Pablo Medina Forero – 202323276

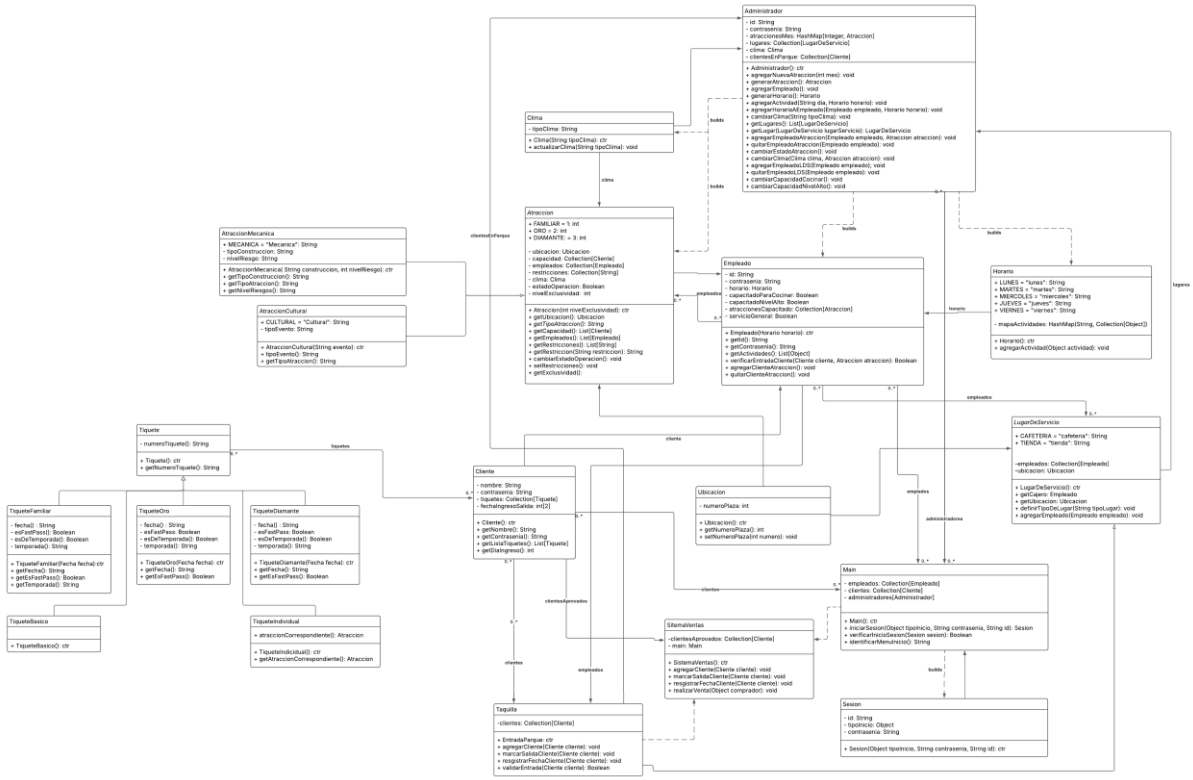
Juan David Suárez - 202223801

Sebastián Barbosa – 202316512

Análisis proyecto Atracción

1. Modelo UML

Para este proyecto se tiene en cuenta el siguiente diagrama UML:



Donde encontramos las siguientes clases con sus respectivas descripciones y atributos, junto con las relaciones que estas mantienen entre ellas.

Empezando por la clase Main, la cual se encarga de iniciar la aplicación de manera virtual y permitirle al usuario interactuar con el sistema que se está ofreciendo.

Atributos:

- empleados: Collection[Empleado]

Tiene relación con la clase Empleado, pues tiene una colección de tipo empleado la cual le permite saber el main cuales son los empleados registrados en el sistema.
- clientes: Collection[Cliente]

Tiene relación con Cliente con el fin de tener una colección con de tipo Cliente.
- administradores[Administrador]

Tiene una relación con la clase Administrador para poder tener en cuenta quienes son los administradores

Métodos:

+ Main(): ctr

Constructor de Main
+ iniciarSesion(Object tipoInicio, String contrasenia, String id): Sesion

Inicia sesión el usuario y devuelve una clase tipo Sesión.
+ verificarInicioSesion(Sesion sesion): Boolean

Verifica si la Sesion existe y devuelve un booleano
+ identificarMenuInicio(): String

Cuando idenfica que la Sesion existe, entonces busca el menu de opciones adecuado para el usuario dependiendo de su tipo.

Por otro lado, tenemos la clase Sesion, la cual se encarga de ayudar a generar los menus necesarios dependiendo del tipo de usuario.

