Análisis del Proyecto 1. Grupo 2 Entrega 2

Juan Sebastian Garcia 202323466

Santiago Andrés Rojas 202221434

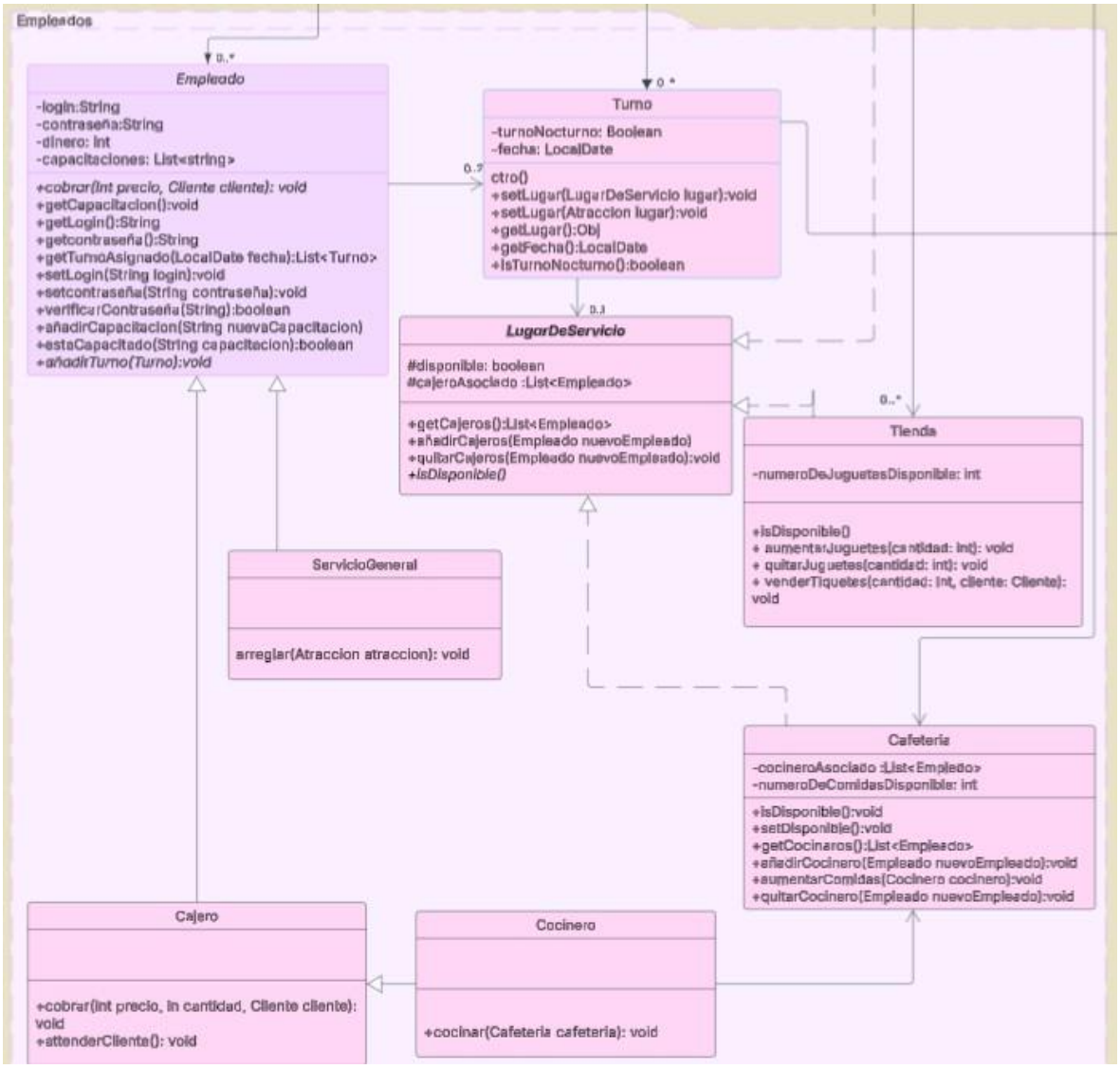
Hernando José Díaz 202220213

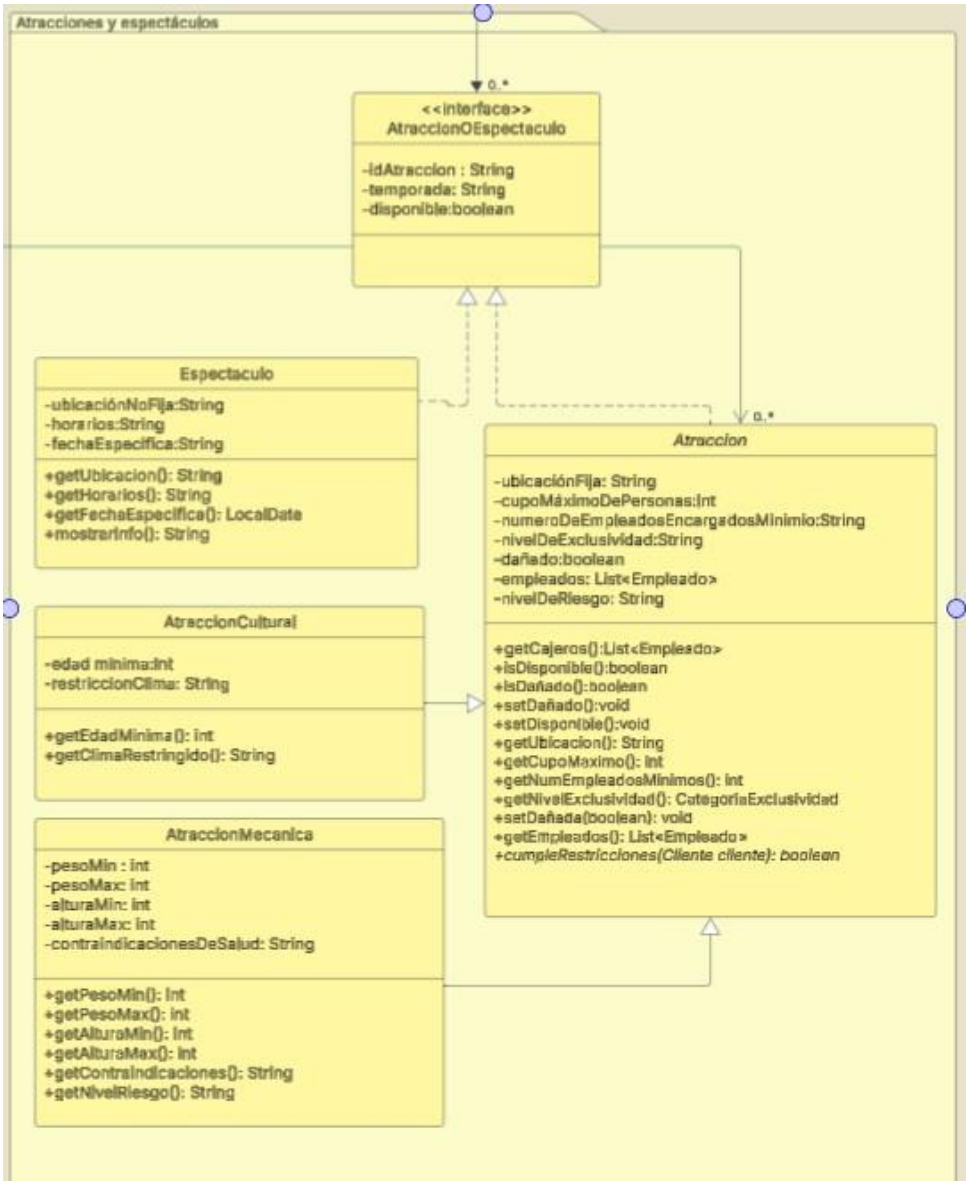
Introducción del proyecto

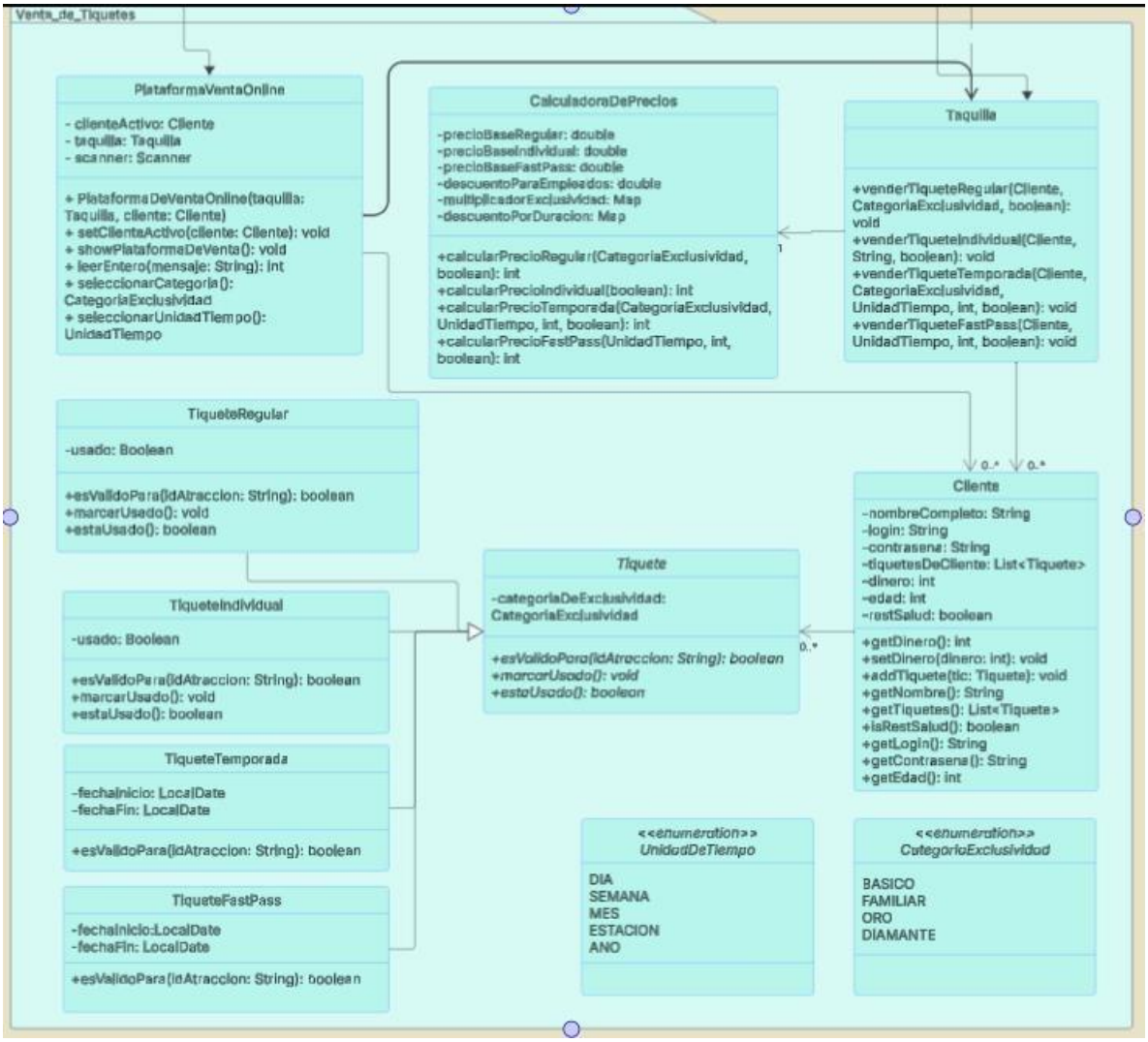
"En este proyecto construirá un sistema que apoye las operaciones administrativas de un parque de diversiones, que incluye atracciones