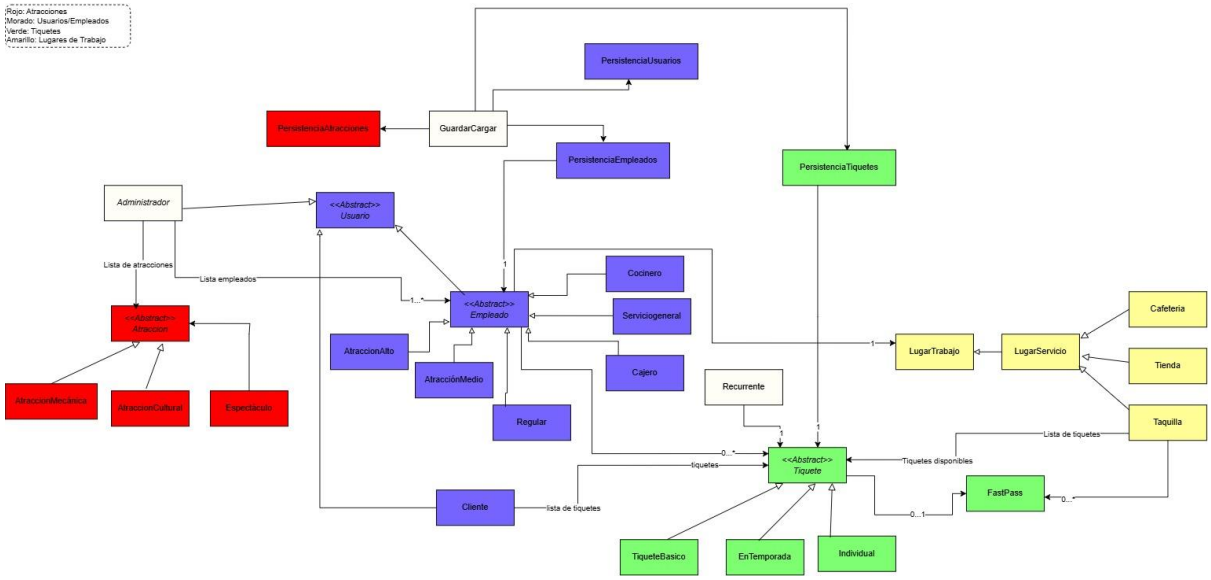
Análisis proyecto 1

Valeria Martínez 202410228, Baruc Jerónimo Triana Bastidas 202416093, Cristian Rúgeles 202411069

Este documento ofrece un análisis detallado al diseño del sistema de gestión para un parque de atracciones se incluye tres funciones principales: un catálogo de atracciones, la gestión de empleados y la venta de tiquetes. A continuación, se presenta una descripción detallada del diseño, junto con las razones detrás de las decisiones tomadas y los diagramas que ilustran la estructura y el comportamiento del sistema.

1. Diagrama de Clases de Alto Nivel (imagen con más claridad en la carpeta “docs” del repositorio de github)

El siguiente diagrama muestra las principales entidades del sistema y sus relaciones, sin incluir atributos ni métodos para facilitar la comprensión de la estructura general:



La estructura del sistema se organiza en torno a cinco paquetes principales que representan los dominios funcionales del parque de atracciones:

-modelo.atracciones: Contiene las clases relacionadas con las atracciones del parque.

