Juan Fernando Vanegas-202320303

Juan Felipe Salazar-202325542

Analisis:

Contexto

Se busca crear una aplicación para apoyar operaciones administrativas en un parque de atracciones que incluye diferentes tipos de atracciones y espectáculos en vivo. En cuanto a las atracciones, hay dos tipos de estas, las mecánicas y las culturales. Por último, el sistema debe tener 3 funcionalidades principales, tener un catálogo con las diferentes atracciones, gestionar empleados y labores de atracciones y finalmente permitir venta de tiquetes para los clientes.

Requerimientos funcionales:

Atracciones y espectáculos:

- Ambas atracciones tienen una ubicación fija, y se diferencian entre mecánicas y culturales.
- Los espectáculos pueden ocurrir en cualquier ubicación, y operan bajo horarios y fechas específicas.
- Las atracciones tienen una capacidad máxima y una cantidad mínima de empleados a cargo de esta (int), además cada atracción tiene ciertas restricciones. También cada atracción tiene definido máximos y mínimos de altura y peso para los clientes que desean usarla.
- Las atracciones mecánicas tienen un nivel de riesgo (String) medio o alto, y dependiendo de esto se le asignan los empleados.
- Además, pueden tener una serie de contraindicaciones médicas (Lista de String).
- Esto solo era para las atracciones mecánicas, para las culturales solo tienen restricciones (atributos) de edad.
- Ambos tipos de atracciones deberían poder abrir o cerrar dependiendo del clima que haya.
- Algunos espectáculos y atracciones son de temporada, osea que solo están disponibles en cierto momento del año, y el periodo de tiempo que abren puede ser en meses o un solo día como un espectáculo especial.
- Por último y muy importante, el único que puede modificar la información de atracciones y espectáculos es el administrador del parque, y tanto los clientes como empleados pueden consultar la información de atracciones y

espectáculos.

Empleados:

- Hay varios tipos de empleado (una clase abstracta), empleado de atracción, empleado regular, empleado servicios generales, empleado cocinero y empleado administrador.
- Los empleados pueden operar solo en 3 diferentes lugares de servicio(clase), cafeterías tiendas y taquillas
- En cuanto a estos lugares, cada uno tiene atributos como cajero asociado.
- En el caso de taquilla o atracciones, el cajero revisará los tiquetes.
- Ahora las cafeterías, tienen asociados cocineros capacitados. Quienes pueden también operar en caja, pero un empleado común no puede operar en la cocina.
- Hay empleados conocidos como de Servicio general, quienes no estarán asociados a un turno.
- Los empleados de atracciones tienen capacidad media para atracciones de riesgo medio o capacidad específica de atracción, para manejo de atracciones de riesgo alto.
- Por último, el administrador es el encargado de repartir turnos y cambiar información de los empleados.
- Los empleados pueden consultar sus turnos y tareas del día, y hay dos tipos de turno (clase), de apertura y de cierre, un empleado puede tener asignado ambos turnos, uno o ninguno

Tiquetes

- Existen cuatro tipos de tiquete(Abstract) principales, Familiar Oro, diamante y básico, y un cliente puede tener cuantos clientes desee.
- Los tiquetes Familiar sólo permiten acceso a atracciones de exclusividad familiar, Los de oro a familiar y exclusividad Oro y diamante permite el acceso a todas las atracciones.
- Para los tiquetes regulares, sirven para un único día, entonces después de ingresar al parque al momento de salir, estos ya no son útiles más.
- Para los tiquetes básicos, solo permite el acceso al parque y ya, además no tienen una fecha relacionada y son válidos al momento de ingresar al parque.
- Adicionalmente existe otro tiquete que es de temporada, que permite acceso al parque ilimitado en un rango de fechas estacionales. Este tipo de tiquete también se divide en básico, familiar, oro y diamante.
- Aparte de los tiquetes, existen las entradas individuales, que permiten ingresar a una atracción una única vez.
- Los tiquetes y entradas pueden tener una funcionalidad FastPass o no (clase), pero esto solo se puede comprar para un día específico, y no por un rango de fechas como los distintos tipos de tiquetes.
- Los clientes pueden tener un atributo como un boolean, que defina si ya fue utilizado o no.
- Existen cajeros de taquillas (empleados), quienes registran los tiquetes y lo asocian a su comprador.
- Finalmente, los empleados y el administrador pueden comprar tiquetes

para ellos también, pero estos tendrán un descuento especial por ser empleados.
- Adicionalmente, en el UML se especifica pero el tiquete básico le dimos un precio de 20, familiar de 30, oro de 40 y diamante de 50.
- Para los tiquetes de temporada, tiene un precio estándar de 200 + la exclusividad del tiquete, por ejemplo si se compra un tiquete de temporada oro, tendría un precio de 240.
- Los fastpass tienen un precio de 15.
- Las entradas dependen del nivel de atracción alto o medio, si es medio valen 10 y si es alto valen 20.

Requerimientos adicionales

- Debe tener una implementación de persistencia, donde tiquetes usados por usuarios, clientes empleados atracciones e información que guarde la información.
- Todos los usuarios(abstracto) como empleados o clientes deben tener un login y una contraseña, que se pueda verificar si es correcta o no.