mecánicas, atracciones culturales y espectáculos en vivo. Este sistema tiene 3 funcionalidades principales. En primer lugar, tendrá un catálogo de las diferentes atracciones que tiene el parque. En segundo lugar, un sistema de gestión de empleados y labores en las atracciones. En tercer lugar, permitirá la venta de tiquetes a los visitantes del parque."

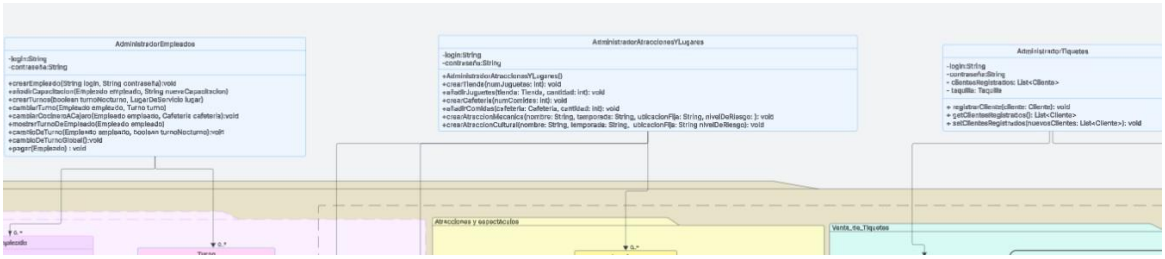
Requerimientos minimos para el documento