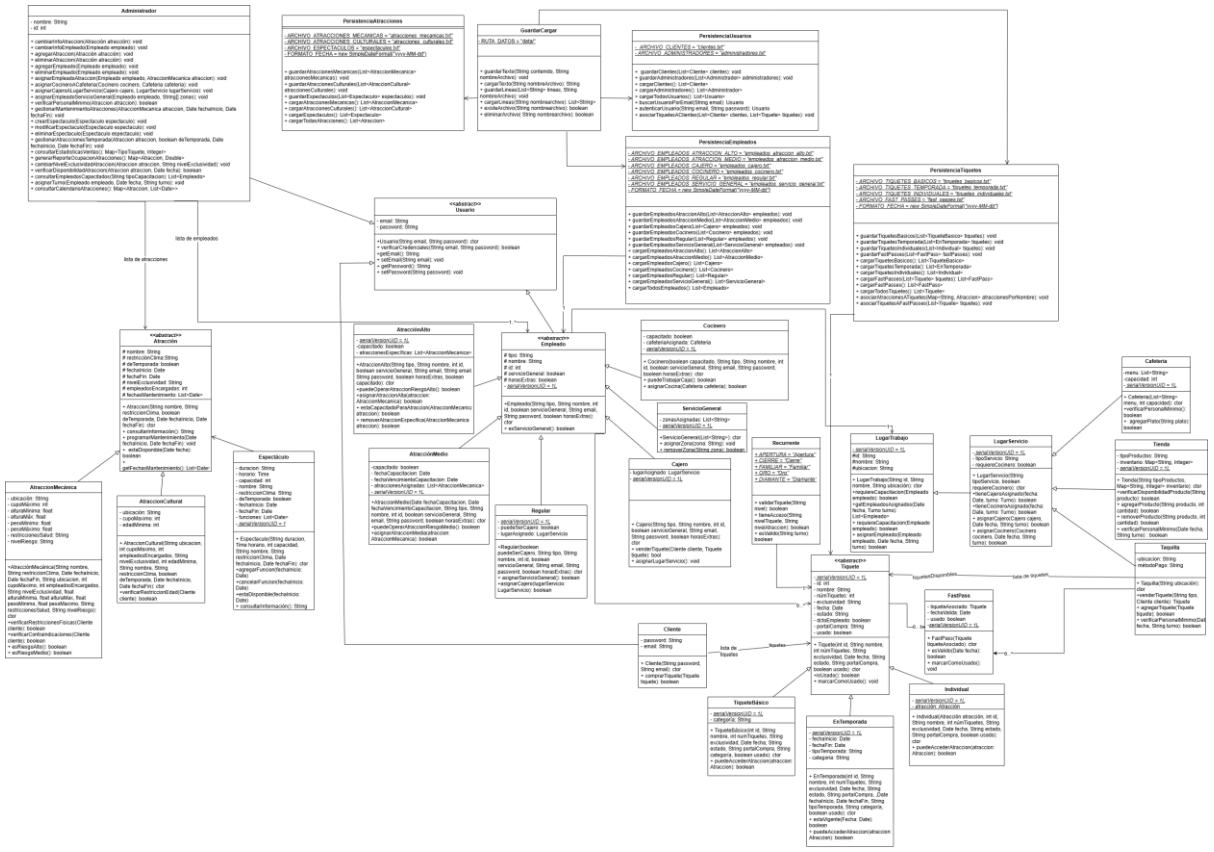
-modelo.empleados: Incluye las clases que representan a los diferentes tipos de empleados.

-modelo.lugares: Agrupa las clases relacionadas con los lugares de trabajo.

-modelo.tiquetes: Contiene las clases que modelan los diferentes tipos de tiquetes.

-modelo.usuarios: Incluye las clases que representan a los usuarios del sistema.

2. UML con atributos y métodos (imagen con más claridad en la carpeta “docs” del repositorio de github)



Clase: ConsolaParqueAtracciones

Métodos:

- main(String[] args): Método principal que inicia la aplicación. Carga los datos iniciales y presenta un menú con opciones para gestionar el parque de atracciones.
- cargarDatos(): Carga datos desde archivos en la carpeta "data". Inicializa las listas de atracciones, espectáculos, empleados, clientes y tiquetes. Muestra un resumen de los datos cargados.
- crearDatosEjemplo(): Crea datos de ejemplo para probar la aplicación. Inicializa atracciones mecánicas y culturales, espectáculos, empleados de distintos tipos, clientes y tiquetes.
- gestionarAsignacionEmpleados() throws EmpleadoException: Gestiona la asignación de empleados a atracciones mecánicas. Permite seleccionar una atracción y un empleado capacitado, y asignarle un turno en una fecha específica.
- gestionarVentaTiquetes() throws TiqueteException: Gestiona la venta de tiquetes a clientes. Permite seleccionar o crear un cliente, elegir el tipo de tiquete a vender, y verificar el acceso a atracciones.

- gestionarMantenimientoAtracciones() throws AtraccionException: Gestiona el mantenimiento de atracciones mecánicas. Permite programar fechas de mantenimiento y verificar la disponibilidad de atracciones en fechas específicas.
- guardarDatos(): Guarda todos los datos del sistema en archivos en la carpeta "data". Almacena atracciones, espectáculos, empleados, clientes y tiquetes.
- mismodia(Date fecha1, Date fecha2): Verifica si dos fechas corresponden al mismo día, ignorando la hora. Convierte las fechas a días desde una fecha de referencia y compara los resultados.

Paquete: excepciones

Clase: AtraccionException

Métodos:

- AtraccionException(String mensaje): Constructor que inicializa la excepción con un mensaje específico.
- AtraccionException(String mensaje, Throwable causa): Constructor que inicializa la excepción con un mensaje y una causa origen.