Atributos:

- id: String

Id se la sesion
- tipoInicio: Object

Tipo de inicio dependiendo del tipo de usuario.
- contrasenia: String

Contraseña de la sesion.

Metodos:

+ Sesion(Object tipoInicio, String contrasenia, String id): ctr

Solo cuenta con su constructor.

Seguimos con la clase SistemaVentas, esta se encarga de verificar y resgistrar las ventas de los clientes.

Atributos:

-clientesAprovados: Collection[Cliente]

Una coleccion que contiene Cliente

- main: Main

Tiene la relacion con el main, pues asi puede acceder a la sesion activa en este momento.

Metodos:

+ SistemaVentas(): ctr

Construye el sistema de ventas
+ agregarCliente(Cliente cliente): void

Agrega el cliente a la lista de ventas
+ marcarSalidaCliente(Cliente cliente): void

Este ayuda cuando el cliente no completa la venta
+ resgistrarFechaCliente(Cliente cliente): void

Registra la fecha de la venta en el cliente
+ realizarVenta(Object comprador): void

Realiza el cobro al cliente y confirma que eventualmente este puede, recordemos que los empleados del parque tienen descuentos, por eso el parámetro es comprados, pues dependiendo del tipo del comprador el valor del tiquete cambiara.

Ahora para la clase de Tiquete, esta esta conectada con la clase Cliente ya que cada cliente puede tener la cantidad de tiquetes que quera, y la clase Tiquete es la super clase de todos los tipos de tiquetes que el parque maneja, esta clase tiene el atributo

- numeroTiquete(): String

cada tiquete tiene su respectivo número (numeroTiquete) para así poder diferenciarlos entre ellos. Y esta clase simplemente cuenta con dos métodos, su constructor y get:

+ Tiquete(): ctr

+ getNumeroTiquete(): String

Donde así se puede identificar que tiquete es.

La clase tiquete tiene 5 subclases las cuales son: TiqueteFamiliar, TiqueteOro, TiqueteDiamante, TiqueteBasico y TiqueteIndividual. El primer tipo de tiquete (el TiqueteFamiliar) al igual que el TiqueteOro y el TiqueteDiamante, tienen como atributos:

- fecha() : String

- esFastPass(): Boolean

- esDeTemporada(): Boolean

- temporada(): String

En donde la fecha es la fecha para la cual es útil ese tiquete, un booleano que responde a la pregunta si es FastPass o no, un booleano que dice si es un tiquete de temporada y un String que dice de cuanto tiempo es la temporada. Los métodos de estas tres clases son sus respectivos constructores y los gets de sus métodos:

+ TiqueteFamiliar(Fecha fecha):ctr

+ getFecha(): String

+ getEsFastPass(): Boolean

+ getEsDeTemporada

+ getTemporada(): String

getFecha para saber para cuándo es el tiquete, getEsFastPass para saber si tiene este beneficio, getEsDeTemporada y getTemporada para así poder saber cuánto durara el tiquete.

La clase de TiqueteBasico es diferente ya que esta no permite el uso de ninguna atracción y no tiene fecha definida, por lo que se separa de los otros tiquetes, quedando con un único método que es su constructor.

+ TiqueteBasico(): ctr

Y la ultima subclase de Tiquete es el TiqueteIndividual, donde este simplemente permite el uso de la atracción para la que se compró, esta clase tiene como atributo:

+ atraccionCorrespondiente(): Atraccion

Que es la atraccion a la que permite entrar el tiquete, y tiene como método:

+ TiqueteIndicidual(): ctr

+ getAtraccionCorrespondiente(): Atracción

Que son su constructor y el getAtraccion que da la atraccion que tiene definida.

La clase administrador está relacionada con el clima, ya que un administrador es el que puede actualizar el clima, también está relacionado con la atracción ya que puede generar y cerrar atracciones, al igual que lugares de servicio. Esta clase tiene los métodos:

- id: String

- contrasenia: String

- atraccionesMes: HashMap[Integer, Atraccion]

- lugares: Collection[LugarDeServicio]

- clima: Clima

- clientesEnParque: Collection[Cliente]

Donde el id es diferente para cada administrador y así diferenciarlos, cada administrador tiene su propia contrasaña por seguridad, tienen el registro de las atracciones hasta el momento y de los lugares de dervicio, además de tener el registro del clima y de los clientes que tiene el paruqe en ese momento. Y esta clase tiene los siguientes métodos:

+ Administrador(): ctr

+ agregarNuevaAtraccion(int mes): void

+ generarAtraccion(): Atraccion

+ agregarEmpleado(): void

+ generarHorario(): Horario

+ agregarActividad(String dia, Horario horario): void

+ agregarHorarioAEmpleado(Empleado empleado, Horario horario): void

+ cambiarClima(String tipoClima): void

+ getLugares(): List[LugarDeServicio]

+ getLugar(LugarDeServicio lugarServicio): LugarDeServicio

+ agregarEmpleadoAtraccion(Empleado empleado, Atraccion atraccion): void

+ quitarEmpleadoAtraccion(Empleado empleado): void

+ cambiarEstadoAtraccion(): void

+ cambiarClima(Clima clima, Atraccion atraccion): void

+ agregarEmpleadoLDS(Empleado empleado); void

+ quitarEmpleadoLDS(Empleado empleado): void

+ cambiarCapacidadCocinar(): void

+ cambiarCapacidadNivelAlto(): void

En estos métodos, se encuentra el constructor respectivo, los get de los atributos que ya se mencionaros (para poder conocer la información), además de métodos de agregar nuevos empleados, atracciones, lugares de servicio, capacidades de las atracciones y cambiar el clima.

La clase LugarDeServicio es una clase que tiene 2 tipos, pues tiene cafeteria y tienda, sin embargo, tiene una tercera que es taquilla, esta tiene todo lo de un lugar de servicio, pero tiene otras condiciones que hacer que relacione como herencia, lo veremos luego (recordemos que sin una taquilla es una tienda de tiquetes).

Atributos:

+ CAFETERIA = "cafeteria": String
+ TIENDA = "tienda": String
Variables estaticas que tienen importancia para el constructor
- empleados: Collection[Empleado]

Lista de Empleado que se encuentran trabajando en el lugar.
- ubicacion: Ubicacion

Clase de Ubicacion que permite identificar donde se encuentra el lugar de servicio.

Metodos:

+ LugarDeServicio(): ctr

Contruye el lugar de servicio
+ getCajero: Empleado

Brinda el empleado que hace de cajero en aquel lugar
+ getUbicacion: Ubicacion

Brinda Ubicacion de lugar.
+ definirTipoDeLugar(String tipoLugar): void

Define el tipo de lugar.
+ agregarEmpleado(Empleado empleado): void

Agregra un empleado a la lista de empleados que trabajan ahi.

Tenemos la clase Atraccion, la cual es abstracta pues hay 2 diferentes tipos de atracciones en si.

Atributos:

FAMILIAR = 1: int

ORO = 2: int

DIAMANTE = 3: int

ubicacion: Ubicacion

capacidad: Collection[Cliente]

empleados: Collection[Empleado]

restricciones: Collection[String]

clima: Clima

estadoOperacion: Boolean

nivelExclusividad: int

Metodos:

Atraccion(int nivelExclusividad): ctr

Construye la atracción con un nivel de exclusividad.

getUbicacion(): Ubicacion

Brinda la ubicación de la atracción.

getTipoAtraccion(): String

Devuelve el tipo de atracción según su nivel de exclusividad.

getCapacidad(): List[Cliente]

Brinda la lista de clientes que pueden ingresar a la atracción.

getEmpleados(): List[Empleado]

Devuelve la lista de empleados asignados a la atracción.

getRestricciones(): List[String]

Brinda la lista de restricciones de la atracción.

getRestriccion(String restriccion): String

Devuelve una restricción específica de la atracción.

cambiarEstadoOperacion(): void

Cambia el estado de operación de la atracción.

setRestricciones(): void

Establece las restricciones de la atracción.

getExclusividad(): int

Devuelve el nivel de exclusividad de la atracción.

Clase Cliente

Atributos:

- nombre: String

- Propósito: Almacena el nombre del cliente, fundamental para identificarlo en el sistema.

-contrasenia: String

- Propósito: Guarda la contraseña del cliente, utilizada para la autenticación y la seguridad de la cuenta.

-tiquetes: Collection[Tiquete]

- Propósito: Contiene la colección de tiquetes asociados al cliente.
- Uso: Permite gestionar los diferentes tipos de tiquetes (TiqueteFamiliar, TiqueteOro, TiqueteDiamante, TiqueteBasico o TiqueteIndividual).
- Nota: Al trabajar con esta colección, puede requerirse realizar un cast para acceder a métodos o atributos específicos de cada subclase.

- fechaIngresoSalida: int[2]

- Propósito: Arreglo que registra dos enteros: la fecha de ingreso y la fecha de salida del parque.
- Uso: Facilita el seguimiento del tiempo de permanencia del cliente, importante para análisis de flujo y control de accesos.