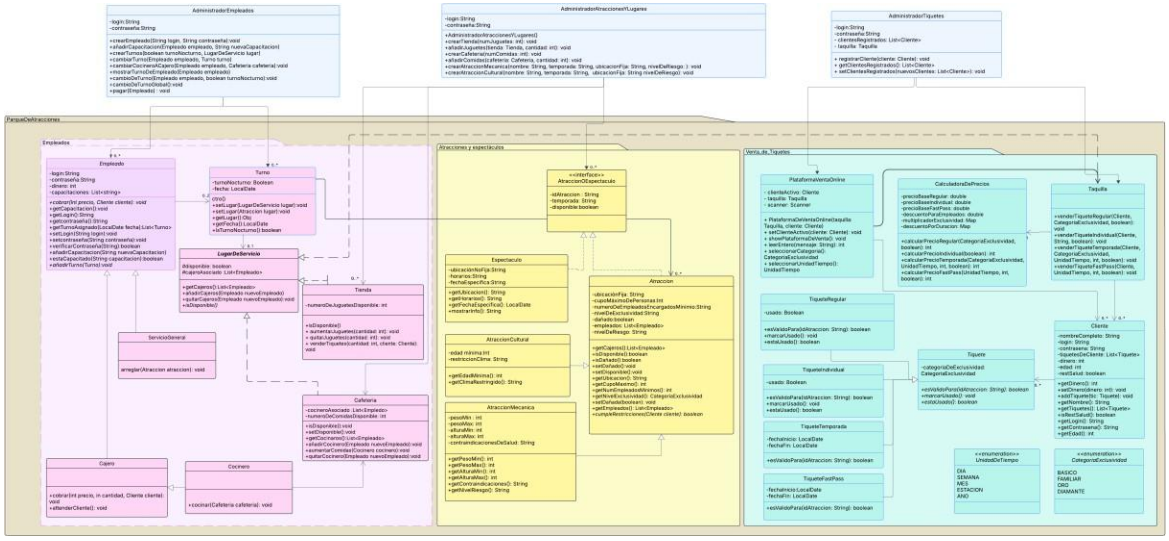
a. Al menos un diagrama de clases que incluya todas las clases del sistema, incluyendo sus relaciones, atributos y métodos. Los diagramas deben cubrir tanto la interfaz como la parte del diseño dedicado a la lógica de dominio.


Empleados
Empleado
-login:String
-contraseña:String
-dinero: Int
-capacitaciones: List<string>
+cobrar(Int precio, Cliente cliente): void
+getCapacitacion():void
+getLogin():String
+getcontraseña():String
+getTurnoAsignado(LocalDate fecha):List<Turno>
+setLogin(String login):void
+setcontraseña(String contraseña):void
+verificarContraseña(String):boolean
+añadirCapacitacion(String nuevaCapacitacion)
+estaCapacitado(String capacitacion):boolean
+añadirTurno(Turno):void
Turno
-turnoNocturno: Boolean
-fecha: LocalDate
ctro()
+setLugar(LugarDeServicio lugar):void
+setLugar(Atraccion lugar):void
+getLugar():Obj
+getFecha():LocalDate
+IsTurnoNocturno():boolean
LugarDeServicio
#disponible: boolean
#cajeroAsociado :List<Empleado>
+getCajeros():List<Empleado>
+añadirCajeros(Empleado nuevoEmpleado)
+quitarCajeros(Empleado nuevoEmpleado):void
+IsDisponible()
Tienda
-numeroDeJuguetesDisponible: int
+IsDisponible()
+ aumentarJuguetes(cantidad: int): void
+ quitarJuguetes(cantidad: int): void
+ venderTiquetes(cantidad: int, cliente: Cliente): void
ServicioGeneral
arreglar(Atraccion atraccion): void
Cafeteria
-cocineroAsociado :List<Empleado>
-numeroDeComidasDisponible: int
+IsDisponible():void
+setDisponible():void
+getCocineros():List<Empleado>
+añadirCocinero(Empleado nuevoEmpleado):void
+aumentarComidas(Cocinero cocinero):void
+quitarCocinero(Empleado nuevoEmpleado):void
Cajero
+cobrar(int precio, in cantidad, Cliente cliente): void
+attenderCliente(): void
Cocinero
+cocinar(Cafeteria cafeteria): void


Atracciones y espectáculos
0..*
<<interface>>
AtraccionOEspectaculo
-idAtraccion : String
-temporada: String
-disponible:boolean
Espectaculo
-ubicaciónNoFija:String
-horarios:String
-fechaEspecifica:String
+getUbicacion(): String
+getHorarios(): String
+getFechaEspecifica(): LocalDate
+mostrarInfo(): String
0..*
Atraccion
-ubicaciónFija: String
-cupoMáximoDePersonas:Int
-numeroDeEmpleadosEncargadosMinimio:String
-nivelDeExclusividad:String
-dañado:boolean
-empleados: List<Empleado>
-nivelDeRiesgo: String
+getCajeros():List<Empleado>
+isDisponible():boolean
+isDañado():boolean
+setDañado():void
+setDisponible():void
+getUbicacion(): String
+getCupoMaximo(): Int
+getNumEmpleadosMinimos(): Int
+getNivelExclusividad(): CategoriaExclusividad
+setDañada(boolean): void
+getEmpleados(): List<Empleado>
+cumpleRestricciones(Cliente cliente): boolean
AtraccionCultural
-edad minima:Int
-restriccionClima: String
+getEdadMinima(): Int
+getClimaRestringido(): String
AtraccionMecanica
-pesoMin : Int
-pesoMax: Int
-alturaMin: Int
-alturaMax: Int
-contraindicacionesDeSalud: String
+getPesoMin(): Int
+getPesoMax(): Int
+getAlturaMin(): Int
+getAlturaMax(): Int
+getContraindicaciones(): String
+getNivelRiesgo(): String


Venta_de_Tiquetes
PlataformaVentaOnline
- clienteActivo: Cliente
- taquilla: Taquilla
- scanner: Scanner
+ PlataformaDeVentaOnline(taquilla: Taquilla, cliente: Cliente)
+ setClienteActivo(cliente: Cliente): void
+ showPlataformaDeVenta(): void
+ leerEntero(mensaje: String): int
+ seleccionarCategoria(): CategoriaExclusividad
+ seleccionarUnidadTiempo(): UnidadTiempo
CalculadoraDePrecios
-precioBaseRegular: double
-precioBaseIndividual: double
-precioBaseFastPass: double
-descuentoParaEmpleados: double
-multiplicadorExclusividad: Map
-descuentoPorDuracion: Map
+calcularPrecioRegular(CategoriaExclusividad, boolean): int
+calcularPrecioIndividual(boolean): int
+calcularPrecioTemporada(CategoriaExclusividad, UnidadTiempo, int, boolean): int
+calcularPrecioFastPass(UnidadTiempo, int, boolean): int
Taquilla
+venderTiqueteRegular(Cliente, CategoriaExclusividad, boolean): void
+venderTiqueteIndividual(Cliente, String, boolean): void
+venderTiqueteTemporada(Cliente, CategoriaExclusividad, UnidadTiempo, int, boolean): void
+venderTiqueteFastPass(Cliente, UnidadTiempo, int, boolean): void
TiqueteRegular
-usado: Boolean
+esValidoPara(idAtraccion: String): boolean
+marcarUsado(): void
+estaUsado(): boolean
TiqueteIndividual
-usado: Boolean
+esValidoPara(idAtraccion: String): boolean
+marcarUsado(): void
+estaUsado(): boolean
TiqueteTemporada
-fechaInicio: LocalDate
-fechaFin: LocalDate
+esValidoPara(idAtraccion: String): boolean
TiqueteFastPass
-fechaInicio:LocalDate
-fechaFin: LocalDate
+esValidoPara(idAtraccion: String): boolean
Tiquete
-categoriaDeExclusividad: CategoriaExclusividad
+esValidoPara(idAtraccion: String): boolean
+marcarUsado(): void
+estaUsado(): boolean
Cliente
-nombreCompleto: String
-login: String
-contrasena: String
-tiquetesDeCliente: List<Tiquete>
-dinero: int
-edad: int
-restSalud: boolean
+getDinero(): int
+setDinero(dinero: int): void
+addTiquete(tic: Tiquete): void
+getNombre(): String
+getTiquetes(): List<Tiquete>
+isRestSalud(): boolean
+getLogin(): String
+getContrasena(): String
+getEdad(): int
0..*
1
<<enumeration>>
UnidadDeTiempo
DIA
SEMANA
MES
ESTACION
ANO
<<enumeration>>
CategoriaExclusividad
BASICO
FAMILIAR
ORO
DIAMANTE