Clase: EmpleadoException

Métodos:

- EmpleadoException(String mensaje): Constructor que inicializa la excepción con un mensaje específico.
- EmpleadoException(String mensaje, Throwable causa): Constructor que inicializa la excepción con un mensaje y una causa origen.

Clase: LugarTrabajoException

Métodos:

- LugarTrabajoException(String mensaje): Constructor que inicializa la excepción con un mensaje específico.
- LugarTrabajoException(String mensaje, Throwable causa): Constructor que inicializa la excepción con un mensaje y una causa origen.

Clase: PersistenciaException

Métodos:

- PersistenciaException(String mensaje): Constructor que inicializa la excepción con un mensaje específico.
- PersistenciaException(String mensaje, Throwable causa): Constructor que inicializa la excepción con un mensaje y una causa origen.

Clase: TiqueteException

Métodos:

- TiqueteException(String mensaje): Constructor que inicializa la excepción con un mensaje específico.
- TiqueteException(String mensaje, Throwable causa): Constructor que inicializa la excepción con un mensaje y una causa origen.

Clase: UsuarioException

Métodos:

- UsuarioException(String mensaje): Constructor que inicializa la excepción con un mensaje específico.
- UsuarioException(String mensaje, Throwable causa): Constructor que inicializa la excepción con un mensaje y una causa origen.

Clase: ValidacionException

Métodos:

- ValidacionException(String mensaje): Constructor que inicializa la excepción con un mensaje específico.
- ValidacionException(String mensaje, Throwable causa): Constructor que inicializa la excepción con un mensaje y una causa origen.

Paquete: funcionesrecurrentes

Clase: Recurrente

Métodos:

- validarTiquete(String nivel): Valida que el nivel de exclusividad de un tiquete sea válido (Familiar, Oro o Diamante). Retorna true si es válido.
- tieneAcceso(String nivelTiquete, String nivelAtraccion): Verifica si un tiquete con cierto nivel de exclusividad permite acceder a una atracción con cierto nivel de exclusividad. Los tiquetes Diamante acceden a todas las atracciones, los Oro acceden a atracciones Familiar y Oro, y los Familiar solo a atracciones Familiar.
- esValido(String turno): Verifica si un turno es válido (Apertura o Cierre). Retorna true si es válido.

Paquete: modelo.atracciones

Clase: Atraccion (abstracta)

Métodos:

- Atraccion(String nombre, String restriccionClima, boolean deTemporada, Date fechaInicio, Date fechaFin, String nivelExclusividad, int empleadosEncargados): Constructor que inicializa una atracción con sus atributos básicos.

- consultarInformacion(): Método abstracto que debe ser implementado por las subclases para proporcionar información específica sobre la atracción.
- programarMantenimiento(Date fechaInicio, Date fechaFin): Programa mantenimiento para la atracción en un rango de fechas. Agrega cada día del rango a la lista de fechas de mantenimiento.
- estaDisponible(Date fecha): Verifica si la atracción está disponible en una fecha específica. Considera si la fecha está en mantenimiento y, si es de temporada, si la fecha está dentro del periodo de disponibilidad.
- mismodia(Date fecha1, Date fecha2): Verifica si dos fechas corresponden al mismo día, ignorando la hora.
- Getters y setters para todos los atributos
- equals(Object obj): Compara dos atracciones por su nombre.
- hashCode(): Genera un código hash basado en el nombre de la atracción.
- toString(): Devuelve una representación textual de la atracción.

Clase: AtraccionCultural

Métodos:

- AtraccionCultural(String nombre, String restriccionClima, boolean deTemporada, Date fechaInicio, Date fechaFin, String nivelExclusividad, int empleadosEncargados, String ubicacion, int cupoMaximo, int edadMinima): Constructor que inicializa una atracción cultural con sus atributos específicos.
- verificarRestriccionEdad(Cliente cliente): Verifica si un cliente cumple con la restricción de edad mínima para la atracción.
- consultarInformacion(): Implementación que proporciona información detallada sobre la atracción cultural, incluyendo ubicación, nivel de exclusividad, cupo máximo y restricciones de edad.
- Getters y setters para todos los atributos específicos.

Clase: AtraccionMecanica

Métodos:

- AtraccionMecanica(String nombre, String restriccionClima, boolean deTemporada, Date fechaInicio, Date fechaFin, String nivelExclusividad, int empleadosEncargados, String ubicacion, int cupoMaximo, float alturaMinima, float alturaMaxima, float pesoMinimo, float pesoMaximo, String restriccionesSalud, String nivelRiesgo): Constructor que inicializa una atracción mecánica con sus atributos específicos.
- verificarRestriccionesFisicas(Cliente cliente): Verifica si un cliente cumple con las restricciones de altura y peso para la atracción.
- verificarContraindicaciones(Cliente cliente): Verifica si un cliente tiene alguna condición de salud que contraindique el uso de la atracción.

