Diseño - Proyecto 1

Sergio Alejandro Torres Melendez - 202412255
Julian Camilo Rivera Guzmán - 202412458
Felipe Rojas Díaz - 202414023

Contexto a partir del diseño:

A partir de un respectivo análisis previo del problema, este apartado de diseño se enfoca en sobre todo en encontrar una solución basada explícitamente en lo que quiero, que en este caso se define como el dominio que se va a manejar en la correspondiente implementación y en lo que se demarca en este documento.

En esta vía, la solución del sistema planteado se basa primeramente en aquellas restricciones dadas en el análisis y las cuales restringen el dominio, estas restricciones fueron las siguientes y fueron las descritas y/o implementadas en el documento de análisis:

Restricciones de Información y persistencia:

- La aplicación debe estar hecha en Java.
- La carpeta no puede ser la misma carpeta donde se encuentre el código fuente de la aplicación.
- La carpeta no puede ser la misma carpeta donde se encuentre el código fuente de la aplicación

Restricciones de Disponibilidad:

- Algunas atracciones y espectáculos tienen disponibilidad por temporada o en fechas/horarios específicos.

Restricciones de Acceso a Atracciones:

- Las atracciones mecánicas tienen límites de altura y peso para los clientes.
- Las atracciones culturales tienen un cupo máximo de personas.
- El acceso a ciertas atracciones (culturales) está restringido por el tipo de tiquete (Familiar, Oro, Diamante).

Restricciones de Acceso a Espectáculos:

- Algunos espectáculos tienen restricciones de edad para el ingreso.

Restricciones Operativas (Atracciones y Espectáculos):

- Las atracciones culturales requieren un número mínimo de empleados para operar.
- Si no hay suficientes empleados asignados, una atracción no estará en servicio.

- Algunas atracciones tienen una operación restringida (vinculada a la venta de tiquetes).
- Algunos espectáculos tienen una operación restringida (aunque no necesariamente con empleados asociados).

Restricciones de Empleados:

- El día se divide en dos turnos (apertura y cierre).
- Para operar atracciones mecánicas de nivel alto, los empleados deben tener capacitación específica.
- Tiendas y taquillas deben tener al menos un cajero.

Restricciones de Venta y Uso de Tiquetes:

- El Tiquete Básico no da acceso a atracciones.
- Los tiquetes Familiar, Oro y Diamante tienen acceso limitado a ciertos tipos de atracciones.
- Los tiquetes de temporada tienen validez por un período específico y pueden ser de diferentes categorías (Familiar, Oro, Diamante).
- Las entradas individuales son válidas para una sola atracción y un solo uso.
- La compra de FastPass está limitada a un día específico.
- El sistema debe poder verificar si un tiquete ya ha sido utilizado.

A partir de las restricciones anteriores, se denota dentro del contexto un proceso iterativo, en el cual se basó para construir la solución.

Objetivos:

Con lo que respecta a los objetivos del diseño, se tiene muy en cuenta sobre todo la identificación de problemática. En este caso explícitamente se ve a la hora de, implementarlo en Java de una forma la cual no es correcta y no se sigue los lineamentos planteados anteriormente en el diagrama de casas (UML). Del mismo modo, una problemática presente es el hecho de un mal manejo de datos a la hora de probar la lógica y de realizar o no ponerlo bien en el tipo de archivo indicado, que en este caso es JSON.

A partir de haber identificado estas problemáticas, se explayan los siguientes objetivos del sistema teniendo en cuanta lo anterior:

1. Controlar las diferentes atracciones y espectáculos.
2. Manejar a los empleados.
3. Dar un catálogo de diferentes atracciones que tiene el parque.
4. Vender tiquetes a los visitantes del parque.

Estos cuatro aspectos me generalizan lo que debe hacer el sistema y/o la solución planteada. Esto se conecta directamente con los objetivos a la hora de plantear el diseño, dado que tanto con el dominio, como con las preocupaciones transversales define como y por cuáles caminos si o si debe tomar, así como no debe hacerlo.

No objetivos:

En la solución planteada se tiene que tener en cuenta aquellos escenarios donde se puede confundir y/o malinterpretar lo que se pide a implementar. En este caso, explícitamente se pide un sistema que haga esto:

1. Controlar las diferentes atracciones y espectáculos.
2. Manejar de los empleados.
3. Dar un catálogo de diferentes atracciones que tiene el parque.
4. Vender tiquetes a los visitantes del parque.

En los casos que puede llegar a malinterpretar lo que se tiene que hacer en la solución es, por ejemplo:

1. Modelo de un parque que aborde absolutamente todos los conceptos que lo puede conllevar.
2. Un sistema el cual cree montañas rusas.
3. Un sistema que maneje y estructure a los empleados en contextos como lo es pagos/nómina.

Responsability - Driven Design

En este aspecto se incluirá esta metodología, la cual se basa en 3 conceptos principales; primero, Roles; segundo, Responsabilidades y tercero, Colaboraciones entre los Roles.

Roles:

Para este aspecto se darán los roles/objetos **candidatos** según el problema y la solución a realizar, también teniendo en cuenta que estos a un bajo nivel son **Clases** en Java:

• Atracciones • Atracciones Mecánicas • Atracciones Culturales • Espectáculos • Empleado • Cliente • Lugar de servicio • Cajero	• Administrador • Cocina • Tiquete Básico • Tiquete Familiar • Tiquete Oro • Tiquete Diamante • Tiquete Temporada • Tiquete Fastpass • Sistema parque

Estereotipos:

De acuerdo a las clases candidatas mencionadas anteriormente, se les pondrán unas series de etiquetas para facilitar la elección de responsabilidades:

- Atracciones <<Information holder>>
- Atracciones Mecánicas <<Information holder>>

- Atracciones Culturales <<Information holder>>
- Espectáculos <<Information holder>>
- Empleado <<Information holder>>
- Cliente <<Information holder>>
- Lugar de servicio <<Information holder>>
- Cajero <<Service provider>>
- Administrador <<Information holder>>
- Cocina <<Information holder>>
- Tiquete Básico <<Information holder>>
- Tiquete Familiar <<Information holder>>
- Tiquete Oro <<Information holder>>
- Tiquete Diamante <<Information holder>>
- Tiquete Temporada <<Information holder>>
- Tiquete Fastpass <<Information holder>>
- Sistema parque <<Coordinator>> / <<Controller>> / <<Interfacer>>

Responsabilidades:

En este apartado se van a señalar a más bajo nivel los métodos de los roles o ya directamente de las clases en la solución implementadas, asimismo, se agruparán estas responsabilidades en grupos para no ser redundantes y para hacerlo directamente en los subgrupos que conllevan aquellas respectivas clases.