Explicación de Clases del Proyecto

Parque

Modelo::Parque
-direccion: String -nombre: String -capacidad: int -abierto: boolean -usuarios: HashMap<String, Usuario> -atracciones: List<Atraccion> -espectaculos: List<Espectaculo> -lugaresServicios: List<LugarServicio>
+Parque(String direccion, String nombre, int capacidad): ctor +agregarUsuario(Usuario usuario): void +getNombre(): String +getCapacidad(): int +getDireccion(): String +getAbierto(): boolean +agregaratraccion(Atraccion atraccion): void +agregarEspectaculo(Espectaculo espectaculo): void +agregarLugar(LugarServicio lugar): void +getUsuarios(): Collection<Usuario> +getEmpleados(): List<Empleado> +getEmpleado(List<Empleado> empleados, int possicion): Empleado +usuarioAdministrador(Usuario usuario): boolean +autenticarIngreso(String login, String password): boolean +getAtracciones(): List<Atraccion> +getEspectaculo(): List<Espectaculo> +getLugares(): List<LugarServicio> +setNombre(String Nombre): void +setCapacidad(int capacity): void +setDireccion(String direct): void +abrirParque(): void +cerrarParque(): void +eliminarUsuario(Usuario usuario): boolean +eliminarAtraccion(Atraccion atraccion): void +eliminarEspectaculo(Espectaculo espectaculo): void +eliminarLugar(LugarServicio lugar): void +cargarData(): void +salvarData(): void +AsignarTurno(Empleado empleado, String turno): void +AsignarLabor(Empleado empleado, String labor, LugarServicio lugar, Usuario usuario): void +retirarTurnoEmpleado(Empleado empleado, String turno, Usuario usuario): void +asignarLugarEmpleado(Empleado empleado, LugarServicio lugar, Usuario usuario): void +asignarAtraccionEmpleado(Empleado empleado, AtraccionMecanica atraccion, Usuario usuario): void +registrarCompraTiquetesTaquilla(Usuario usuario, List<Tiquete> tiquetes, List<FastPass> fast, Taquilla taquilla): void +IngresarAtraccion(Atraccion atraccion, Tiquete tiquete): void +registrarOtraCompra(CompraServicio compra): void +registrarEntrada(Tiquete tiquete): boolean +registrarSalida(Tiquete tiquete, String fechaHoy): void

La clase Parque representa el modelo central del sistema, encargado de gestionar el estado y la lógica de funcionamiento de un parque de diversiones. Esta clase almacena atributos como el nombre, dirección, capacidad máxima y estado de apertura del parque, así como las colecciones de usuarios, atracciones, espectáculos y lugares de servicio.

Responsabilidades:

- Gestionar el registro, eliminación y autenticación de usuarios (clientes, empleados y administradores).
- Administrar las listas de atracciones mecánicas, espectáculos y lugares de servicio (como tiendas o taquillas).
- Permitir operaciones exclusivas para administradores, como asignar turnos, labores, atracciones o lugares a empleados.
- Registrar compras de tiquetes y fast pass tanto online como en la taquilla, aplicando tarifas diferenciadas para empleados.
- Validar el ingreso de usuarios a atracciones y controlar el uso de tiquetes y fast pass según la fecha.
- Manejar la entrada y salida de usuarios del parque.
- Implementar persistencia del estado del parque mediante serialización a archivos binarios (parque.bin), permitiendo guardar y cargar el estado completo del sistema.

Métodos Clave:

- agregarUsuario, eliminarUsuario: administración de usuarios.
- abrirParque, cerrarParque: control del estado del parque.
- asignarTurno, asignarLabor, asignarLugarEmpleado, asignarAtraccionEmpleado: operaciones administrativas.
- registrarCompraTiquetesOnline, registrarCompraTiquetesTaquilla: gestión de ventas de tiquetes.
- registrarEntrada, registrarSalida: control de acceso al parque.
- salvarData1, cargarData1: persistencia del estado del parque.

Atraccion

La clase abstracta Atraccion representa el modelo base para todas las atracciones del parque de diversiones. Esta clase encapsula los atributos y comportamientos comunes a las distintas atracciones, ya sean mecánicas, espectáculos u otras, y permite una extensión clara mediante herencia.

Responsabilidades:

- Almacenar y gestionar la información esencial de una atracción, como nombre, capacidad, ubicación, exclusividad, temporada y tipo.

- Controlar el estado de apertura o cierre de la atracción.
- Administrar la asignación de empleados específicos (de tipo EmpleadoAtracciones) y cajeros.
- Proveer métodos para agregar o eliminar empleados, y para establecer configuraciones operativas básicas.

Atributos Clave:

- nombre, capacidad, ubicacion: información básica de identificación y localización.
- empleadosMin: número mínimo de empleados requeridos para operar.
- exclusividad, temporada: condiciones de operación (por ejemplo, restricciones por edad o temporada del año).
- abierta: indica si la atracción está operativa o no.
- tipo: categoría de la atracción.
- cajero: empleado asignado para gestionar el punto de cobro.
- empleados: lista de empleados asignados a la atracción.

Métodos Clave:

- Getters y setters para atributos operativos (por ejemplo, getNombre, setCapacidad, getExclusividad, etc.).
- abrirAtraccion / cerrarAtraccion: control de disponibilidad de la atracción.
- agregarEmpleado, EliminarEmpleado: gestión del personal asignado.
- sacarCajero, setCajero: gestión del cajero asociado.

AtraccionMecanica

La clase AtraccionMecanica representa un tipo específico de atracción del parque: las atracciones mecánicas. Hereda de la clase abstracta Atraccion e incorpora atributos y comportamientos adicionales propios de este tipo de atracciones, como restricciones de altura y nivel de riesgo.

Responsabilidades:

- Definir restricciones de seguridad adicionales, como altura mínima y máxima.
- Establecer el nivel de riesgo asociado a la atracción.
- Gestionar una lista personalizada de restricciones adicionales (por ejemplo, salud, edad, etc.).

Atributos Clave:

- alturaMax, alturaMin: determinan las alturas permitidas para los usuarios de la atracción.

- nivelRiesgo: clasifica el nivel de peligrosidad o intensidad (por ejemplo: bajo, medio, alto).
- restricciones: lista de restricciones adicionales que puedan aplicar a los usuarios.

Métodos Clave:

- Getters y setters para alturaMin, alturaMax y nivelRiesgo.
- agregarRestriccion: permite añadir nuevas restricciones a la lista.
- eliminarRestriccion: elimina una restricción específica, si está presente.

AtraccionCultural

La clase AtraccionCultural modela una atracción de tipo cultural dentro del parque. Extiende la clase abstracta Atraccion e introduce una restricción particular basada en la edad mínima requerida para ingresar a la atracción.