- consultarInformacion(): Implementación que proporciona información detallada sobre la atracción mecánica, incluyendo restricciones físicas, nivel de riesgo y contraindicaciones.
- esRiesgoAlto(): Indica si la atracción es de riesgo alto.
- esRiesgoMedio(): Indica si la atracción es de riesgo medio.
- Getters y setters para todos los atributos específicos.

Clase: Espectaculo

Métodos:

- Espectaculo(String nombre, String restriccionClima, boolean deTemporada, Date fechaInicio, Date fechaFin, String duracion, String horario, int capacidad): Constructor que inicializa un espectáculo con sus atributos.
- agregarFuncion(Date fechaFuncion): Agrega una función del espectáculo en una fecha específica, verificando que no exista ya una función en esa fecha.
- cancelarFuncion(Date fechaFuncion): Cancela una función del espectáculo en una fecha específica.
- estaDisponible(Date fecha): Verifica si el espectáculo está disponible en una fecha específica, considerando si es de temporada y si hay función programada.
- mismodia(Date fecha1, Date fecha2): Verifica si dos fechas corresponden al mismo día, ignorando la hora.
- consultarInformacion(): Proporciona información detallada sobre el espectáculo, incluyendo duración, horario, capacidad y próximas funciones.
- Getters y setters para todos los atributos, con copias defensivas para las fechas y listas.
- equals(Object obj): Compara dos espectáculos por su nombre.
- hashCode(): Genera un código hash basado en el nombre del espectáculo.
- toString(): Devuelve una representación textual del espectáculo.

Paquete: modelo.empleados

Clase: AtraccionAlto

Métodos:

- AtraccionAlto(String tipo, String nombre, int id, boolean servicioGeneral, String email, String password, boolean horasExtras, boolean capacitado): Constructor que inicializa un empleado capacitado para atracciones de alto riesgo.
- puedeOperarAtraccionRiesgoAlto(): Verifica si el empleado está capacitado para operar atracciones de alto riesgo.
- asignarAtraccionAlta(AtraccionMecanica atraccion): Asigna una atracción mecánica de alto riesgo al empleado, verificando que esté capacitado y que la atracción sea de alto riesgo.

- estaCapacitadoParaAtraccion(AtraccionMecanica atraccion): Verifica si el empleado está capacitado específicamente para una atracción de alto riesgo.
- Getters y setters para el atributo capacitado y métodos para gestionar la lista de atracciones específicas.

Clase: AtraccionMedio

Métodos:

- AtraccionMedio(Date fechaCapacitacion, Date fechaVencimientoCapacitacion, String tipo, String nombre, int id, boolean servicioGeneral, String email, String password, boolean horasExtras): Constructor que inicializa un empleado capacitado para atracciones de riesgo medio.
- puedeOperarAtraccionRiesgoMedio(): Verifica si el empleado está capacitado y su certificación no ha vencido.
- asignarAtraccionMedia(AtraccionMecanica atraccion): Asigna una atracción mecánica de riesgo medio al empleado, verificando que esté capacitado y que la atracción sea de riesgo medio.
- Getters y setters para todos los atributos específicos, con copias defensivas para las fechas.

Clase: Cajero

Métodos:

- Cajero(String tipo, String nombre, int id, boolean servicioGeneral, String email, String password, boolean horasExtras): Constructor que inicializa un empleado de tipo cajero.
- venderTiquete(Cliente cliente, Tiquete tiquete) throws TiqueteException: Vende un tiquete a un cliente, verificando que el cliente y el tiquete no sean nulos.
- asignarLugarServicio(LugarServicio lugarServicio): Asigna un lugar de servicio al cajero.
- Getters y setters para el atributo lugarAsignado.

Clase: Cocinero

Métodos:

- Cocinero(boolean capacitado, String tipo, String nombre, int id, boolean servicioGeneral, String email, String password, boolean horasExtras): Constructor que inicializa un empleado de tipo cocinero.
- puedeTrabajarCaja(): Indica si el cocinero puede trabajar como cajero (siempre retorna true).

- asignarCocina(Cafeteria cafeteria): Asigna una cafetería al cocinero, verificando que esté capacitado.
- Getters y setters para los atributos capacitado y cafeteriaAsignada.