Descripción y restricciones.

El lugar de trabajo de un empleado se define a través de su turno, Cada Turno contiene la fecha, si es diurno o nocturno, y el lugar donde el empleado trabajará. Esto permite mayor flexibilidad al asignar. Asimismo, un turno puede estar asociado a una atracción, un lugar de servicio o ser nulo (servicio general). Si un empleado ya tiene un turno asignado para una fecha y jornada, el nuevo con misma fecha y jornada lo remplazaría. Al cambiar de turno, el empleado se remueve automáticamente del lugar anterior y se agrega al nuevo. Por lo que, lo unico que hay que hacer es hacer cambiar de turno a todos los empleados, (ya habiendo creado sus respectivos turnos, lo cual es tarea del administrador de empleados).

Las atracciones solo están disponibles si cumplen con temporada, empleados mínimos y no están dañadas.

Los cocineros inician con la capacitación “COCINA” por defecto. Cada cocinero tendra una productividad de 1, lo cual significa que cada vez que trabajen aumentaran la cantidad de comidas en la cafeteria, las cuales vienes por defecto 90 comidas y 90 juguetes en la tienda.

Solo empleados capacitados pueden ser asignados a atracciones de riesgo o tareas especializadas.

La temporada de una atracción se define como las estaciones y todo el año.

Las listas de empleados están separadas por tipo: Cajeros, Cocineros, Servicio General, también las Atracciones, cafeterías y tiendas, para facilitar la persistencia de los datos.

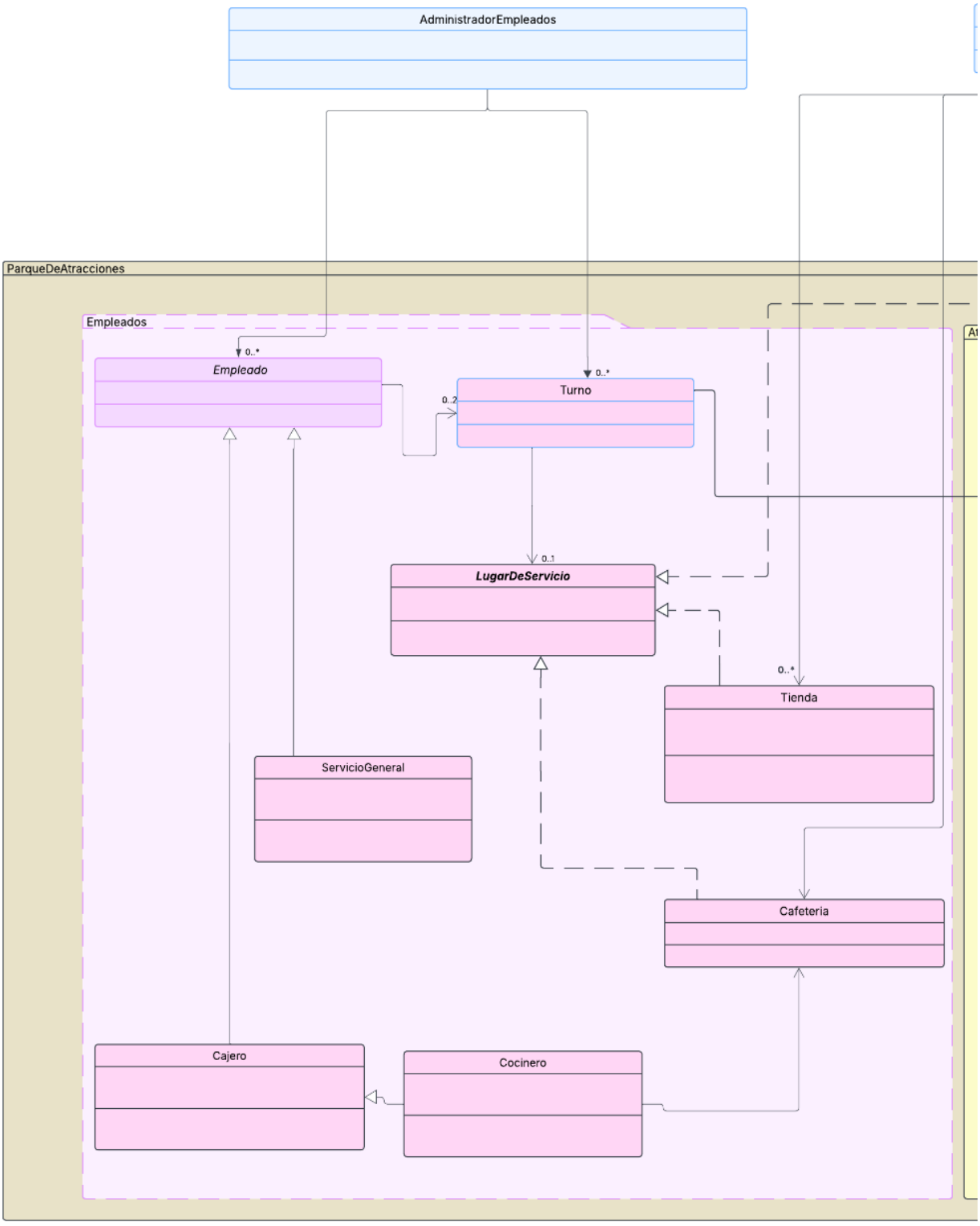
El administrador puede ejecutar un cambio de turno global,este proceso itera por cada tipo de empleado y actualiza sus asignaciones diarias según las reglas ya definidas, asegurando que el parque se organice automáticamente.