I. Responsabilidades de Gestión de Atracciones y Espectáculos.

1. **Gestionar Datos Maestros de Atracciones:**
    - Almacenar, modificar y consultar la información central de cada atracción: tipo (mecánica, cultural), nombre, descripción, ubicación (fija para culturales).
    - Gestionar las restricciones específicas: límites de altura/peso, nivel de riesgo (para asignación de empleados), capacidad máxima (culturales), número mínimo de empleados requeridos.
    - Administrar el nivel de exclusividad (Familiar, Oro, Diamante) asociado a cada atracción.
    - (Opcional) Gestionar información de salud/contraindicaciones.
2. **Gestionar Datos Maestros de Espectáculos:**
    - Almacenar, modificar y consultar la información central de cada espectáculo: nombre, descripción, restricciones de edad (si aplican).
    - Gestionar las fechas y horarios específicos en los que ocurren.
3. **Controlar la Disponibilidad Temporal:**
    - Gestionar y aplicar las reglas de "temporada" para atracciones y espectáculos (disponibilidad por rangos de fechas o meses).
    - Determinar si una atracción/espectáculo está programado para estar disponible en una fecha/hora específica.
4. **Determinar el Estado Operativo:**
    - Evaluar en tiempo real si una atracción o espectáculo puede operar, considerando:
        - Disponibilidad temporal (según temporada/horario).
        - Suficiencia de empleados *calificados* asignados (interactuando con el módulo de Empleados).
        - Estado de "Operación Restringida" (si está marcado como no disponible por otros motivos, ej. mantenimiento - implícito).

5. **Validar Cumplimiento de Restricciones del Cliente:**
    - Proveer la lógica para verificar si un cliente cumple con las restricciones físicas (altura, peso) y de edad para una atracción o espectáculo específico.
6. **Proveer Información Consultable:**
    - Permitir a Clientes, Empleados y Administradores consultar la información relevante y el estado actual de atracciones y espectáculos.

II. Responsabilidades de Gestión de Empleados.

7. **Gestionar Datos Maestros de Empleados:**
    - Almacenar, modificar y consultar la información de los empleados: datos personales, tipo/rol (Normal, Alto Riesgo, Cocinero, Cajero).
    - Gestionar las capacitaciones y habilidades: qué atracciones mecánicas pueden operar (medio/alto riesgo), si son aptos para cocina, etc.
8. **Gestionar la Definición de Lugares de Servicio:**
    - Almacenar y administrar los diferentes tipos de lugares de servicio (Cafeterías, Tiendas, Taquillas) y sus requerimientos operativos (ej. necesidad de cajero, cocinero).
9. **Planificar y Asignar Turnos:**
    - Definir los turnos de trabajo diarios (apertura, cierre).
    - Asignar empleados a turnos específicos.
    - Asignar un lugar de trabajo (atracción específica, lugar de servicio, o "Servicio General") y una tarea a cada empleado para cada turno, respetando sus capacitaciones y los requerimientos del lugar (ej. cocinero a cocina, personal entrenado a atracción de alto riesgo).
10. **Informar Asignaciones Diarias:**
    - Permitir a los empleados consultar su lugar y tarea asignados para el turno actual.
11. **Soportar la Operatividad de Atracciones:**
    - Proveer información al módulo de Atracciones sobre cuántos empleados (y si son los adecuados) están asignados a cada atracción en un momento dado para determinar su estado operativo.

III. Responsabilidades de Venta y Control de Tiquetes.

12. **Gestionar Tipos de Tiquetes:**
    - Definir y administrar las características de cada tipo de tiquete: Básico, Familiar, Oro, Diamante, de Temporada (semanal, mensual, etc.), Entradas Individuales, FastPass.
    - Establecer las reglas de acceso que cada tipo de tiquete otorga (qué niveles de exclusividad cubre).
    - Gestionar la validez temporal de los tiquetes (ej. tiquetes de temporada, FastPass para un día).
13. **Procesar Ventas de Tiquetes:**

    - Registrar la venta de tiquetes a través de los diferentes canales (en línea, taquillas).
    - Calcular precios, aplicando descuentos correspondientes (empleados, promociones de temporada).
    - Asociar tiquetes vendidos a un comprador (cuando sea necesario o relevante).
14. **Validar Tiquetes para Acceso:**
    - Verificar la validez de un tiquete en puntos de control (entrada al parque, acceso a atracción).
    - Comprobar que el tipo de tiquete permite el acceso según la exclusividad de la atracción.
    - Verificar la fecha/hora de validez (para tiquetes de temporada o FastPass).
15. **Controlar el Uso y Prevenir Fraude:**
    - Marcar un tiquete como "usado" después de su validación (especialmente para tiquetes individuales o sin fecha específica) para evitar reutilización.
    - Implementar mecanismos para identificar intentos de fraude.
16. **Gestionar FastPass:**
    - Manejar la venta de FastPass asociados a un día específico.
    - Validar el FastPass en las atracciones correspondientes.

IV. Responsabilidades Transversales / de Soporte

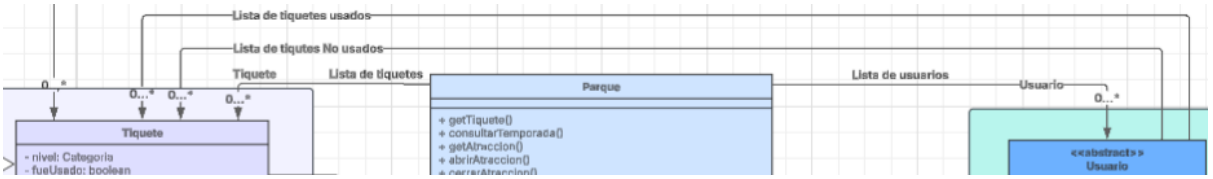
17. **Autenticación y Autorización:**
    - Verificar la identidad de los usuarios (Login para Admin/Empleados).
    - Controlar qué acciones puede realizar cada tipo de usuario (Admin puede modificar, Empleado/Cliente solo consultar ciertas cosas).
18. **Persistencia de Datos:**
    - Almacenar de forma segura y recuperable toda la información gestionada por los módulos anteriores (en bases de datos, etc.).
19. **Configuración del Sistema:**
    - Permitir a los administradores ajustar parámetros generales del sistema (ej. horarios del parque, quizás umbrales para alertas).