Clase: Empleado (abstracta)

Métodos:

- Empleado(String tipo, String nombre, int id, boolean servicioGeneral, String email, String password, boolean horasExtras): Constructor que inicializa un empleado con sus atributos básicos.
- esServicioGeneral(): Indica si el empleado está asignado a servicio general.
- Getters y setters para todos los atributos.
- equals(Object obj): Compara dos empleados por su ID.
- hashCode(): Genera un código hash basado en el ID del empleado.
- toString(): Devuelve una representación textual del empleado.

Clase: Regular

Métodos:

- Regular(boolean puedeSerCajero, String tipo, String nombre, int id, boolean servicioGeneral, String email, String password, boolean horasExtras): Constructor que inicializa un empleado regular.
- asignarServicioGeneral(): Asigna al empleado a servicio general, desasociándolo de cualquier lugar específico.
- asignarCajero(LugarServicio lugarServicio): Asigna al empleado como cajero en un lugar de servicio, verificando que pueda ser cajero.
- Getters y setters para los atributos puedeSerCajero y lugarAsignado.

Clase: ServicioGeneral

Métodos:

- ServicioGeneral(List<String> zonasAsignadas, String tipo, String nombre, int id, String email, String password, boolean horasExtras): Constructor que inicializa un empleado de servicio general.
- asignarZona(String zona): Asigna una zona de trabajo al empleado, verificando que no esté ya asignada.
- removerZona(String zona): Remueve una zona de trabajo del empleado.
- tieneZonaAsignada(String zona): Verifica si el empleado tiene asignada una zona específica.
- Getters y setters para el atributo zonasAsignadas, con copias defensivas.
- toString(): Devuelve una representación textual del empleado, incluyendo sus zonas asignadas.

Paquete: modelo.lugares

Clase: Cafeteria

Métodos:

- Cafeteria(String id, String nombre, String ubicacion, List<String> menu, int capacidad): Constructor que inicializa una cafetería con sus atributos específicos.
- verificarPersonalMinimo(Date fecha, String turno): Verifica si la cafetería tiene el personal mínimo necesario (cajero y cocinero) en una fecha y turno específicos.
- agregarPlato(String plato): Agrega un plato al menú de la cafetería, verificando que no exista ya.
- removerPlato(String plato): Remueve un plato del menú de la cafetería.
- Getters y setters para los atributos menu y capacidad, con copias defensivas para el menú.
- toString(): Devuelve una representación textual de la cafetería.

Clase: LugarServicio

Métodos:

- LugarServicio(String id, String nombre, String ubicacion, String tipoServicio, boolean requiereCocinero): Constructor que inicializa un lugar de servicio con sus atributos específicos.
- tieneCajeroAsignado(Date fecha, String turno): Verifica si el lugar de servicio tiene un cajero asignado en una fecha y turno específicos.
- tieneCocineroAsignado(Date fecha, String turno): Verifica si el lugar de servicio tiene un cocinero asignado en una fecha y turno específicos (si lo requiere).
- asignarCajero(Cajero cajero, Date fecha, String turno): Asigna un cajero al lugar de servicio en una fecha y turno específicos.
- asignarCocinero(Cocinero cocinero, Date fecha, String turno): Asigna un cocinero al lugar de servicio en una fecha y turno específicos, verificando que lo requiera.
- Getters y setters para los atributos tipoServicio y requiereCocinero.
- toString(): Devuelve una representación textual del lugar de servicio.

Clase: LugarTrabajo

Métodos:

- LugarTrabajo(String id, String nombre, String ubicacion): Constructor que inicializa un lugar de trabajo con sus atributos básicos.
- requiereCapacitacion(Empleado empleado): Indica si el lugar de trabajo requiere capacitación específica para un empleado (por defecto, retorna false).

- getEmpleadosAsignados(Date fecha, String turno): Obtiene la lista de empleados asignados al lugar de trabajo en una fecha y turno específicos.
- asignarEmpleado(Empleado empleado, Date fecha, String turno): Asigna un empleado al lugar de trabajo en una fecha y turno específicos, verificando que no esté ya asignado.
- Getters y setters para todos los atributos.
- equals(Object obj): Compara dos lugares de trabajo por su ID.
- hashCode(): Genera un código hash basado en el ID del lugar de trabajo.
- toString(): Devuelve una representación textual del lugar de trabajo.

Clase: Taquilla

Métodos:

- Taquilla(String id, String nombre, String ubicacion, String metodoPago): Constructor que inicializa una taquilla con sus atributos específicos.
- venderTiquete(String tipo, Cliente cliente) throws TiqueteException: Vende un tiquete de tipo especificado a un cliente, verificando disponibilidad.
- agregarTiquete(Tiquete tiquete): Agrega un tiquete a la lista de tiquetes disponibles para venta.
- verificarPersonalMinimo(Date fecha, String turno): Verifica si la taquilla tiene un cajero asignado en una fecha y turno específicos.
- Getters y setters para los atributos metodoPago y tiquetesDisponibles, con copias defensivas para la lista de tiquetes.
- toString(): Devuelve una representación textual de la taquilla.