Por otro lado, la venta de tiquetes funciona de la siguiente manera. Los métodos para poder vender tiquetes se encuentran en Taquilla, la taquilla viene con una calculadora de precios, que basado en unos parámetros calculara el precio para cada tiquete particular. El Administrador de Venta de tiquetes tiene la capacidad de cambiar estos atributos. Los métodos de la taquilla se llamarán de dos maneras. Por un empleado, que podrá vender tiquetes a clientes u otros empleados o por la Plataforma de venta online, donde los clientes podrán comprar sus propios tiquetes.

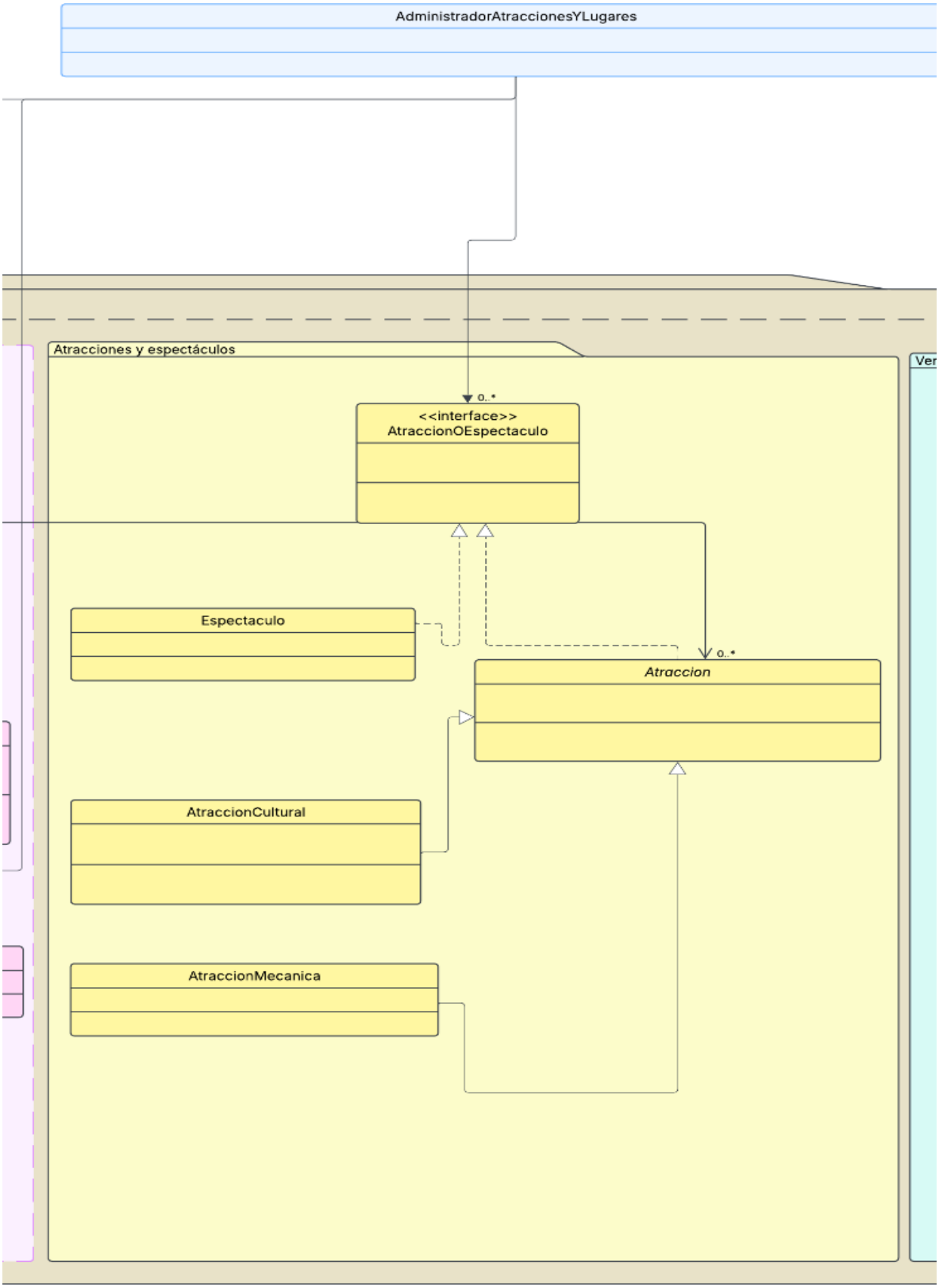
b. Un diagrama de clases de alto nivel, que incluya todas las clases del sistema y sus relaciones.

Se ha optado por mostrar el diagrama de clases de alto nivel en tres partes fundamentales