Responsabilidades:

- Representar atracciones culturales del parque (como museos, exhibiciones, recorridos históricos, etc.).
- Restringir el acceso según una edad mínima definida.

Atributos Clave:

- edadMin: define la edad mínima que debe tener un visitante para poder ingresar a la atracción.

Métodos Clave:

- getEdadMin(): devuelve la edad mínima requerida.
- setEdadMin(int edad): permite modificar la edad mínima establecida.

Usuario

La clase abstracta Usuario representa a cualquier persona registrada en el sistema del parque, ya sea visitante o personal. Gestiona la información básica del usuario y su interacción con el sistema a través de tiquetes, fast passes y compras.

Responsabilidades:

- Modelar datos comunes a todos los tipos de usuario del sistema.
- Almacenar credenciales de acceso y características personales relevantes (edad y altura).
- Gestionar la relación del usuario con tiquetes, fast passes y compras.

Atributos Clave:

- login: nombre de usuario para autenticación.

- password: contraseña asociada.
- tipoUsuario: define el rol del usuario en el sistema (por ejemplo, visitante o empleado).
- edad y altura: características físicas utilizadas para restricciones en atracciones.
- tiquetesUsar / tiquetesUsados: listas que permiten llevar control de los tiquetes activos y los ya utilizados.
- fastPasses / fastPassesUsados: listas similares, pero para fast passes.
- historialCompra: lista de objetos Compra, que representan el historial de compras del usuario.

Métodos Clave:

- Métodos getters para todos los atributos, lo que permite consultar la información del usuario.
- AgregarTiquete(Tiquete), UsarTiquete(Tiquete): permiten agregar y marcar como usados los tiquetes, asegurando consistencia entre las listas.
- agregarFastPass(FastPass), usarFastPass(FastPass): gestionan los fast passes de forma análoga a los tiquetes.
- agregarCompra(Compra): registra una compra en el historial del usuario.

Cliente

La clase Cliente es una subclase concreta de Usuario que representa a los visitantes del parque de diversiones. Su objetivo es encapsular a los usuarios que adquieren tiquetes y fast passes, y que hacen uso de las atracciones.

Responsabilidades:

- Representar a los clientes o visitantes del parque.
- Heredar y utilizar todas las funcionalidades de la clase Usuario, como la gestión de tiquetes, fast passes y compras.

Atributos Clave:

- No introduce nuevos atributos. Utiliza todos los atributos heredados de Usuario, como login, password, edad, altura, tiquetesUsar, etc.

Métodos Clave:

- Solo cuenta con un constructor, que define el tipo de usuario como "cliente".

Administrador

La clase Administrador extiende a Usuario y representa a los usuarios encargados de gestionar y coordinar al personal dentro del parque de diversiones. Se encarga de asignar labores, turnos y ubicaciones tanto en atracciones como en lugares de servicio, permitiendo mantener el

parque operando de manera organizada y eficiente.

Responsabilidades:

- Asignar turnos a empleados (excepto a los de servicios generales).
- Asignar labores específicas según el tipo de empleado y el lugar asignado.
- Asignar empleados a atracciones o lugares de servicio.
- Retirar turnos, labores o ubicaciones previamente asignadas.
- Validar que las asignaciones se hagan correctamente según el rol del empleado.

Atributos Clave:

- Hereda todos los atributos de la clase Usuario (como login, contraseña, edad, altura, etc.).
- No introduce nuevos atributos propios.

Métodos Clave:

- AsignarTurno(Empleado empleado, String turno): Asigna turnos a empleados, validando que no sean de servicios generales.
- AsignarLabor(Empleado empleado, String labor, LugarServicio lugar): Asigna una labor a un empleado con validación del tipo de empleado y del lugar.
- retirarTurnoEmpleado(Empleado empleado, String turno): Retira un turno asignado, si corresponde.
- asignarLugarEmpleado(Empleado empleado, LugarServicio lugar): Asigna un lugar de servicio al empleado dependiendo de su tipo.
- asignarAtraccionEmpleado(Empleado empleado, AtraccionMecanica atraccion): Asocia un empleado o cajero a una atracción mecánica.
- retirarLabor(Empleado empleado): Elimina la labor asignada al empleado.
- retirarLugarEmpleado(Empleado empleado): Retira cualquier asignación de lugar o atracción del empleado, si la tenía.

Cajero

La clase Cajero extiende a Empleado y representa a los empleados encargados de validar el ingreso de usuarios a las atracciones o lugares de servicio mediante la verificación de tiquetes. Puede estar asignado a una atracción o a un lugar de servicio, pero nunca a ambos simultáneamente.

Responsabilidades:

- Validar la validez de los tiquetes que presentan los usuarios antes de permitir el ingreso.
- Verificar condiciones específicas como altura, edad, exclusividad o vigencia de los tiquetes según el tipo de atracción.
- Asociarse a una atracción mecánica, atracción cultural o lugar de servicio.
- Actualizar el estado de uso del tiquete y del historial del usuario cuando corresponde.

Atributos Clave:

- atraccion: Atraccion: Representa la atracción a la que está asignado el cajero (puede ser nula).
- lugar: LugarServicio: Representa el lugar de servicio al que está asignado el cajero (puede ser nulo).

Métodos Clave:

- validarTiquete(Tiquete tiquete): Método principal que verifica si el tiquete presentado es válido para la atracción asignada. Realiza validaciones de tipo de atracción, altura, edad, uso previo del tiquete y exclusividad.
- validarTemporada(Tiquete tiquete): Verifica si el tiquete de temporada está dentro del rango de fechas válido y si cumple con la exclusividad requerida.
- validarEntrada(Tiquete tiquete): Verifica que el tiquete de tipo entrada corresponde a la atracción en cuestión y lo marca como usado.
- validarExclusividad(Tiquete tiquete): Comprueba si el nivel de exclusividad del tiquete es compatible con el de la atracción.
- Métodos de acceso (get y set) para atraccion y lugar.