Clase: Tienda

Métodos:

- Tienda(String id, String nombre, String ubicacion, String tipoProductos, Map<String, Integer> inventario): Constructor que inicializa una tienda con sus atributos específicos.
- verificarDisponibilidadProducto(String producto): Verifica si un producto está disponible en el inventario de la tienda.
- agregarProducto(String producto, int cantidad): Agrega una cantidad de un producto al inventario, aumentando la cantidad si ya existe.
- removerProducto(String producto, int cantidad): Remueve una cantidad de un producto del inventario, verificando disponibilidad.
- verificarPersonalMinimo(Date fecha, String turno): Verifica si la tienda tiene un cajero asignado en una fecha y turno específicos.
- Getters y setters para los atributos tipoProductos e inventario, con copias defensivas para el inventario.
- toString(): Devuelve una representación textual de la tienda.

Paquete: modelo.tiquetes

Clase: EnTemporada

Métodos:

- EnTemporada(int id, String nombre, int numTiquetes, String exclusividad, Date fecha, String estado, String portalCompra, Date fechaInicio, Date fechaFin, String tipoTemporada, String categoria, boolean usado): Constructor que inicializa un tiquete de temporada con sus atributos específicos.
- estaVigente(Date fecha): Verifica si el tiquete está vigente en una fecha específica, considerando su periodo de validez.
- puedeAccederAtraccion(Atraccion atraccion): Implementación que verifica si el tiquete permite acceder a una atracción específica, considerando su vigencia y nivel de exclusividad.
- Getters y setters para todos los atributos específicos, con copias defensivas para las fechas.
- toString(): Devuelve una representación textual del tiquete de temporada.

Clase: FastPass

Métodos:

- FastPass(Tiquete tiqueteAsociado, Date fechaValida): Constructor que inicializa un FastPass con un tiquete asociado y una fecha válida.
- esValido(Date fecha): Verifica si el FastPass es válido en una fecha específica, considerando si ha sido usado y si la fecha coincide con la fecha válida.
- marcarComoUsado(): Marca el FastPass como usado.
- mismodia(Date fecha1, Date fecha2): Verifica si dos fechas corresponden al mismo día, ignorando la hora.
- Getters y setters para todos los atributos, con copias defensivas para las fechas.
- toString(): Devuelve una representación textual del FastPass.

Clase: Individual

Métodos:

- Individual(Atraccion atraccion, int id, String nombre, int numTiquetes, String exclusividad, Date fecha, String estado, String portalCompra, boolean usado): Constructor que inicializa un tiquete individual con sus atributos específicos.
- puedeAccederAtraccion(Atraccion atraccion): Implementación que verifica si el tiquete permite acceder a una atracción específica, considerando si es la atracción asociada y si el tiquete no ha sido usado.
- Getters y setters para el atributo atraccion.
- toString(): Devuelve una representación textual del tiquete individual.

Clase: Tiquete (abstracta)

Métodos:

- Tiquete(int id, String nombre, int numTiquetes, String exclusividad, Date fecha, String estado, String portalCompra, boolean usado): Constructor que inicializa un tiquete con sus atributos básicos.
- isUsado(): Indica si el tiquete ha sido usado.
- marcarComoUsado(): Marca el tiquete como usado.
- puedeAccederAtraccion(Atraccion atraccion): Método abstracto que debe ser implementado por las subclases para verificar acceso a atracciones.
- Getters y setters para todos los atributos, con copias defensivas para las fechas.
- equals(Object obj): Compara dos tiquetes por su ID.
- hashCode(): Genera un código hash basado en el ID del tiquete.
- toString(): Devuelve una representación textual del tiquete.

Clase: TiqueteBasico

Métodos:

- TiqueteBasico(int id, String nombre, int numTiquetes, String exclusividad, Date fecha, String estado, String portalCompra, String categoria, boolean usado): Constructor que inicializa un tiquete básico con sus atributos específicos.
- puedeAccederAtraccion(Atraccion atraccion): Implementación que verifica si el tiquete permite acceder a una atracción específica, considerando su nivel de exclusividad.
- Getters y setters para el atributo categoria.
- toString(): Devuelve una representación textual del tiquete básico.