Las responsabilidades fueron segmentadas por Gemini a partir de la lluvia de ideas proporcionada en el documento de análisis de este proyecto.

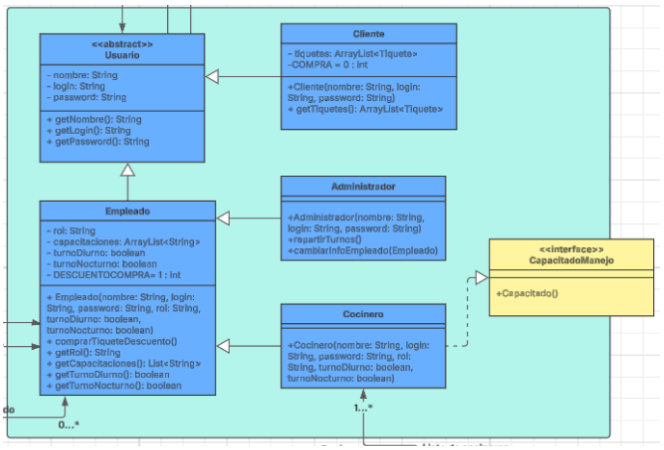
Colaboraciones:

Ya teniendo claro tanto cuáles son los las clases o roles a tener en cuenta y las responsabilidades o métodos de estas, llega el momento de relacionarlas, esto implicando claramente mínimo dos roles dentro de las relaciones a aplicar sobre una misma; sin embargo, también se puede ver relaciones que contribuyan a más de una clase, por ejemplo la clase central, entre otras. Para verificar estos **contratos** que se tienen dentro de la solución planteada dentro del sistema, se utilizara el UML (diagrama de clases) para ilustrarlo de una buena manera.

Clase **Parque** se relaciona con: **Tiquete** (Da una lista de tiquetes), **Usuario** (Da una lista de usuarios), **Atracciones** (Da una lista tanto de atracciones como de espectáculos) y **LugarServicio** (Da una lista de lugares de servicio).

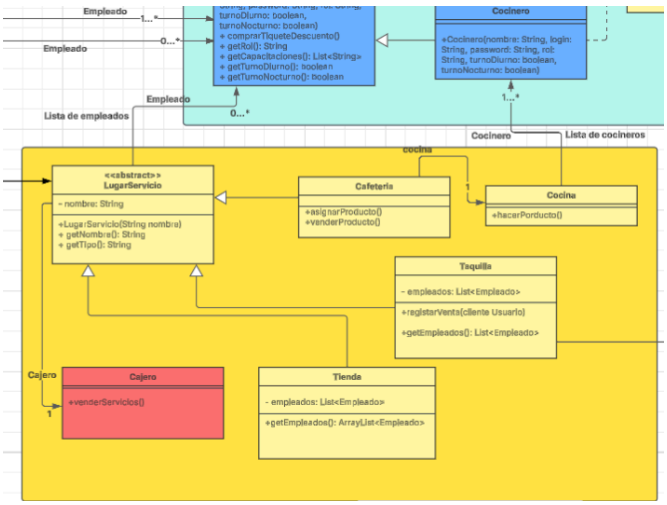


Clase **Usuario** se relaciona con: **Tiquete** (Tiene dos listas de tiquetes usados y no usados)



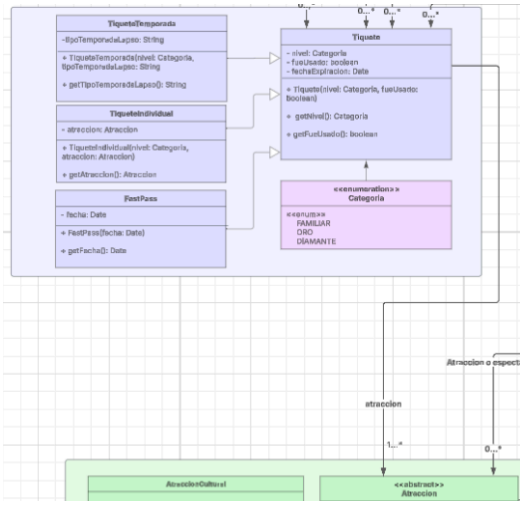
Clase principal subgrupo **Usuario:**

- Subclases de **Usuario:**
  - **Cliente** se relaciona con: **Usuario** (Dado que hereda de la clase abstracta)
  - **Empleado** se relaciona con: **Usuario** (Dado que hereda de la clase abstracta)
    - **Administrador** se relaciona con: **Empleado** (Dado que hereda de esa clase)
    - **Cocinero** se relaciona con: **Empleado** (Dado que hereda de esa clase) y **CapacitadoManejo** (Dado que hereda de una interfaz que le obliga a implementar ese método)



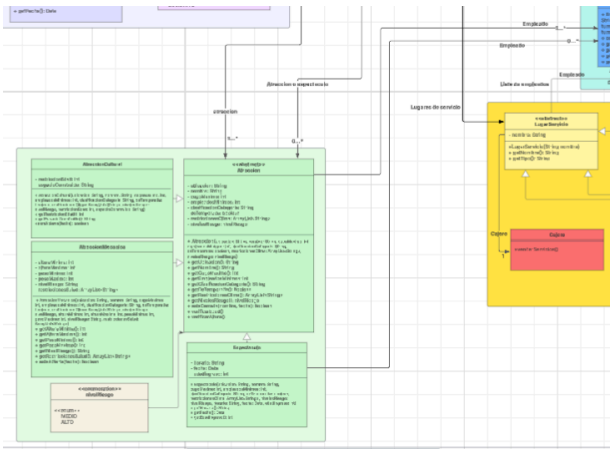
Clase principal subgrupo **Lugares de servicio:**

- **LugarServicio** se relaciona con**: Empleado** (Tiene una lista de empleados del lugar)
- Subclases de **LugarServicio:**
    - **Tienda** se relaciona con: **LugarServicio** (Dado que hereda de la clase abstracta)
    - **Taquilla** se relaciona con: **LugarServicio** (Dado que hereda de la clase abstracta) y **Tiquete** (Tiene una lista de tiquetes para vender)
    - **Cafetería** se relaciona con: **LugarServicio** (Dado que hereda de la clase abstracta) y **Cafetería** (Dado que una cafetería tiene una cocina)
        - **Cafetería** se relaciona con: **Cocinero** (Tiene una lista de cocineros)



