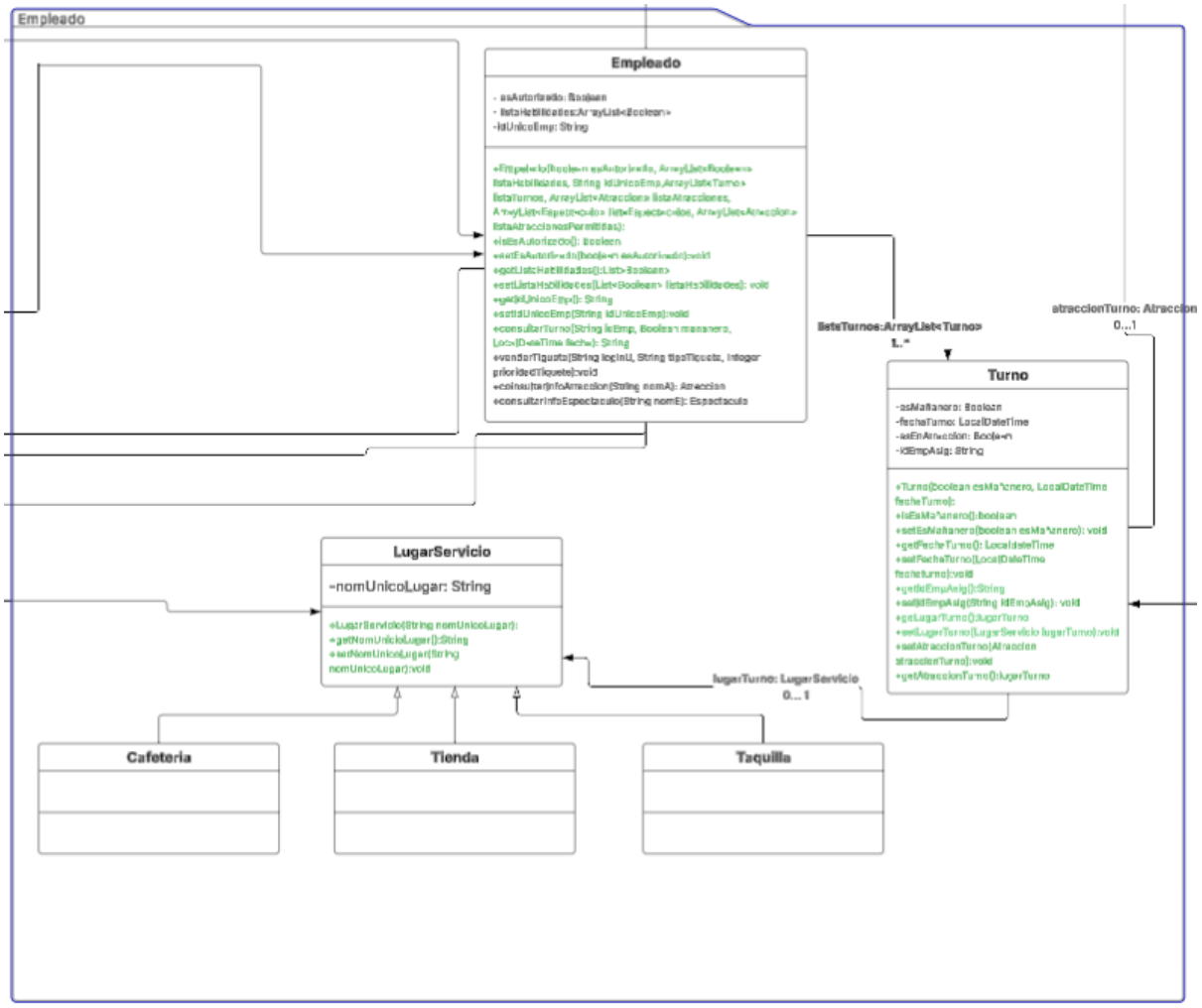
Cuenta con las siguientes responsabilidades:

- Registrar y editar atracciones o espectáculos con sus respectivas características y restricciones de uso.
- Definir para cada atracción que restricciones tiene y que modalidad de tiquete permite su uso.
- Disponibilidad según época del año, clima y mantenimiento.