Paquete: modelo.usuarios

Clase: Administrador

Métodos:

- Administrador(String nombre, int id, String email, String password): Constructor que inicializa un administrador con sus atributos específicos.
- cambiarInfoAtraccion(Atraccion atraccion) throws AtraccionException: Cambia la información de una atracción, verificando que exista en el sistema.
- cambiarInfoEmpleado(Empleado empleado) throws EmpleadoException: Cambia la información de un empleado, verificando que exista en el sistema.
- agregarAtraccion(Atraccion atraccion) throws AtraccionException: Agrega una atracción al sistema, verificando que no exista otra con el mismo nombre.
- eliminarAtraccion(Atraccion atraccion) throws AtraccionException: Elimina una atracción del sistema, verificando que exista.
- agregarEmpleado(Empleado empleado) throws EmpleadoException: Agrega un empleado al sistema, verificando que no exista otro con el mismo ID.

- eliminarEmpleado(Empleado empleado) throws EmpleadoException: Elimina un empleado del sistema, verificando que exista.
- asignarEmpleadoAtraccion(Empleado empleado, AtraccionMecanica atraccion, Date fecha, String turno) throws EmpleadoException: Asigna un empleado a una atracción en una fecha y turno específicos, realizando múltiples validaciones.
- asignarCocineroACafeteria(Cocinero cocinero, Cafeteria cafeteria, Date fecha, String turno) throws EmpleadoException: Asigna un cocinero a una cafetería en una fecha y turno específicos, verificando su capacitación.
- asignarCajeroALugarServicio(Cajero cajero, LugarServicio lugarServicio, Date fecha, String turno) throws EmpleadoException: Asigna un cajero a un lugar de servicio en una fecha y turno específicos.
- asignarEmpleadoServicioGeneral(Empleado empleado, String[] zonas, Date fecha, String turno) throws EmpleadoException: Asigna un empleado a servicio general en zonas específicas en una fecha y turno.
- verificarPersonalMinimo(Atraccion atraccion, Date fecha, String turno): Verifica si una atracción tiene el personal mínimo necesario en una fecha y turno específicos.
- gestionarMantenimientoAtracciones(AtraccionMecanica atraccion, Date fechaInicio, Date fechaFin) throws AtraccionException: Programa mantenimiento para una atracción en un rango de fechas.
- crearEspectaculo(Espectaculo espectaculo) throws AtraccionException: Crea un nuevo espectáculo, verificando que no exista otro con el mismo nombre.
- modificarEspectaculo(Espectaculo espectaculo) throws AtraccionException: Modifica la información de un espectáculo, verificando que exista.
- eliminarEspectaculo(Espectaculo espectaculo) throws AtraccionException: Elimina un espectáculo, verificando que exista.
- gestionarAtraccionesTemporada(Atraccion atraccion, boolean deTemporada, Date fechaInicio, Date fechaFin) throws AtraccionException: Configura una atracción como de temporada, estableciendo sus fechas de disponibilidad.
- consultarEstadisticasVentas(): Retorna estadísticas de ventas por tipo de tiquete.
- generarReporteOcupacionAtracciones(): Genera un reporte de ocupación de atracciones.
- cambiarNivelExclusividadAtraccion(Atraccion atraccion, String nivelExclusividad) throws AtraccionException: Cambia el nivel de exclusividad de una atracción, verificando que el nivel sea válido.
- verificarDisponibilidadAtraccion(Atraccion atraccion, Date fecha): Verifica si una atracción está disponible en una fecha específica.
- consultarEmpleadosCapacitados(String tipoCapacitacion): Consulta empleados capacitados según el tipo de capacitación (medio o alto).
- asignarTurno(Empleado empleado, Date fecha, String turno) throws EmpleadoException: Asigna un turno a un empleado en una fecha específica.

- consultarCalendarioAtracciones(): Consulta el calendario de disponibilidad de atracciones.
- obtenerEmpleadosAsignadosTurno(Date fecha, String turno): Obtiene la lista de empleados asignados en una fecha y turno específicos.
- estaEmpleadoAsignado(Empleado empleado, Date fecha, String turno): Verifica si un empleado está asignado en una fecha y turno específicos.
- obtenerLugarAsignado(Empleado empleado, Date fecha, String turno): Obtiene el lugar asignado a un empleado en una fecha y turno específicos.
- liberarAsignacion(Empleado empleado, Date fecha, String turno) throws EmpleadoException: Libera la asignación de un empleado en una fecha y turno específicos.
- repartirTurnos(Empleado empleado) throws EmpleadoException: Reparte turnos a un empleado.
- gestionarDescuentoEmpleado(Tiquete tiquete, Empleado empleado) throws EmpleadoException: Aplica descuento de empleado a un tiquete.
- gestionarCapacitacionEmpleado(Empleado empleado, String tipoCapacitacion) throws EmpleadoException: Gestiona la capacitación de un empleado según el tipo especificado.
- generarReporteVentasPorPeriodo(Date fechaInicio, Date fechaFin): Genera un reporte de ventas por periodo de fechas.
- generarReporteAfluenciaPorAtraccion(Date fecha): Genera un reporte de afluencia por atracción en una fecha específica (método con implementación mínima).
- setEmpleados(List<Empleado> empleados): Establece la lista de empleados del administrador.
- Getters para los atributos nombre, id, atracciones, empleados y espectaculos.
- equals(Object obj): Compara dos administradores por su ID.
- hashCode(): Genera un código hash basado en el ID del administrador.
- toString(): Devuelve una representación textual del administrador.