Clase principal subgrupo **Tiquetes:**

- **Tiquete** se relaciona con**: Atracción** (Un tiquete tiene relacionado una atracción o más atracciones)
- Subclases de **Tiquete:**
    - **TiqueteTemporada** se relaciona con: **Tiquete** (Dado que hereda de la clase abstracta)
    - **TiqueteIndividual** se relaciona con: **Tiquete** (Dado que hereda de la clase abstracta)
    - **Fastpass** se relaciona con: **Tiquete** (Dado que hereda de la clase abstracta)

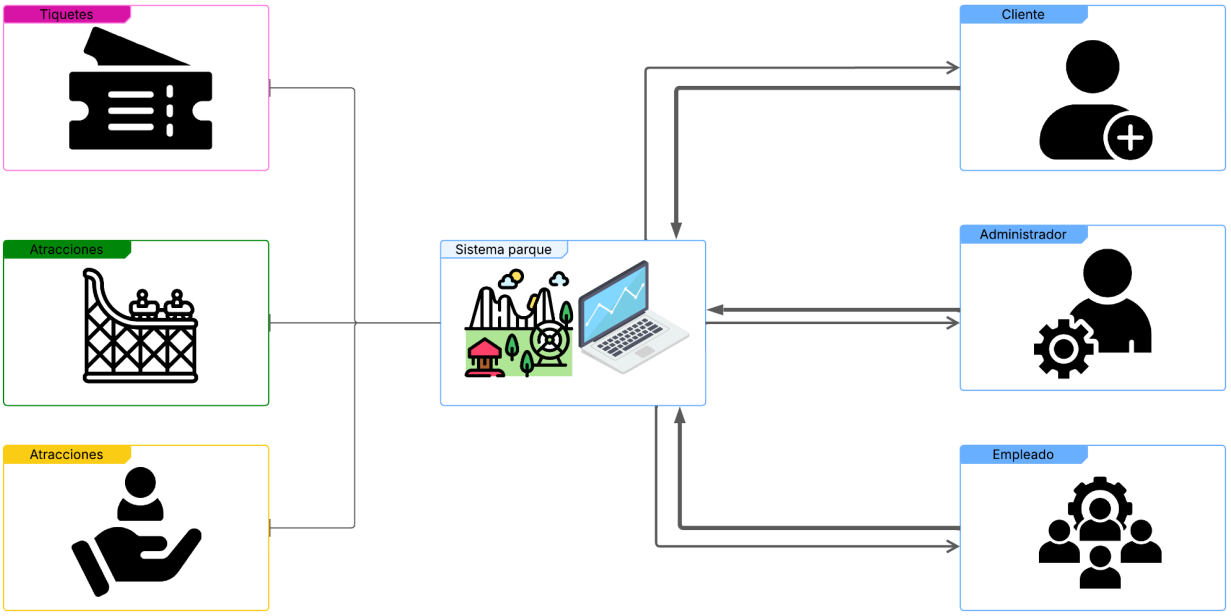


Clase principal subgrupo **Atracciones y Espectáculos:**

- **Atracción** se relaciona con**: Empleado** (Tiene una lista de empleados, o sea tiene al menos un empleado asignado a la atracción)
- Subclases de **Atraccion:**
    - **AtracciónMecanica** se relaciona con: **Atracción** (Dado que hereda de la clase abstracta)
    - **AtraccionCultural** se relaciona con: **Atracción** (Dado que hereda de la clase abstracta)
    - **Espectaculo** se relaciona con: **Atracción** (Dado que hereda de la clase abstracta) y **Empleado** (Puede tener una lista de empleados, o sea puede haber un empleado asignado a la atracción *NO ES OBLIGATORIO*)

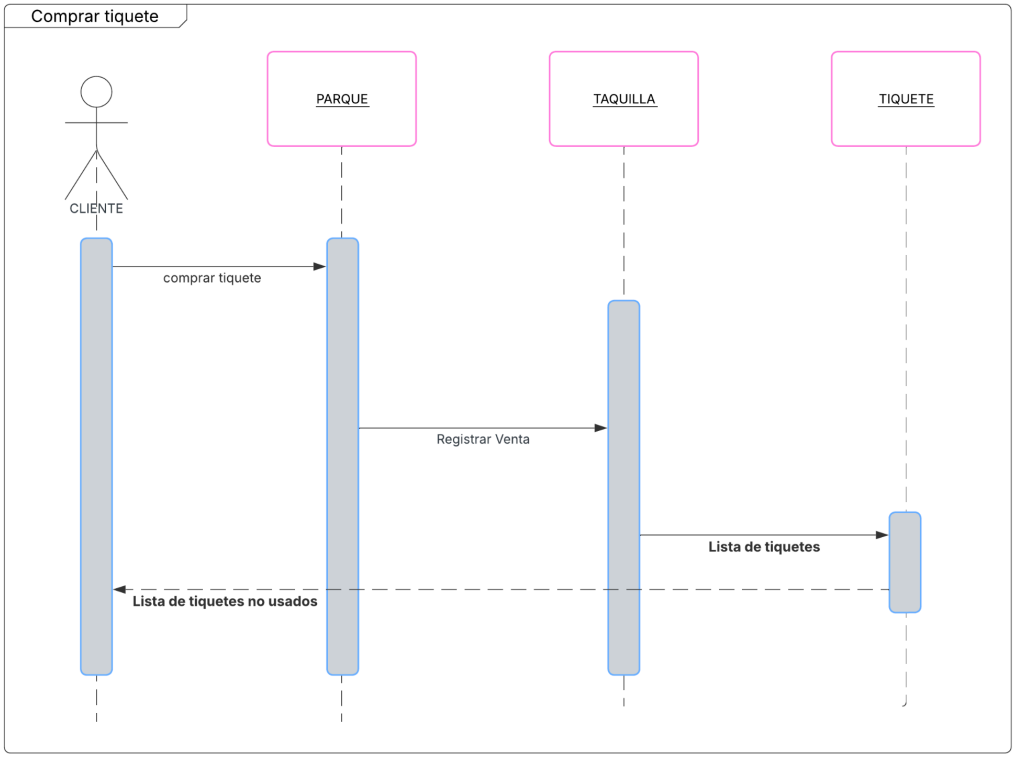
Diagrama de contexto:

El diagrama presentado a continuación masifica y/o ejemplifica de forma dinámica, a partir de imágenes cotidianas, el funcionamiento de la solución presentada y la interacción con el sistema.