Cocinero

La clase Cocinero extiende a Empleado y representa a los trabajadores encargados de preparar alimentos en las cafeterías del parque de diversiones. Cada cocinero está asociado exclusivamente a una cafetería.

Responsabilidades:

- Preparar alimentos en la cafetería a la que está asignado.
- Asociarse y trabajar exclusivamente dentro de una cafetería específica.
- Ejecutar labores de cocina según las necesidades del parque.

Atributos Clave:

- cafeteria: Cafeteria: Cafetería en la que el cocinero está trabajando. Puede ser null si no está asignado a ninguna.

Métodos Clave:

- getCafeteria(): Devuelve la cafetería en la que el cocinero está trabajando actualmente.
- setCafeteria(Cafeteria cafeteria): Asigna una cafetería al cocinero.
- prepararPlato(String plato): Simula la acción de preparar un plato. Imprime un mensaje indicando que el plato ha sido preparado.

Empleado

La clase abstracta Empleado extiende a Usuario y modela el comportamiento común de todos los empleados del parque. Define atributos y funcionalidades compartidas, como los turnos asignados, la labor desempeñada y el tipo específico de empleado.

Responsabilidades:

- Representar a los distintos tipos de empleados del parque (cajeros, cocineros, personal de servicios generales, etc.).
- Gestionar la asignación y remoción de turnos de trabajo.
- Almacenar y modificar la labor que desempeña el empleado.

Atributos Clave:

- tipoEmpleado: String: Especifica el subtipo del empleado (ej. "cajero", "cocinero").
- turnos: List<String>: Lista de turnos asignados al empleado.
- labor: String: Tarea o función principal que desempeña el empleado en su lugar de trabajo.

Métodos Clave:

- gettipo(): Devuelve el tipo específico del empleado.

- getLabor() / setLabor(String labor): Obtiene o asigna la labor del empleado.
- AsignarTurno(String turno): Asigna un nuevo turno al empleado.
- RetirarTurno(String turno): Elimina un turno específico si está asignado.
- getTurnos(): Devuelve todos los turnos asignados al empleado.

EmpleadoAtracciones

La clase EmpleadoAtracciones extiende a Empleado y representa al personal encargado de operar atracciones dentro del parque. Su funcionalidad está especialmente orientada a controlar atracciones mecánicas, respetando restricciones de seguridad asociadas al nivel de riesgo.

Responsabilidades:

- Operar una atracción asignada, si está capacitado para ella.
- Registrar y mantener un historial de atracciones para las cuales está capacitado.
- Asegurar que solo pueda operar atracciones acordes a su capacitación o de riesgo medio.

Atributos Clave:

- nivelRiesgo: String: Nivel de riesgo que puede manejar el empleado (ej. "medio", "alto").
- atraccionesCapacitadas: List<Atraccion>: Lista de atracciones mecánicas para las que ha sido capacitado.
- atraccion: Atraccion: Atracción actualmente asignada al empleado.

Métodos Clave:

- setAtraccion(Atraccion atract): Asigna una atracción al empleado si cumple con el nivel de riesgo o está capacitado. Lanza una excepción si no cumple.
- agregarAtraccion(Atraccion atraccion): Registra una atracción como capacitada para el empleado.
- sacarAtraccion(): Desvincula al empleado de su atracción actual.
- getnivel() / setNivel(String nivel): Obtiene o modifica el nivel de riesgo manejado por el empleado.
- getAtraccion(): Devuelve la atracción actualmente asignada al empleado.

EmpleadoServiciosGenerales

La clase EmpleadoServiciosgenerales extiende a Empleado y representa al personal de servicios generales que se encarga de tareas como limpieza o mantenimiento en lugares de servicio dentro del parque.

Responsabilidades:

- Realizar labores generales en lugares de servicio (como cafeterías o tiendas).
- No requiere turnos definidos, por lo que se le asigna automáticamente "N/A".
- Está asociado a un único LugarServicio en el que desempeña su labor.

Atributos Clave:

- lugar: LugarServicio: Lugar en el que el empleado de servicios generales está asignado actualmente.

Métodos Clave:

- getLugar(): Retorna el lugar asignado al empleado.
- setLugar(LugarServicio lugar): Asigna un nuevo lugar al empleado.

FastPass

La clase FastPass representa un beneficio adicional que un usuario puede adquirir para tener acceso preferencial a las atracciones del parque en un día específico.

Responsabilidades:

- Representar un pase rápido (FastPass) asociado a un usuario (Usuario) para un día concreto.
- Indicar si el pase está aún válido para ser utilizado.
- Asociar un precio al FastPass.

Atributos Clave:

- dia: String: Día en el que el FastPass es válido.
- cliente: Usuario: Usuario al que está asignado el pase.
- precio: float: Costo del pase.
- valido: boolean: Estado del pase (true si no ha sido usado, false si ya se usó).

Métodos Clave:

- getDia(), getCliente(), getPrecio(), getValido(): Accesores para los atributos principales.
- setPrecio(float price): Modifica el precio del pase.
- usar(): Marca el FastPass como usado y notifica al usuario mediante el método usarFastPass.

Tiquete

La clase Tiquete representa el concepto general de un tiquete utilizado por un usuario para

acceder a los servicios del parque. Es una clase base de la cual derivan distintos tipos de tiquetes con funcionalidades específicas.

Responsabilidades:

- Definir atributos y comportamiento comunes a todos los tipos de tiquetes (como entradas, temporada, etc.).
- Gestionar el estado de uso del tiquete.
- Asociar el tiquete a un usuario y un nivel de exclusividad.

Atributos Clave:

- exclusividad: String: Nivel de acceso del tiquete (familiar, oro, diamante, etc.).
- tipoTiquete: String: Indica el tipo de tiquete (entrada, temporada, regular, etc.).
- usado: boolean: Indica si el tiquete ya fue utilizado.
- usuario: Usuario: Referencia al usuario que posee el tiquete.
- precio: int: Costo del tiquete.