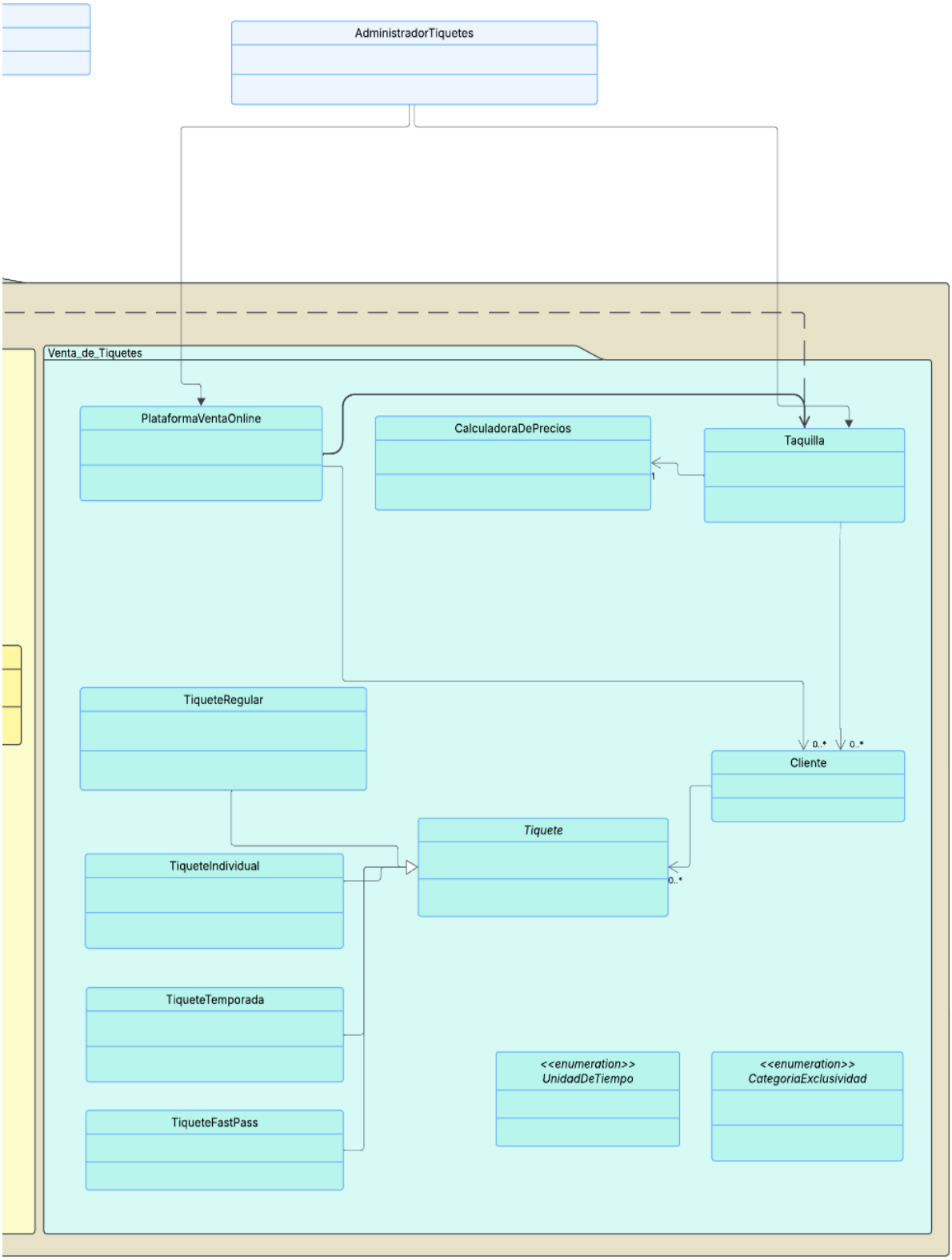
Empleados


AdministradorEmpleados
ParqueDeAtracciones
Empleados
Empleado
Turno
0..*
0..2
0..1
LugarDeServicio
Tienda
ServicioGeneral
Cafeteria
Cajero
Cocinero

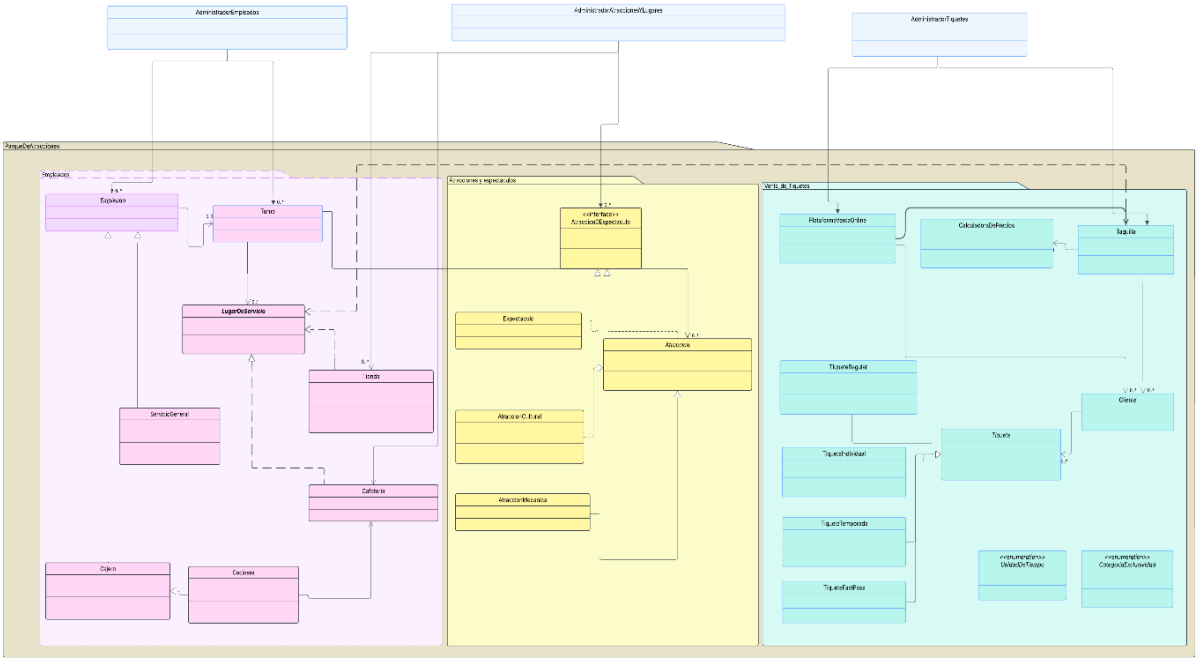
Atracciones y espectaculos



Venta de tiquetes



En conjunto se vería así :



I . Diagramas de secuencia para las funcionalidades que usted considere críticas.

Diagrama de secuencia turno empleado

Representa cómo se gestiona la asignación de turnos a los empleados, asegura que los empleados estén correctamente capacitados antes de asignarles responsabilidades, lo cual es crucial para mantener la seguridad y el buen funcionamiento del parque. Además, este diagrama ayuda a visualizar la lógica condicional (empleado capacitado o no) y muestra cómo el sistema responde a cada situación; ¿Cómo funciona? La clase AdministradorEmpleados crea un turno y verifica si el empleado está capacitado. Si lo está, se le asigna el turno y se le muestra; si no, se le añade una capacitación antes de proceder. Finalmente, el sistema permite visualizar el turno y gestionar el pago correspondiente