Diagramas de secuencia:

En este apartado, se dará a conocer por medio de diagramas de secuencia aquellas responsabilidades o métodos más importantes en la solución planteada para el sistema correspondiente.



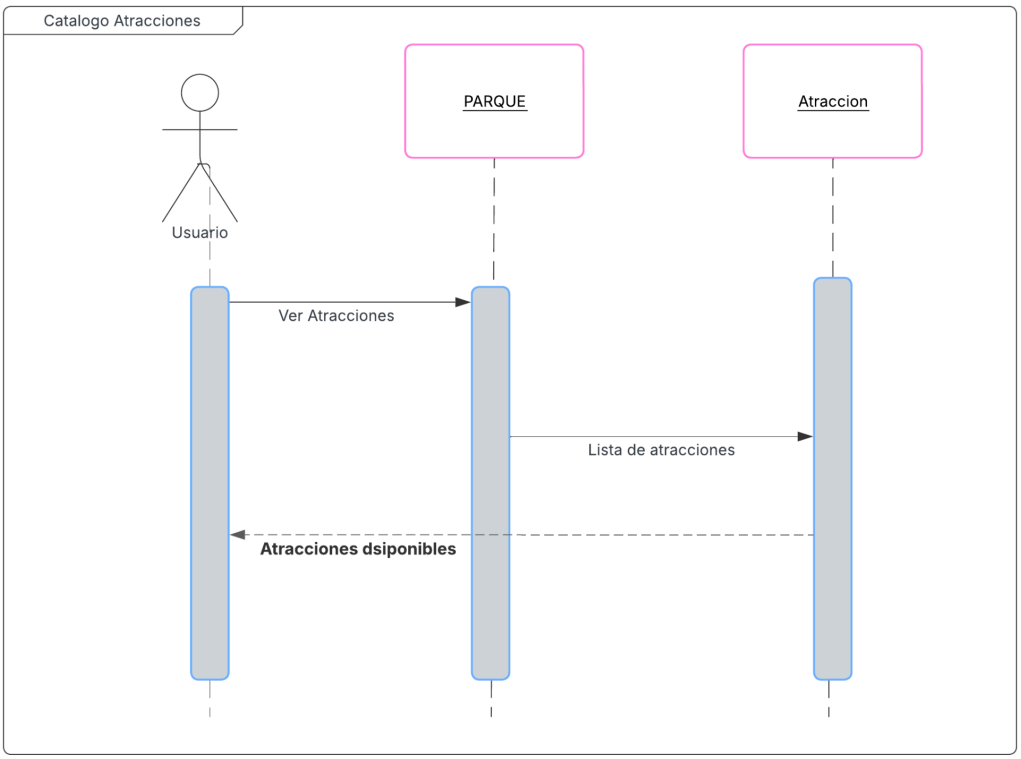
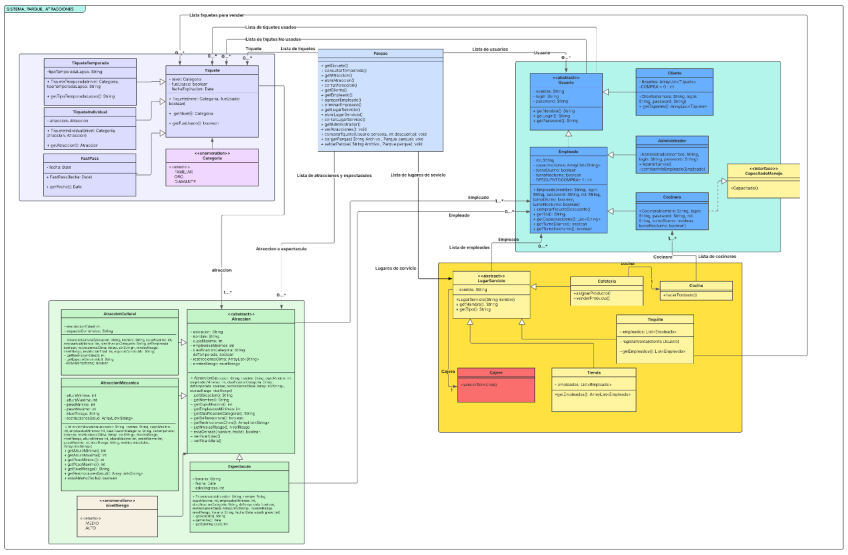
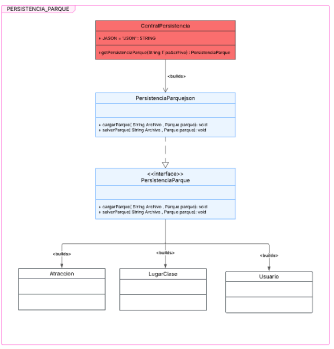
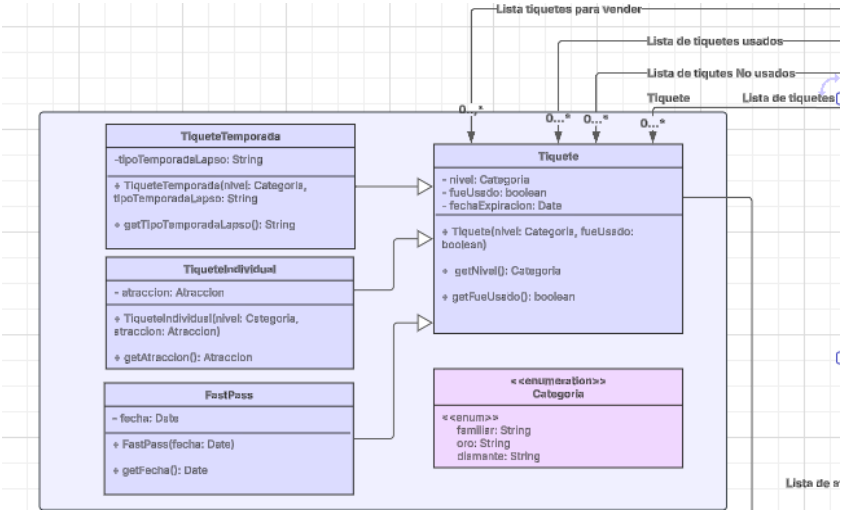
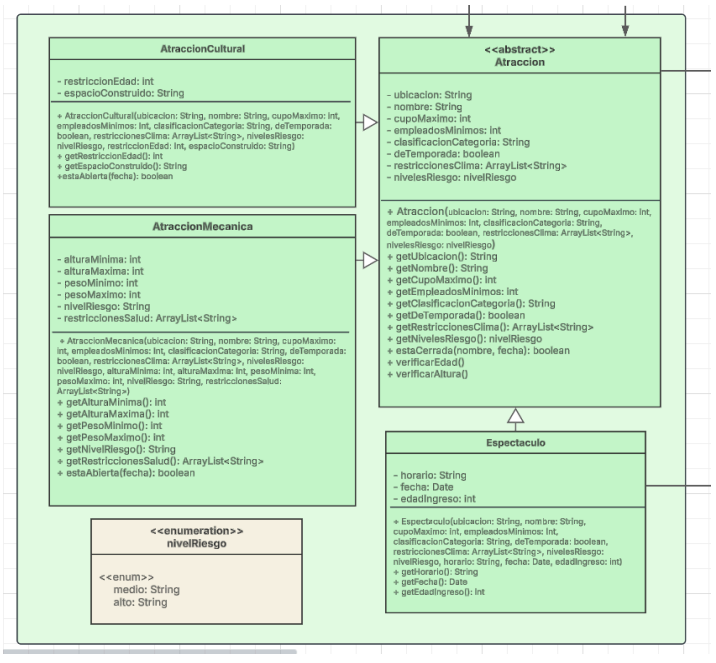