Métodos Clave:

- getExclusividad(), getUsuario(), getPrecio(), getUsado(), getTipo(): Accesores para los atributos del tiquete.
- setExcllusividad(String excl), setPrecio(int price): Modificadores de atributos.
- usarTiquete(): Marca el tiquete como usado e informa al usuario correspondiente mediante el método UsarTiquete.

Entrada

La clase Entrada representa un tiquete específico que da acceso a una sola atracción en el parque. Hereda de la clase abstracta Tiquete.

Responsabilidades:

- Representar un tiquete de entrada individual para una atracción determinada.
- Asociar directamente al usuario con la atracción que desea visitar.
- Mantener información básica sobre el acceso (precio, estado de uso, etc.).

Atributos Clave:

- atraccion: Atraccion: La atracción a la que el tiquete permite el acceso.

Los demás atributos y métodos básicos como exclusividad, tipo, uso, y usuario son heredados de la clase Tiquete.

Métodos Clave:

- getAtraccion(): Devuelve la atracción asociada a esta entrada.

TiqueteRegular

TiqueteRegular es una subclase de Tiquete que representa un tiquete de acceso general a las atracciones del parque, sujeto a restricciones de **exclusividad**.

Responsabilidades:

- Representar un tiquete estándar que permite el ingreso a múltiples atracciones según el nivel de exclusividad.
- Asociarse a un usuario y controlar su uso y validez.

Atributos Clave (heredados de Tiquete):

- exclusividad: String: Puede ser "familiar", "oro" o "diamante".
- usuario: Usuario: Persona que posee el tiquete.
- precio: int: Costo del tiquete.
- usado: boolean: Indica si el tiquete ya fue utilizado.
- tipoTiquete: String: Fijo en "regular".

TiqueteTemporada

TiqueteTemporada es una subclase de Tiquete que representa un tiquete válido durante un periodo específico de tiempo, con ciertas restricciones de exclusividad.

Responsabilidades:

- Permitir el acceso al parque o a atracciones durante un rango de fechas determinado.
- Asociarse a un usuario y controlar su uso y validez temporal.

Atributos adicionales:

- inicio: String: Fecha de inicio de validez del tiquete (formato esperado "dd-MM").
- fin: String: Fecha de fin de validez del tiquete (formato esperado "dd-MM").

Atributos heredados (desde Tiquete):

- exclusividad: String: Nivel de acceso (e.g., "familiar", "oro", "diamante").
- usuario: Usuario: Usuario asociado.
- precio: int: Costo del tiquete.
- usado: boolean: Si el tiquete ya fue utilizado.
- tipoTiquete: String: Fijo en "temporada".

Métodos Clave:

- getInicio(): Devuelve la fecha de inicio.
- getFin(): Devuelve la fecha de fin.
- Hereda métodos como usarTiquete(), getUsuario(), getPrecio(), getExclusividad(), etc.

Compra

Compra es una clase abstracta que representa una transacción realizada por un comprador, con la capacidad de agregar productos y almacenar detalles relacionados con la compra.

Responsabilidades:

- Gestionar los detalles generales de una compra.
- Permitir la modificación del precio y asociar productos al monto total de la compra.
- Mantener el tipo de compra (por ejemplo, si es una compra de tiquete, fast pass, etc.).

Atributos:

- precio: float: Monto total de la compra. Inicialmente se establece en 0, pero se puede modificar agregando productos.
- codigo: int: Código único de la compra.

- comprador: Usuario: Usuario que realizó la compra.
- tipo: String: Tipo de compra (por ejemplo, "tiquete", "fastpass", etc.).

Métodos:

- getPrecio(): Devuelve el precio total de la compra.
- getTipo(): Devuelve el tipo de la compra.
- getCodigo(): Devuelve el código único de la compra.
- getComprador(): Devuelve el comprador de la compra.
- setPrecio(float i): Establece el precio de la compra.
- agregarProducto(float i): Añade el precio de un producto a la compra, aumentando el monto total.

LugarServicio

LugarServicio representa un lugar dentro del parque (por ejemplo, una cafetería, una tienda, un área de juegos, etc.). Esta clase maneja la información sobre los empleados que trabajan en el lugar, las ventas realizadas y el historial de compras.

Responsabilidades:

- Gestionar la información de los empleados asignados a un lugar.
- Registrar las ventas realizadas y mantener un historial de compras.
- Asignar un cajero al lugar y manejar la venta de productos.

Atributos:

- empleados: List<Empleado>: Lista de empleados que trabajan en este lugar.
- nombre: String: Nombre del lugar (por ejemplo, "Cafetería Central").
- cajero: Empleado: Empleado que está asignado como cajero en este lugar.
- tipoLugar: String: Tipo de lugar (por ejemplo, "cafetería", "tienda", etc.).
- ventas: int: Total de ventas realizadas en el lugar.
- historialCompras: List<Compra>: Lista de compras realizadas en este lugar.

Métodos:

- getTipo(): Devuelve el tipo del lugar (por ejemplo, "cafetería").
- getVentas(): Devuelve el total de ventas realizadas en este lugar.
- getCajero(): Devuelve el empleado que está asignado como cajero.
- getNombre(): Devuelve el nombre del lugar.

- setVentas(): Establece el total de ventas realizadas en este lugar.
- setCajero(): Asigna un empleado como cajero del lugar.
- setNombre(): Establece el nombre del lugar.
- sacarCajero(): Elimina el cajero asignado a este lugar.
- registrarVenta(): Registra una compra realizada en el lugar y actualiza el total de ventas.
- agregarEmpleado(): Agrega un empleado al lugar de servicio.
- eliminarEmpleado(): Elimina un empleado del lugar de servicio.

Cafeteria

La clase Cafeteria extiende LugarServicio y representa una cafetería dentro del parque de diversiones. Administra los productos ofrecidos, sus precios, y al cocinero responsable.

Hereda de:

- LugarServicio

Responsabilidades:

- Gestionar los productos disponibles en la cafetería junto con sus precios.
- Asignar y obtener el cocinero a cargo.
- Proveer acceso a la lista de productos y sus respectivos precios.