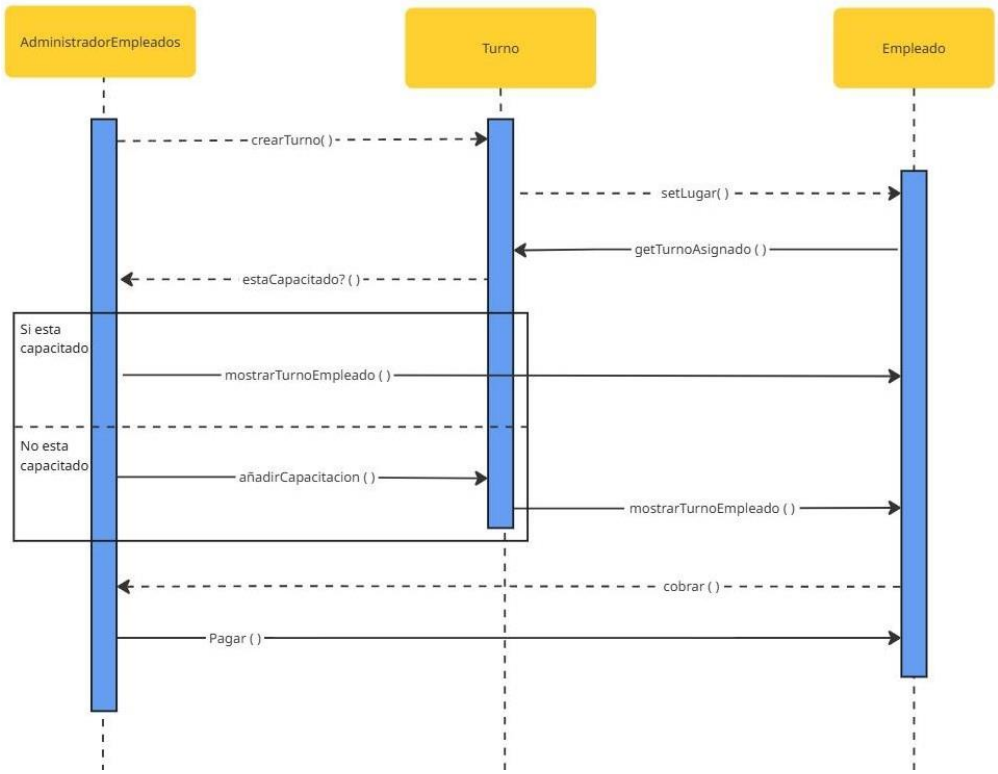


Diagrama de secuencia control acceso atracciones

Es el proceso de validación y control de acceso de los clientes a las atracciones, garantizando que solo aquellos con tiquetes válidos y que cumplan las restricciones puedan ingresar. Esto es clave para mantener la seguridad, la organización y el buen uso de los recursos del parque, evitando el ingreso no autorizado o duplicado a las atracciones. La clase cliente muestra su tiquete al EmpleadoCajero, quien solicita una validación del mismo. El sistema verifica si el tiquete ya fue usado y si el cliente cumple con las restricciones de la atracción. Según los resultados, se permite o rechaza el ingreso, y si es válido, se marca el tiquete como usado.

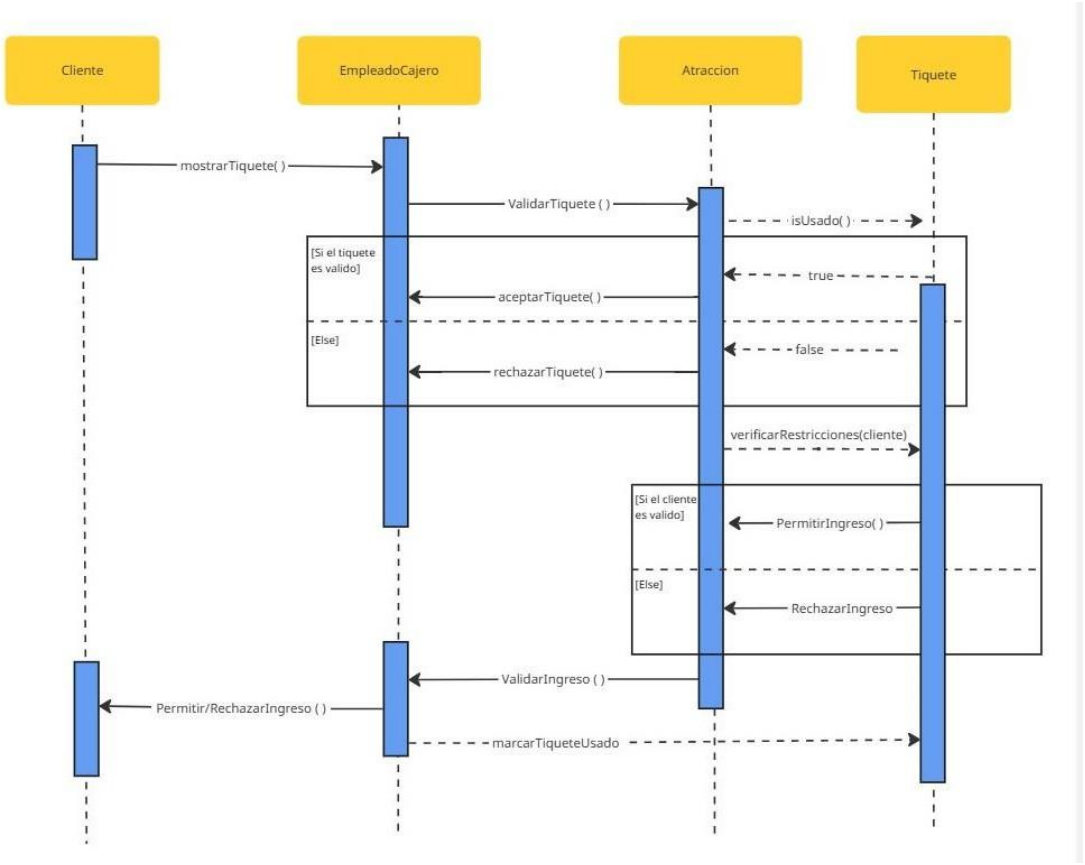
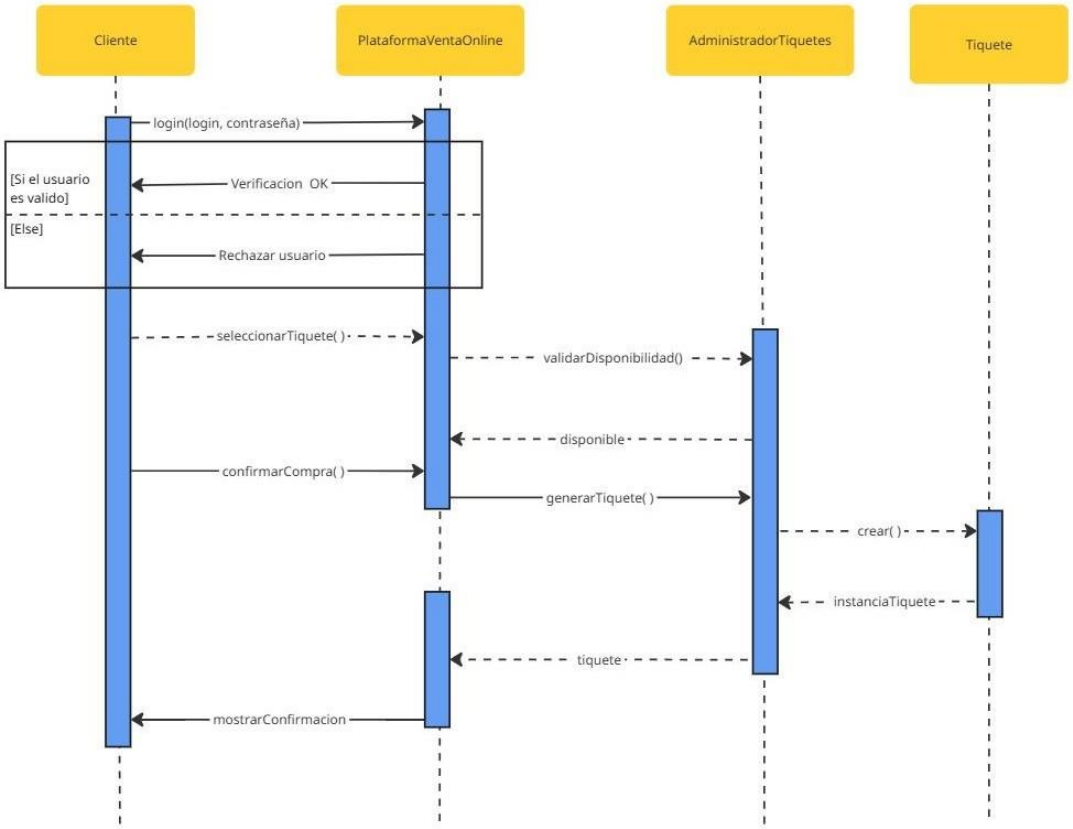


Diagrama secuencia compra de tiquetes

Este es el proceso de compra de tiquetes en la plataforma en línea, una funcionalidad fundamental que permite a los clientes adquirir sus entradas de forma remota, rápida y segura. El cliente inicia sesión en la PlataformaVentaOnline. Si sus credenciales son válidas, puede seleccionar un tiquete. La plataforma consulta al AdministradorTiquetes para verificar la disponibilidad, y si hay cupos, procede a generar el tiquete, creando una instancia. Finalmente, se muestra al cliente la confirmación de la compra junto con el tiquete genera.



c. Un diagrama de clases de alto nivel de la interfaz, que muestre qué elementos se incluye y cómo se relacionan con los elementos del dominio.

d. Sobre la implementación de la persistencia.