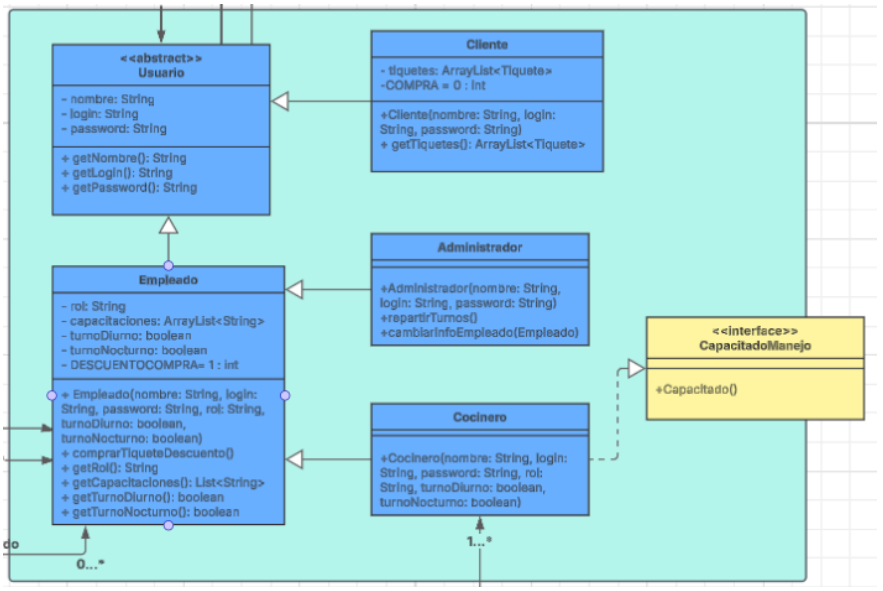
Diagrama de clases (UML):

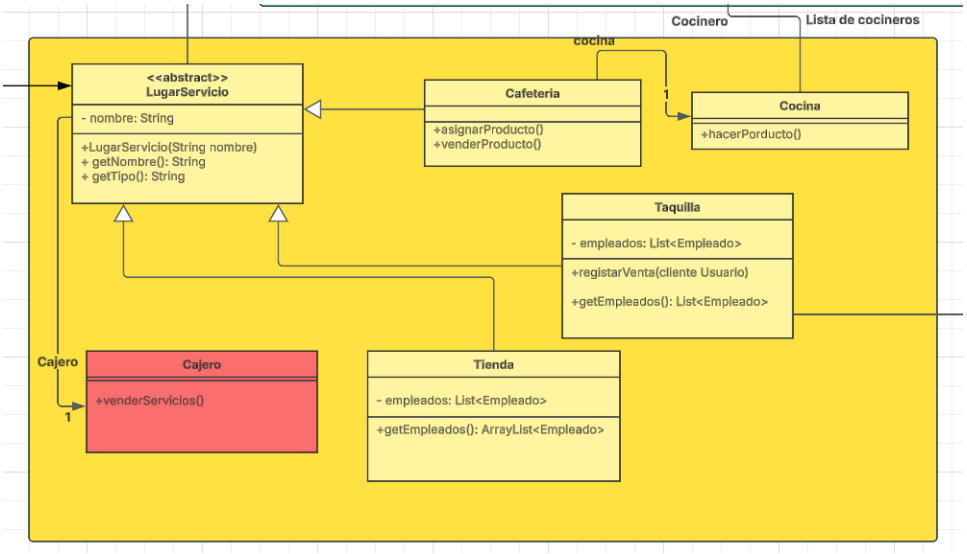









<<abstract>>
Usuario
- nombre: String
- login: String
- password: String
+ getNombre(): String
+ getLogin(): String
+ getPassword(): String
Cliente
- tiquetes: ArrayList<Tiquete>
-COMPRA = 0 : int
+Cliente(nombre: String, login: String, password: String)
+ getTiquetes(): ArrayList<Tiquete>
Administrador
+Administrador(nombre: String, login: String, password: String)
+repartirTurnos()
+cambiarInfoEmpleado(Empleado)
Empleado
- rol: String
- capacitaciones: ArrayList<String>
- turnoDiurno: boolean
- turnoNocturno: boolean
- DESCUENTOCOMPRA= 1 : int
+ Empleado(nombre: String, login: String, password: String, rol: String, turnoDiurno: boolean, turnoNocturno: boolean)
+ comprarTiqueteDescuento()
+ getRol(): String
+ getCapacitaciones(): List<String>
+ getTurnoDiurno(): boolean
+ getTurnoNocturno(): boolean
<<interface>>
CapacitadoManejo
+Capacitado()
Cocinero
+Cocinero(nombre: String, login: String, password: String, rol: String, turnoDiurno: boolean, turnoNocturno: boolean)
do
0...*
1...*