Atributos:

- productos: HashMap<String, Integer>: Mapa de productos ofrecidos en la cafetería, donde la clave es el nombre del producto y el valor es su precio.
- cocinero: Cocinero: Cocinero asignado a la cafetería.

Métodos:

- agregarProducto(String nombre, int precio): Añade un producto al menú con su respectivo precio.
- quitarProducto(String nombre): Elimina un producto del menú.
- getPrecioProducto(String producto): Devuelve el precio de un producto específico.
- getProductos(): Devuelve una colección con los nombres de todos los productos disponibles.
- getCocinero(): Devuelve el cocinero asignado a la cafetería.
- setCocinero(Cocinero cocinero): Asigna un cocinero a la cafetería.

Espectaculo

La clase Espectaculo representa un evento o show programado dentro del parque de diversiones. Maneja su nombre, fecha, horario y estado de apertura.

Responsabilidades:

- Almacenar la información del espectáculo (nombre, fecha, horario).
- Controlar si el espectáculo está abierto o cerrado.
- Verificar si el espectáculo se realiza en la fecha actual.

Atributos:

- nombre: String: Nombre del espectáculo.
- fecha: Date: Fecha en la que se presentará el espectáculo.
- horario: String: Horario en que se presentará.
- abierto: boolean: Indica si el espectáculo está actualmente abierto o no.

Métodos:

- getNombre(): Retorna el nombre del espectáculo.
- getfecha(): Retorna la fecha del espectáculo.
- getHorario(): Retorna el horario del espectáculo.
- getAbierto(): Retorna si el espectáculo está abierto.
- setNombre(String name): Cambia el nombre del espectáculo.
- setFecha(Date date): Cambia la fecha del espectáculo.
- setHorario(String hor): Cambia el horario del espectáculo.
- abrir(): Marca el espectáculo como abierto.
- cerrar(): Marca el espectáculo como cerrado.
- verificarFecha(): Compara la fecha del espectáculo con la fecha actual. Si coincide, retorna true. Si no, cierra el espectáculo y retorna false.

Persistencia

La clase Persistencia proporciona métodos utilitarios para guardar y cargar objetos en archivos mediante serialización en Java.

Responsabilidades:

- Serializar y guardar cualquier objeto Java que implemente Serializable.
- Leer y deserializar objetos previamente guardados.

Métodos estáticos:

guardarObjeto(Object objeto, String rutaArchivo)

Guarda un objeto serializable en la ruta de archivo especificada.

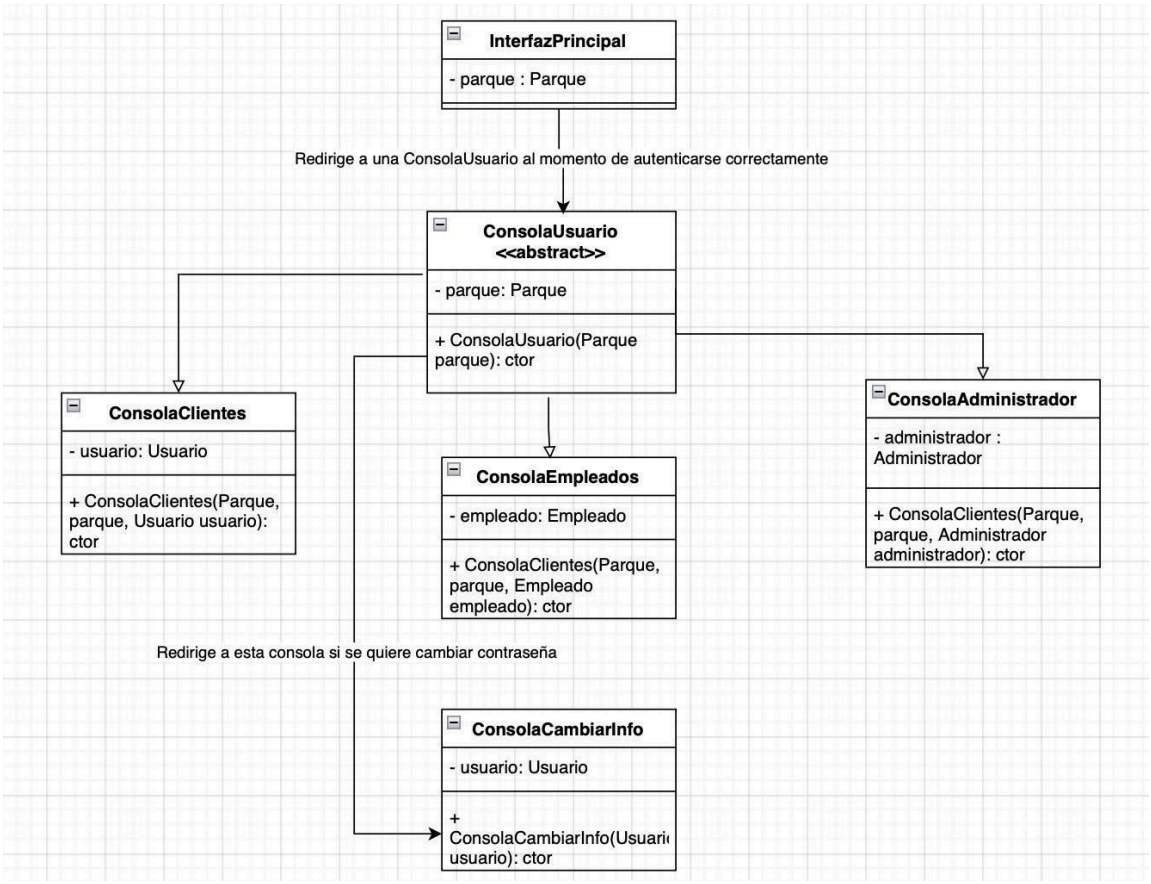
- **Parámetros:**
  - objeto: El objeto a guardar. Debe implementar Serializable.
  - rutaArchivo: Ruta del archivo donde se guardará el objeto.
- **Funcionamiento:**
  - Crea los directorios necesarios si no existen.
  - Usa ObjectOutputStream para escribir el objeto en un archivo binario.
  - Muestra un mensaje de éxito o imprime el error completo en consola si ocurre una excepción.

cargarObjeto(String rutaArchivo)

Carga y devuelve un objeto desde el archivo binario especificado.