Para la implementación del sistema de persistencia de este proyecto, se tomó la decisión de utilizar la librería Gson, una herramienta de código abierto desarrollada por Google que permite convertir objetos Java a su representación en formato JSON y viceversa, facilitando así los procesos de serialización y deserialización de datos.

El uso de Gson responde a varias razones de peso:

Eficiencia y simplicidad: Gson permite manejar estructuras de datos complejas de forma clara y directa, lo que reduce significativamente la cantidad de código necesario para gestionar la persistencia de objetos.

Buena práctica profesional: Durante el curso se destacó la importancia de utilizar bibliotecas externas cuando estas han sido debidamente evaluadas y ofrecen una solución robusta. En este caso, Gson representa una alternativa ampliamente adoptada en la industria para el tratamiento de datos en formato JSON.

Licencia abierta: Gson se distribuye bajo la licencia Apache 2.0, lo que permite su uso libre en proyectos académicos y profesionales, siempre y cuando se reconozca su autoría y se respeten los términos de la licencia. En ese sentido, dejamos explícito que el código fuente de la librería no nos pertenece y no estamos intentando presentarlo como propio.

En resumen, el uso de Gson responde a una decisión técnica fundamentada en principios de eficiencia, buenas prácticas de desarrollo y cumplimiento legal. Esta elección nos permitió enfocar nuestros esfuerzos en la lógica del sistema, delegando la complejidad del manejo de persistencia a una herramienta confiable y bien documentada.

e. Funcionalidades clave a mayor detalle.

Funcionalidad: Cálculo dinámico de precios

La lógica para la asignación de precios en el sistema está centralizada en un componente denominado **`CalculadoraDePrecios`**. Esta clase se encarga de definir las reglas bajo las cuales se determina el valor de los diferentes tipos de tiquetes disponibles en la plataforma, considerando múltiples factores definidos en el contexto del sistema.

Cada vez que se crea una **taquilla**, se instancia automáticamente un objeto `CalculadoraDePrecios`, ya que este componente es esencial para calcular de forma coherente los precios de todas las combinaciones posibles de tiquetes. Las principales variables que se consideran en estos cálculos son:

- **Tipo de tiquete**: Se definen tres tipos principales — regular, individual y fast pass — cada uno con un precio base distinto.
- **Categoría de exclusividad**: El sistema contempla varias categorías (BÁSICO, FAMILIAR, ORO y DIAMANTE), cada una con un multiplicador asociado que modifica el precio base del tiquete regular.
- **Duración**: Para ciertos tiquetes (como fast pass o tiquetes de temporada), se aplican descuentos proporcionales según la unidad de tiempo seleccionada (día, semana, mes, estación o año).
- **Condición de empleado**: Si el comprador es un empleado, se aplica un **descuento adicional del 50%** sobre el precio final, tal como se establece en el contexto del problema.

Un aspecto importante del diseño es que **los parámetros de la `CalculadoraDePrecios` son configurables**. A través del componente **`AdministradorDeTiquetes`**, el administrador del sistema tiene la posibilidad de modificar en tiempo de ejecución los valores base, los multiplicadores por categoría y los descuentos por duración o por condición de empleado. Esto permite **ajustar las tarifas de forma dinámica**, ya sea para responder a cambios en la demanda, políticas internas, promociones o eventos especiales, sin necesidad de alterar la lógica del sistema ni reiniciar la aplicación.

Al centralizar esta lógica en una única clase y permitir su parametrización externa, el diseño favorece la mantenibilidad, la flexibilidad y la escalabilidad del sistema, alineándose con buenas prácticas de ingeniería de software orientada a objetos.

Funcionalidad: Swing y la interfaz de Usuario.

Implementación de la interfaz gráfica con Java Swing

La interfaz gráfica del sistema fue desarrollada utilizando **Java Swing**, adoptando un enfoque modular y estructurado para mantener la separación entre las diferentes funcionalidades según el rol del usuario. Toda la lógica relacionada con la GUI está contenida en el paquete `interfaz`, el cual se subdivide en tres

subpaquetes: `cliente`, `empleado` y `admin`. Esta organización permite manejar de forma independiente las interfaces específicas de cada tipo de usuario.

Cada subpaquete incluye las siguientes clases clave:

- **`VentanaLogin`**: Permite a los usuarios autenticarse según su rol.
- **`Ventana[Rol]`**: Es la ventana principal que se muestra tras un inicio de sesión exitoso, donde se presentan las opciones específicas para el tipo de usuario (por ejemplo, `VentanaCliente`, `VentanaEmpleado`, `VentanaAdmin`).
- **`PanelBotones[Rol]`**: Contiene y organiza los botones correspondientes a las acciones disponibles.
- **`Acciones[Rol]`**: Clase encargada de manejar los eventos y coordinar la lógica del sistema llamando a los métodos del administrador correspondiente (`AdministradorTiquetes`, `AdministradorAtraccionesYLugares` o `AdministradorEmpleados`).

Para mejorar la coherencia visual y la reutilización de componentes, se definieron clases personalizadas como **`MyFrame`** y **`MyButton`**, que extienden de `JFrame` y `JButton`, respectivamente. Estos componentes encapsulan estilos y configuraciones comunes, lo que permite mantener una interfaz uniforme en toda la aplicación y facilita futuras modificaciones estéticas o funcionales.

El manejo de eventos se realiza utilizando **`ActionListener`** en cada botón, con las instancias de los listeners declaradas en la clase `PanelBotones[Rol]`. Al recibir un evento, este se delega a la clase `Acciones[Rol]`, que actúa como intermediaria entre la interfaz y la lógica del dominio, invocando las operaciones correspondientes en los administradores de lógica según el tipo de usuario. Esta separación entre la visualización, la captura de eventos y la lógica de negocio mejora la legibilidad y mantenibilidad del código.

En resumen, la interfaz gráfica basada en Swing está cuidadosamente dividida por roles, construida con componentes reutilizables y organizada para garantizar una experiencia de usuario consistente, a la vez que facilita la evolución futura del sistema.

Para correr use principal en src/swing