Cocinero
Lista de cocineros
cocina
<<abstract>>
LugarServicio
- nombre: String
+LugarServicio(String nombre)
+ getNombre(): String
+ getTipo(): String
Cafeteria
+asignarProducto()
+venderProducto()
1
Cocina
+hacerPorducto()
Taquilla
- empleados: List<Empleado>
+registarVenta(cliente Usuario)
+getEmpleados(): List<Empleado>
Cajero
Cajero
+venderServicios()
1
Tienda
- empleados: List<Empleado>

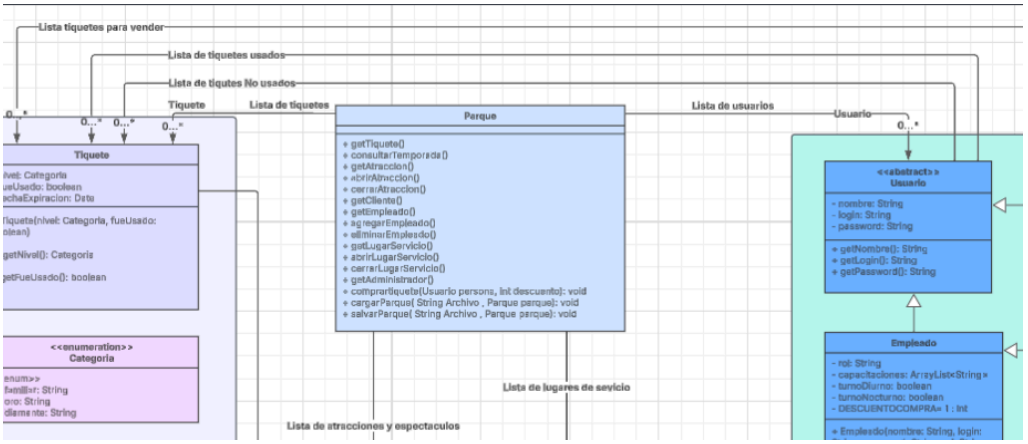
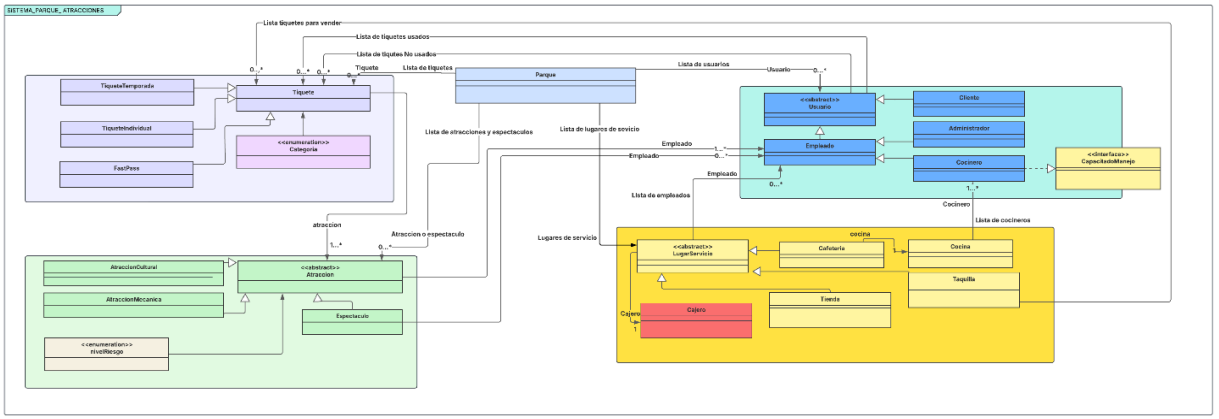
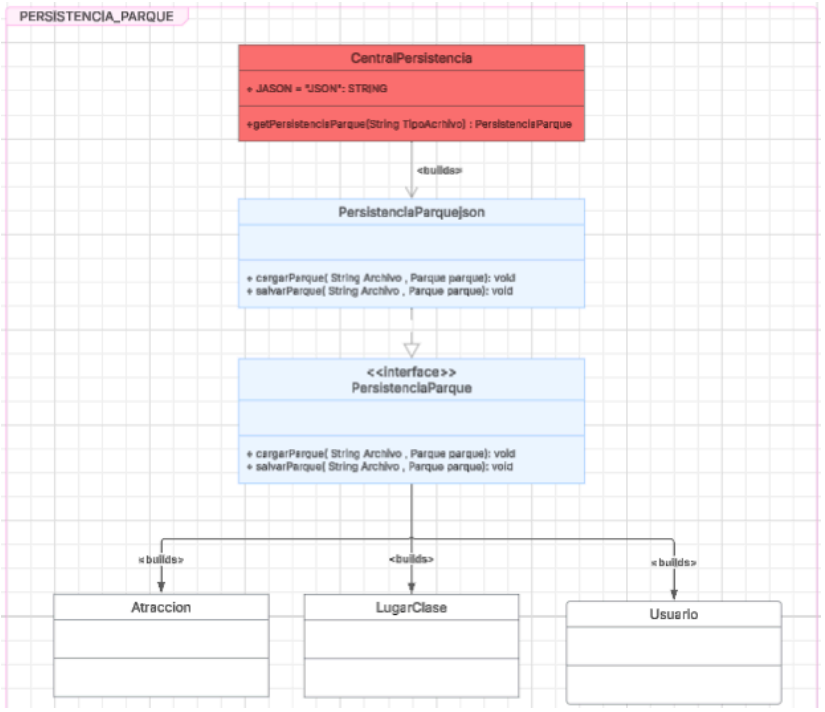

Lista tiquetes para vender
Lista de tiquetes usados
Lista de tiqutes No usados
Tiquete
Lista de tiquetes
Perque
Lista de usuarios
Usuario
Tiquete
<<enumeration>>
Categoria
+ getTiquete()
+ consultarTemporada()
+ getAtraccion()
+ abrirAtraccion()
+ cerrarAtraccion()
+ getCliente()
+ getEmpleado()
+ agregarEmpleado()
+ eliminarEmpleado()
+ getLugarServicio()
+ abrirLugarServicio()
+ cerrarLugarServicio()
+ getAdministrador()
+ comprarTiquete(Usuario persona, Int descuento): void
+ cargarParque(String Archivo , Parque parque): void
+ salvarParque(String Archivo , Parque parque): void
<<abstract>>
Usuario
- nombre: String
- login: String
- password: String
+ getNombre(): String
+ getLogin(): String
+ getPassword(): String
Empleado
- rol: String
- capacitaciones: ArrayList<String>
- turnoDiurno: boolean
- turnoNocturno: boolean
- DESCUENTOCOMPRA= 1 : int
Lista de lugares de sevicio
Lista de atracciones y espectaculos