Métodos:

-Cliente() : str

- Propósito: Constructor de la clase, inicializa un nuevo objeto Cliente.
- Función: Asigna valores por defecto, inicializa la colección de tiquetes y configura los parámetros básicos.

-getNombre() : String

- Propósito: Devuelve el nombre almacenado en el atributo “nombre”.
- Función: Utilizado para visualizar y validar la identidad del cliente.

- getContrasenia() : String

- Propósito: Retorna la contraseña del cliente.
- Función: Útil para la autenticación, considerando que en una implementación real se aplicarían medidas de seguridad como encriptación.

-getListaTiquetes() : List[Tiquete]

- Propósito: Proporciona acceso a la lista de tiquetes asociados al cliente.
- Función: Permite iterar sobre los tiquetes para validar accesos o aplicar reglas específicas (por ejemplo, verificar si un tiquete es de temporada o incluye fast pass).
- Nota: Puede requerirse realizar un cast sobre cada elemento según su tipo.

- getDiaIngreso() : int

- Propósito: Extrae el día de ingreso del cliente al parque.
- Función: Generalmente se retorna el primer elemento del arreglo “fechaIngresoSalida”, útil para validar la duración de la visita o generar estadísticas.

Clase Empleado

Atributos:

-id: String

- Propósito: Identificador único del empleado.
- Uso: Permite la diferenciación y búsqueda del empleado dentro del sistema.

- contraseña: String

- Propósito: Almacena la contraseña del empleado.
- Uso: Esencial para la autenticación y para mantener la seguridad en el acceso al sistema de administración.

- horario: Horario

- Propósito: Objeto que representa la agenda o calendario de actividades asignadas al empleado.
- Uso: Permite gestionar las actividades diarias y semanales.
- Nota: En caso de requerirse detalles específicos, puede ser necesario realizar un cast sobre el objeto.

- capacitadoParaCocinar: Boolean

- Propósito: Indica si el empleado está capacitado para labores de cocina.
- Uso: Ayuda a asignar al empleado a áreas que requieran preparación de alimentos, validando que cumpla con los requisitos.

- capacitadoNivelAlto: Boolean

- Propósito: Señala si el empleado posee habilidades o capacitación avanzada para tareas especializadas.
- Uso: Importante para asignar funciones que demanden mayor experiencia o conocimientos específicos.

-atraccionesCapacitado: Collection[Atraccion]

- Propósito: Contiene las instancias de Atracción en las que el empleado está calificado para trabajar.
- Uso: Permite determinar en qué atracciones se le puede asignar, fundamental para la correcta asignación de personal.

-servicioGeneral: Boolean

- Propósito: Indica si el empleado está disponible para tareas de servicio general.

- Uso: Ofrece flexibilidad para cubrir roles que no requieren una capacitación específica.

Métodos:

- Empleado(Horario horario) : str

- Propósito: Constructor de la clase Empleado.
- Función: Inicializa un nuevo objeto Empleado asignándole un objeto Horario y estableciendo valores iniciales en los atributos.

- getId() : String

- Propósito: Devuelve el identificador único del empleado.
- Función: Facilita la identificación, búsqueda y manejo interno del empleado en el sistema.

- getContrasenia() : String

- Propósito: Retorna la contraseña del empleado.
- Función: Utilizado en procesos de autenticación; se recomienda el uso de encriptación en implementaciones reales.

- getActividades() : List[Object]

- Propósito: Devuelve una lista de actividades asignadas al empleado, extraídas de su objeto Horario.
- Función: Permite acceder a las tareas o eventos programados.
- Nota: La lista retorna objetos genéricos, por lo que puede requerirse realizar un cast para operar sobre una actividad específica.

- verificarEntradaCliente(Cliente cliente, Atraccion atraccion) : Boolean

- Propósito: Valida si un cliente puede acceder a una atracción.
- Función: Analiza, por ejemplo, si el cliente posee un tiquete adecuado y si la atracción se encuentra operativa.

2. Descripción de las Restricciones del proyecto:

A nivel de negocio y lógica del parque, se identifican las siguientes restricciones principales:

1. Estado de Operación de la Atracción:
    a. Una atracción no puede admitir clientes si estadoOperacion es false.
    b. El clima adverso (tipoClima inadecuado) puede forzar a cambiar el estado de operación a false.
2. Capacidad de la Atracción:
    a. Se debe validar que la atracción no exceda el número máximo de clientes en su lista capacidad.
    b. Si se alcanza el límite, no se pueden agregar más clientes.
3. Validación de Tiquetes:
    a. Un cliente debe poseer un tiquete válido para la atracción que desea ingresar.
    b. Se verifica a través de verificarEntradaCliente(Cliente, Atraccion) en la clase Empleado o validarEntrada(Cliente) en la clase Taquilla.
4. Gestión de Empleados y Capacitación:
    a. Solo empleados con la capacitación adecuada (capacitadoParaCocinar, capacitadoNivelAlto, etc.) pueden desempeñar ciertas funciones o trabajar en atracciones específicas.
    b. Se deben respetar las relaciones de asignación y remover a un empleado de una atracción cuando no cumpla los requisitos.
5. Restricciones de Clima:
    a. El Administrador puede cambiar el clima con cambiarClima(...).
    b. Ciertas atracciones (por ejemplo, mecánicas) pueden cerrar automáticamente si el clima no es apto.
6. Ingreso y Salida de Clientes:
    a. La clase Taquilla (o su constructor EntradaParque) y SistemaVentas gestionan el registro de entrada y salida de clientes, junto con la fecha de ingreso/salida.
    b. No se debe permitir la entrada a un cliente si no se ha registrado correctamente la salida anterior, o si no posee un tiquete válido.

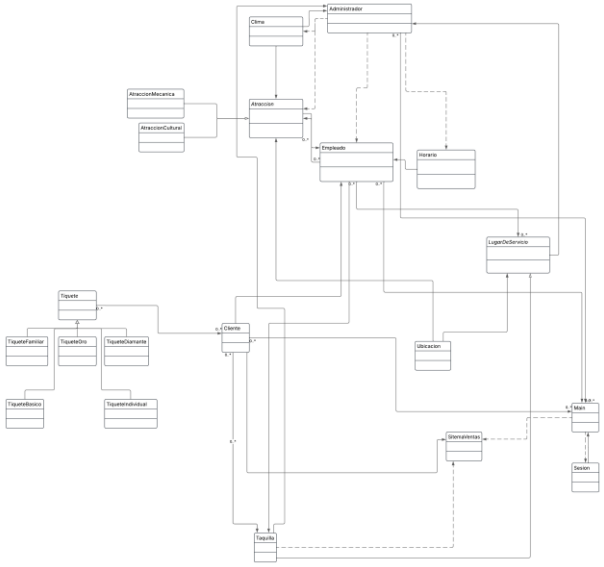
3. Descripción de lo que demuestra cada programa de prueba:

Se proponen diversos programas de prueba para validar la funcionalidad del sistema:

1. Prueba de Creación y Configuración de Atracciones

    a. Objetivo: Verificar la correcta instanciación de Atraccion, incluyendo subclases (mecánica, cultural), y asignación de valores a nivelExclusividad, ubicacion, etc.
    b. Demuestra: Uso de constructores y configuración inicial (restricciones, estado, clima).
2. Prueba de Asignación de Empleados
    a. Objetivo: Validar que un Administrador pueda asignar empleados a una atracción y/o lugar de servicio.
    b. Demuestra: Métodos como agregarEmpleadoAtraccion(...) y la verificación de capacitación (por ejemplo, capacitadoParaCocinar).
3. Prueba de Verificación de Entrada (Empleado / Taquilla)
    a. Objetivo: Comprobar que se valide si un cliente puede ingresar a una atracción.
    b. Demuestra: Uso de verificarEntradaCliente(...) en Empleado y validarEntrada(Cliente) en Taquilla, revisando los tiquetes y el estado de la atracción.
4. Prueba de Clima y Estado de Operación
    a. Objetivo: Cambiar el clima con Administrador y observar cómo impacta el estado operativo de la atracción.
    b. Demuestra: Llamada a cambiarClima(...) y cambiarEstadoOperacion(), validando la lógica que cierra o abre atracciones.
5. Prueba de Manejo de Horarios
    a. Objetivo: Añadir actividades a través de Horario y asociarlas a empleados.
    b. Demuestra: Uso de agregarActividad(...) y la necesidad de *cast* si se requiere manipular tipos específicos de actividades.
6. Prueba de Ingreso y Salida de Clientes (Taquilla / SistemaVentas)
    a. Objetivo: Registrar la entrada y salida de clientes, así como la venta de tiquetes.
    b. Demuestra: Métodos agregarCliente(...), marcarSalidaCliente(...), registrarFechaCliente(...), realizarVenta(...). Verifica el control de acceso y la persistencia de datos de clientes.

Para finalizar, dejamos el diagrama de dominio:


Administrador
Clima
AtraccionMecanica
AtraccionCultural
Atraccion
Empleado
Horario
LugarDeServicio
Tiquete
Cliente
TiqueteFamiliar
TiqueteOro
TiqueteDiamante
TiqueteBasico
TiqueteIndividual
Ubicacion
Main
Sistemaventas
Sesion
Taquilla