Y las clases que tiene son:

- Empleado
- LugarServicio
- Turno

Como se puede ver en la figura 2:



3. Tiquetes:

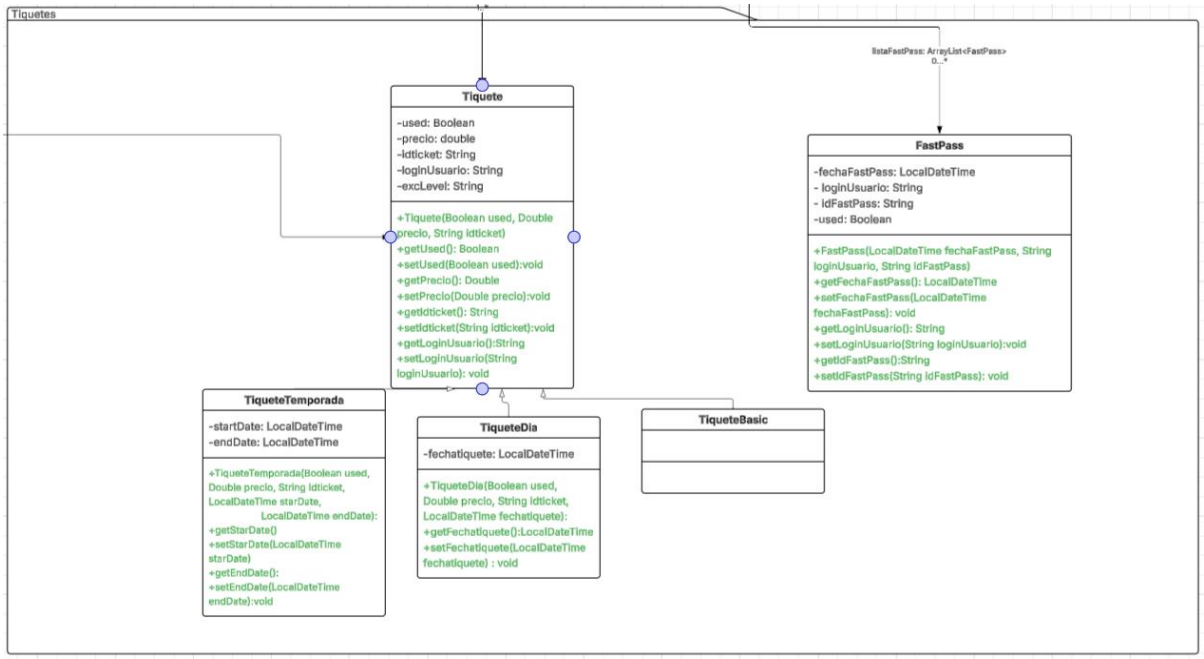
Cuenta con las siguientes responsabilidades:

- Venta en línea y en la taquilla (con descuento para empleados).
- Tipo de tiquete (Generales: Básico, familiar, oro, diamante/Temporada: semanal, mensual, anual/Individual atracción/FastPass).
- Generación de un código QR para hacer uso de las atracciones.

Y contiene las siguientes clases:

- FastPass
- Tiquete
- TiqueteDia
- TiqueteTemporada

Como se puede ver en la figura 3:



4. Persistencia (ahora como clase):

Cuenta con las siguientes responsabilidades:

- Archivos estructurados para almacenar la información (Json, Dat, etc)
- Carpeta con información fuera del código fuente.

Que solo cuenta con una clase:

- Persistencia

5. Tests:

Cuenta con las siguientes responsabilidades:

- Venta en línea y en la taquilla (con descuento para empleados).
- Tipo de tiquete (Generales: Básico, familiar, oro, diamante/Temporada: semanal, mensual, anual/Individual atracción/FastPass).
- Generación de un código QR para hacer uso de las atracciones.

Que cuenta con las clases para realizar los tests:

- EmpleadoTest
- TestConsolaAdministrador
- UsuarioTest

6. Empleadointerfaz:

Cuenta con las siguientes responsabilidades:

- Hacer que el tipo usuario empleado interactue con la aplicación

Que cuenta con 4 clases:

- PanelEmpleadoSuperior
- PanelEmpleadoCentro
- PanelEmpleadoInferior
- VentanaEmpleado

7. UsuarioInterfaz

Cuenta con las siguientes responsabilidades:

- Hacer que el tipo usuario Usuario interactue con la aplicación

Que cuenta con 4 clases:

- PanelUsuarioSuperior
- PanelUsuarioCentro
- PanelUsuarioInferior
- VentanaUsuario

8. LoginInterfaz

Cuenta con las siguientes responsabilidades:

- Hacer que el tipo usuario Usuario interactue con la aplicación

Que cuenta con 4 clases:

- PanelUsuarioSuperior
- PanelUsuarioCentro
- PanelUsuarioInferior
- VentanaUsuario

9. AdministradorInterfaz

Cuenta con las siguientes responsabilidades:

- Hacer que el tipo usuario Usuario interactue con la aplicación

Que cuenta con 4 clases:

- PanelUsuarioSuperior
- PanelUsuarioCentro
- PanelUsuarioInferior
- VentanaUsuario

10. TiqueteInterfaz

Cuenta con las siguientes responsabilidades:

- Hacer que el tipo usuario Usuario interactue con la aplicación

Que cuenta con 4 clases:

- PanelUsuarioSuperior
- PanelUsuarioCentro
- PanelUsuarioInferior
- VentanaUsuario

Objetivos:

La aplicación debe cumplir con ciertos requisitos que ayuden a cumplir con lo requerido por el parque de diversiones. Para ello se debe poder Clasificar las atracciones por tipo y por exclusividad y ser presentadas en el catálogo, se debe poder validar restricciones de acceso para cada atracción (no todas deben tener una restricción necesariamente), la aplicación debe

poder asignar empleados a atracciones según disponibilidad de turno, capacitaciones y riesgo, Debe poder calcular los precios de cada tiquete para clientes normales y precios con descuentos para empleados, debe poder generar códigos QR que permitan el acceso a los servicios del parque y marcar los códigos como usados una vez estos ya hayan sido leídos. Adicionalmente, el código debe tener persistencia en sus archivos, se le debe asignar un login y una contraseña a cada usuario (debe cumplir con credenciales seguras) y el sistema debe ser capaz de manejar una cantidad x de usuarios e información sin problema.

En esta fase se quieren realizar varias interfaces gráficas. La creación de códigos QR también se realiza en esta parte del proyecto.

Diagramas modelo:

- Diagrama de secuencia.
- Diagrama UML de clases.
- Diagrama de casos de uso.
- Diagrama de estados.

APIs

Para realizar la persistencia en el proyecto empleamos la técnica vista en clase de Lector/Escritor de archivos, la cual se trata de una implementación personalizada sencilla considerada como API interna. No se utilizaron APIs externas ya que, al ser un proyecto pequeño, las necesidades de este no las requerían, por lo que optamos por las APIs básicas de Java.io.

La capa de persistencia que se utilizó contiene:

- BufferedReader.
- FileReader.
- PrintWriter.
- Serialización tipo CSV.
- Conceptos previamente vistos en clase.

Código algoritmos críticos

```java
public static void escribirAtraccionesTxt(List<Atraccion> atracciones) {
    try {
        File carpeta = new File("./data/");
        if (!carpeta.exists()) carpeta.mkdirs();

        PrintWriter escritor = new PrintWriter(new File("./data/atracciones.txt"));
        DateTimeFormatter formato = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");

        for (Atraccion a : atracciones) {
            escritor.println(a.getNomAtraccion() + "--" +
                    a.getUbicacion() + "--" +
                    a.getCapMax() + "--" +
                    a.getNumMinEmp() + "--" +
                    a.getDateMin().format(formato) + "--" +
                    a.getDateMax().format(formato) + "--" +
                    a.getExcLevel());
        }

        escritor.close();

    } catch (FileNotFoundException e) {
        System.err.println("Error al escribir atracciones: " + e.getMessage());
    }
}

// Leer Documento
public static ArrayList<Atraccion> leerAtraccionesTxt() {
    ArrayList<Atraccion> atracciones = new ArrayList<>();
    DateTimeFormatter formato = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");

    try (BufferedReader lector = new BufferedReader(new FileReader("./data/atracciones.txt"))) {
        String linea = lector.readLine();
        while (linea != null) {
            String[] datos = linea.split("--");

            String nombre = datos[0];
            String ubicacion = datos[1];
            int capMax = Integer.parseInt(datos[2]);
```

Esta parte del código muestra la estructura que se utilizó para implementar la persistencia por medio del Lector y Escritor de archivos.

Diseño Final:

Aquí se encuentra la solución final al probelma palnteado en un prinicipio con todas las relaciones y clases necesarias: https://lucid.app/lucidchart/3e6aa985-e95a-424c-9769-f035674959fd/edit?viewport_loc=1860%2C-876%2C5240%2C2473%2C0_0&invitationId=inv_3d430c64-2abd-4f91-834a-8a5d0aa1c4a1