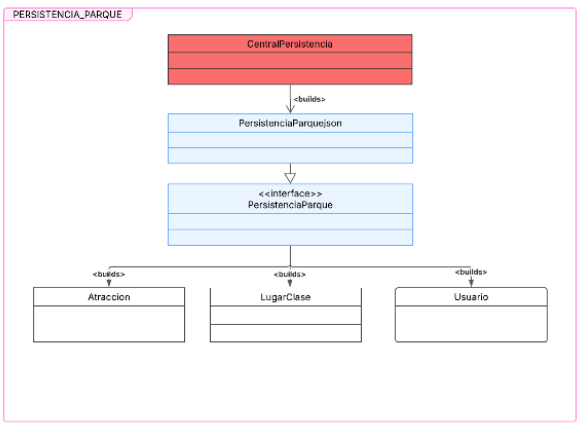
Diagrama de clases de alto nivel (UML-Relaciones):



Persistencia:

- **Diagramas (UML):**





- **Información acerca de la persistencia:**

  La información persistida en la solución planteada se hizo a partir de archivos .JSON, los cuales son excelentes para el entendimiento de estos y para su posterior aplicabilidad dentro del sistema, aplicado a los objetos y sus respectivas instalaciones, entre otras cosas. Se usó para la implementación la libreria GSON (versión 2.10.1), la cual permite hacer todo lo que respecta a la persistencia en java. Del mismo modo, en la mayoría de la implementación de la persistencia se uso como herramienta ***Gemini***, esto para hacer mejor y efectuar tanto la carga del json y sobre todo de los requerimientos.

Fragmentos de pseudocódigo:

- **Package Atracciones:**

  La parte más importante del package de atracciones es la clase abstracta de atracción porque es la base o esqueleto de todo el package, con esta clase se pueden tener los atributos principales de las atracciones o espectáculos. Adicionalmente, a partir de esta clase se diseña el método constructor tanto de los espectáculos como de las atracciones mecánicas y culturales, dentro de estos atributos generales se encuentra el nombre, ubicación, cantidad de empleados mínimos, etc.

```java
public abstract class Atraccion {

    private String ubicacion;
    private String nombre;
    private int cupoMaximo;
    private int empleadosMinimos;
    private String clasificacionCategoria;
    private boolean deTemporada;
    private ArrayList<String> restriccionesClima;
    public NivelesRiesgo nivelRiesgo;

    public Atraccion(String ubicacion, String nombre, int cupoMaximo, int empleadosMinimos,
            String clasificacionCategoria, boolean deTemporada, ArrayList<String> restriccionesClima,
            NivelesRiesgo nivelRiesgo) {
        super();
        this.ubicacion = ubicacion;
        this.nombre = nombre;
        this.cupoMaximo = cupoMaximo;
        this.empleadosMinimos = empleadosMinimos;
        this.clasificacionCategoria = clasificacionCategoria;
        this.deTemporada = deTemporada;
        this.restriccionesClima = restriccionesClima;
        this.nivelRiesgo = nivelRiesgo;
    }
```

Cuerpo clase abstracta atracción

```java
public class AtraccionCultural extends Atraccion {
    public AtraccionCultural(String ubicacion, String nombre, int cupoMaximo, int empleadosMinimos,
            String clasificacionCategoria, boolean deTemporada, ArrayList<String> restriccionesClima,
            NivelesRiesgo nivelRiesgo) {
        super(ubicacion, nombre, cupoMaximo, empleadosMinimos, clasificacionCategoria, deTemporada, restriccionesClima,
                nivelRiesgo);

    }
```

Método constructor (Ejemplo con el método constructor de AtraccionMecanica)

- **Package Tiquetes:**

```java
package sistema_parque.tiquetes;

import java.util.Date;

import sistema_parque.atracciones.Atraccion;

public class Tiquete {

    private Atraccion atraccion;
    public Atraccion getAtraccion() {
        return atraccion;
    }

    public Date getFechaExpiracion() {
        return fechaExpiracion;
    }

    protected Categoria nivel;
    private boolean fueUsado;
    private Date fechaExpiracion;

    public Tiquete(Categoria nivel, boolean fueUsado) {
        super();
        this.nivel = nivel;
        this.fueUsado = fueUsado;
    }

    public Categoria getNivel() {
        return nivel;
    }

    public boolean isFueUsado() {
        return fueUsado;
    }
}
```

cuerpo superclase tiquete

La parte más importante del package de tiquetes es la superclase Tiquete, puesto que de esta se heredan todos los tipos de tiquetes que se pueden comprar (individual, temporada y fastpass). Teniendo en cuenta lo anterior, las subclases heredan algunos atributos de la clase tiquete para que se pueda ahorrar código a la hora de completar la información de cada tipo de tiquete, pues cosas como la fecha de expiración y el nivel lo comparten todos los tipos de tiquete. Además, en el atributo nivel de la clase tiquete se guarda toda la información relacionada a la categoría en la que se encuentra el tiquete, que se obtiene gracias a la enumeración “Categoría”.

Conclusiones - diseño: Durante la elaboración del diseño de este proyecto pudimos notar que, es necesario el buen orden y establecimiento de los atributos y métodos al momento de hacer un esquema/borrador del código, ya que, en algunas ocasiones pueden pasarse por alto algunos detalles o aspectos que cambian radicalmente el funcionamiento y practicidad de la aplicación.

Referencias.

Parque de atracciones iconos creados por Freepik - Flaticon

Ordenador portátil iconos creados por vectorsmarket15 - Flaticon

Nuevo usuario iconos creados por Aswell Studio - Flaticon

Administrador iconos creados por SBTS2018 - Flaticon

Desarrollo de habilidades iconos creados por syafii5758 - Flaticon

Validar boleto iconos creados por Slidicon - Flaticon

Parque-tematico iconos creados por mynamepong - Flaticon

Apoyo iconos creados por Freepik - Flaticon