- **Parámetros:**
  - rutaArchivo: Ruta del archivo desde donde se leerá el objeto.
- **Retorno:**
  - Devuelve el objeto leído si la operación es exitosa, o null si ocurre un error.
- **Funcionamiento:**
  - Usa ObjectInputStream para deserializar el objeto desde el archivo.
  - Captura errores de E/S o de clase no encontrada y los imprime en detalle.
-

Explicación Interfaces



La interfaz principal en la cual opera toda la aplicación tiene por nombre **InterfazPrincipal**, y esta se encarga de poder registrar a nuevos clientes que quieren entrar al parque, guardar los datos existentes y autenticar a los usuarios. Dependiendo el usuario (cliente, empleado o administrador) la Interfaz Principal redirigirá al usuario a la consola de su tipo de usuario, por ejemplo si un empleado se autentica, la interfaz lo redirigirá a la consola empleados.

Como existen tres tipos de usuario, cada consola para estos tres tipos de usuario pertenecen a la clase abstracta **ConsolaUsuario,** la cual tiene un único atributo que es el parque de diversiones, para mantener la información de atracciones, usuarios, etc en las consolas.

ConsolaClientes: En la consola clientes, pueden estar todo tipo de usuarios, donde pueden comprar tiquetes, ver los tiquetes disponibles que tienen para usar, ver el historial de sus compras realizadas previamente y además pueden ingresar al parque, Ya estando en el parque, los usuarios pueden ingresar a atracciones, a los espectáculos disponibles en el momento o comprar entradas en taquilla.

ConsolaEmpleados: Para la consola de empleados, estos pueden ver su turno, su labor, su lugar de trabajo, su información como qué tipo de empleado son y adicionalmente pueden pasar de el menu empleado al menú de cliente, para comprar sus tiquetes o etc. Al momento de comprar sus tiquetes, estos tendrán un descuento del 20% por ser empleados.

ConsolaAdministrador: Para esta consola, el administrador tendrá diferentes funcionalidades como añadir espectáculos, atracciones, empleados o lugares de servicio. Además de manipular estos y cambiarle los atributos, por ejemplo puede asignar empleados a lugares de trabajo o cerrar y abrir atracciones y espectáculos. El administrador también podrá acceder al menú de clientes, para comprar sus tiquetes e ingresar al parque, tal y como el empleado, éste tendrá un 20% de descuento por ser trabajador del parque.

Todos los tipos de Consola de usuario, podrán cambiar su contraseña actual en la consola **ConsolaCambiarInfo.**

TESTS

Administrador Test: La clase `AdministradorTest` contiene pruebas unitarias para verificar el comportamiento de la clase `Administrador`, en especial la asignación de turnos y labores a empleados del parque.

Librerías usadas:

- `org.junit.jupiter.api.*` para pruebas con JUnit 5.
- `org.junit.Assert.*` para verificaciones como `assertThrows`.

Atributos:

- `admin`: Objeto de tipo `Administrador`, inicializado antes de cada prueba.

Métodos de prueba:

`setUp()`

- Inicializa el objeto `admin` antes de cada prueba con valores de ejemplo.
- Uso de la anotación `@BeforeEach` garantiza que cada test empieza con un administrador limpio.

`testAsignarTurno_EmpleadoValido()`

- **Objetivo:** Verificar que un turno válido puede ser asignado a un cocinero sin lanzar excepciones.
- **Validaciones:**
  - Que no se lanza excepción (`assertDoesNotThrow`).
  - Que el turno se asigna correctamente (`assertEquals("mañana", ...)`).

`testAsignarTurno_EmpleadoServiciosGeneralesLanzaExcepcion()`

- **Objetivo:** Confirmar que asignar un turno a un `EmpleadoServiciosgenerales` lanza una excepción con el mensaje correcto.
- **Validaciones:**
  - Uso de `assertThrows` para capturar la excepción.
  - Comprobación del mensaje de error: `"Los empleados de servicios generales no tienen turno"`.

`testAsignarLabor_CocineroCajaEnCafeteria()`

- **Objetivo:** Validar que el administrador puede asignar al cocinero la labor de "caja" en una cafetería.
- **Validaciones:**
  - Que se asigna la labor `"caja"` al cocinero.
  - Que el cocinero queda registrado como cajero en la cafetería correspondiente.

Atraccion test: Esta clase prueba funcionalidades esenciales de la clase `AtraccionMecanica`, verificando que su creación, apertura/cierre y modificación se comporten correctamente.

Librerías utilizadas:

- `org.junit.jupiter.api.*` para pruebas con JUnit 5.
- Métodos de aserción como `assertEquals`, `assertTrue`, `assertFalse`.

Métodos de prueba:

`testCrearAtraccionMecanica()`

- **Objetivo:** Verificar que una atracción mecánica se construye correctamente con los valores dados.
- **Validaciones:**
  - Verifica todos los atributos iniciales: nombre, capacidad, ubicación, cantidad mínima de empleados, exclusividad, tipo, estado de apertura (cerrado), y número de empleados (cero).
- **Resultado esperado:** Todos los getters deben devolver los valores con los que fue construida la instancia.

`testAbrirYCerrarAtraccion()`

- **Objetivo:** Comprobar que los métodos `abrirAtraccion()` y `cerrarAtraccion()` modifican correctamente el estado de la atracción.
- **Validaciones:**
  - Inicialmente, la atracción debe estar cerrada.
  - Después de llamar a `abrirAtraccion()`, debe estar abierta.
  - Después de llamar a `cerrarAtraccion()`, debe estar nuevamente cerrada.

`testSetters()`

- **Objetivo:** Asegurar que los métodos `setNombre()` y `setCapacidad()` funcionan correctamente.
- **Validaciones:**
  - Se modifica el nombre de la atracción y se verifica con `getNombre()`.
  - Se modifica la capacidad y se verifica con `getCapacidad()`.

Entrada test: Esta clase valida el comportamiento de la clase `Entrada`, que representa un tiquete asociado a una atracción en el sistema.

Librerías utilizadas:

- `org.junit.jupiter.api.Test` — Para definir métodos de prueba con JUnit 5.
- `org.junit.jupiter.api.Assertions.*` — Para realizar aserciones como `assertEquals`.

Método de prueba:

`testCrearEntradaConAtraccion()`

- **Objetivo:** Verificar que una instancia de `Entrada` se construye correctamente con los atributos esperados.
- **Preparación:**

- Se crea un objeto `Cliente` como usuario del tiquete.

    - Se crea una `AtraccionMecanica` con valores básicos.
    - Se instancia un objeto `Entrada` con el cliente, precio y atracción.
- **Validaciones:**
    - `getAtraccion()` devuelve la atracción asignada.
    - `getTipo()` devuelve `"entrada"`, validando el tipo del tiquete.
    - `getPrecio()` devuelve el precio correcto asignado.
    - `getUsuario()` devuelve el cliente que compró la entrada.
- **Resultado esperado:** Todos los valores deben coincidir con los usados al construir la entrada.

Entrada test: Esta clase valida el comportamiento de la clase `Entrada`, que representa un tiquete asociado a una atracción en el sistema.

Librerías utilizadas:

- `org.junit.jupiter.api.Test` — Para definir métodos de prueba con JUnit 5.
- `org.junit.jupiter.api.Assertions.*` — Para realizar aserciones como `assertEquals`.

Método de prueba:

`testCrearEntradaConAtraccion()`

- **Objetivo:** Verificar que una instancia de `Entrada` se construye correctamente con los atributos esperados.
- **Preparación:**
    - Se crea un objeto `Cliente` como usuario del tiquete.
    - Se crea una `AtraccionMecanica` con valores básicos.
    - Se instancia un objeto `Entrada` con el cliente, precio y atracción.
- **Validaciones:**
    - `getAtraccion()` devuelve la atracción asignada.
    - `getTipo()` devuelve `"entrada"`, validando el tipo del tiquete.
    - `getPrecio()` devuelve el precio correcto asignado.
    - `getUsuario()` devuelve el cliente que compró la entrada.
- **Resultado esperado:** Todos los valores deben coincidir con los usados al construir la entrada.

Parque test: Esta clase contiene pruebas unitarias que validan el comportamiento básico del sistema central `Parque`, el cual gestiona usuarios, estado del parque y lógica básica de operación.

Preparación

- **Atributo `parque`:** instancia del objeto `Parque` creada antes de cada prueba en el método `setUp()` usando `@BeforeEach`.

Métodos de prueba

`testParqueInicialmenteCerrado()`

- Verifica que al crear el parque, este inicia cerrado (`getAbierto()` debe ser `false`).

`testAbrirYCerrarParque()`

- Prueba las funcionalidades `abrirParque()` y `cerrarParque()`, asegurando que alteran correctamente el estado de apertura.

`testGettersYSettersBasicos()`

- Valida métodos setters y getters para nombre, dirección y capacidad del parque.

`testGenerarCodigoCompra()`

- Confirma que el método `generarCodigoCompra()` devuelve un valor dentro del rango válido (entre 1 y 100,000,000).

`testAgregarUsuarioExitoso()`

- Crea un usuario tipo `Cliente`, lo agrega al parque y verifica que puede recuperarse por su login.

`testAgregarUsuarioRepetidoLanzaExcepcion()`

- Intenta registrar dos usuarios con el mismo login.
- Se espera una excepción con el mensaje `"Ya hay un usuario con ese login"`.

`testAutenticacionExitosa()`

- Verifica que el método `autenticarIngreso()` funciona cuando las credenciales coinciden con un usuario existente.

`testAutenticacionFallida()`

- Confirma que la autenticación falla:
  - Si la contraseña es incorrecta.
  - Si el usuario no existe.

`testUsuarioAdministrador()`

- Añade un `Administrador` al parque y verifica que se reconoce como administrador usando `usuarioAdministrador()`.

`testEliminarUsuario()`

- Agrega un usuario, lo elimina y luego verifica que ya no se encuentra registrado (`getUsuario()` devuelve `null`).

Taquilla test: Esta clase de pruebas unitarias valida el comportamiento básico de la clase `Taquilla`, que extiende `LugarServicio` y representa un punto de venta de entradas en el parque.

Métodos de prueba

`testCrearTaquilla()`

- **Propósito:** Verifica que al instanciar una taquilla, se configuren correctamente:
  - El tipo (`getTipo()`) como `"taquilla"`.
  - El nombre (`getNombre()`) como el proporcionado en el constructor (`"Entrada Principal"`).

`testModificarNombre()`

- **Propósito:** Comprueba que el método `setNombre()` funciona correctamente, permitiendo cambiar el nombre de la taquilla.

Tiquete Temporada Test: Esta clase contiene pruebas unitarias para verificar el comportamiento de la clase `TiqueteTemporada`, un tipo especializado de tiquete con vigencia definida entre dos fechas.

Métodos de prueba

`testCrearTiqueteTemporada()`

- **Propósito:** Verifica que el constructor de `TiqueteTemporada` inicializa correctamente todos los atributos:
  - Exclusividad (`"vip"`)
  - Usuario (`Cliente`)
  - Precio (`10000`)
  - Tipo (`"temporada"`, valor fijo esperado)
  - Fechas de inicio y fin (`"01-01"` y `"31-12"`)

`testGettersDeFechas()`

- **Propósito:** Asegura que los métodos `getInicio()` y `getFin()` retornan correctamente las fechas definidas para la temporada.