Clase: Cliente

Métodos:

- Cliente(String nombre, int id, String email, String password, float altura, float peso, int edad): Constructor completo que inicializa un cliente con toda su información, incluidos datos físicos.
- comprarTiquete(Tiquete tiquete) throws TiqueteException: Permite al cliente comprar un tiquete, verificando que no sea nulo.
- agregarCondicionSalud(String condicion): Agrega una condición de salud al cliente, verificando que no sea nula o vacía.
- tieneCondicionSalud(String condicion): Verifica si el cliente tiene una condición de salud específica.

- Getters y setters para todos los atributos, con copias defensivas para listas y colecciones.
- toString(): Devuelve una representación textual del cliente.

Clase: Usuario (abstracta)

Métodos:

- Usuario(String email, String password): Constructor que inicializa un usuario con email y contraseña.
- verificarCredenciales(String email, String password): Verifica si las credenciales proporcionadas coinciden con las del usuario.
- Getters y setters para los atributos email y password.

Paquete: persistencia

Clase: GuardarCargar

Métodos:

- guardarTexto(String contenido, String nombreArchivo) throws IOException: Guarda contenido textual en un archivo, creando la carpeta de datos si no existe.
- cargarTexto(String nombreArchivo) throws IOException: Carga el contenido textual de un archivo, devolviendo una cadena vacía si no existe.
- guardarLineas(List<String> lineas, String nombreArchivo) throws IOException: Guarda una lista de líneas en un archivo, una por línea.
- cargarLineas(String nombreArchivo) throws IOException: Carga líneas de un archivo en una lista, devolviendo una lista vacía si no existe.
- existeArchivo(String nombreArchivo): Verifica si un archivo existe en la carpeta de datos.
- eliminarArchivo(String nombreArchivo): Elimina un archivo de la carpeta de datos.

Clase: PersistenciaAtracciones

Métodos:

- guardarAtraccionesMecanicas(List<AtraccionMecanica> atraccionesMecanicas) throws AtraccionException: Guarda una lista de atracciones mecánicas en un archivo, serializando cada una como una línea de texto.
- guardarAtraccionesCulturales(List<AtraccionCultural> atraccionesCulturales) throws AtraccionException: Guarda una lista de atracciones culturales en un archivo, serializando cada una como una línea de texto.
- guardarEspectaculos(List<Espectaculo> espectaculos) throws AtraccionException: Guarda una lista de espectáculos en un archivo, serializando cada uno como una línea de texto.

- cargarAtraccionesMecanicas() throws AtraccionException: Carga atracciones mecánicas desde un archivo, deserializando cada línea en un objeto.
- cargarAtraccionesCulturales() throws AtraccionException: Carga atracciones culturales desde un archivo, deserializando cada línea en un objeto.
- cargarEspectaculos() throws AtraccionException: Carga espectáculos desde un archivo, deserializando cada línea en un objeto.
- cargarTodasAtracciones() throws AtraccionException: Carga todas las atracciones (mecánicas y culturales) en una lista combinada.

Clase: PersistenciaEmpleados

Métodos:

- guardarEmpleadosAtraccionAlto(List<AtraccionAlto> empleados) throws EmpleadoException: Guarda una lista de empleados de atracción de alto riesgo en un archivo.
- guardarEmpleadosAtraccionMedio(List<AtraccionMedio> empleados) throws EmpleadoException: Guarda una lista de empleados de atracción de riesgo medio en un archivo.
- guardarEmpleadosCajero(List<Cajero> empleados) throws EmpleadoException: Guarda una lista de cajeros en un archivo.
- guardarEmpleadosCocinero(List<Cocinero> empleados) throws EmpleadoException: Guarda una lista de cocineros en un archivo.
- guardarEmpleadosRegular(List<Regular> empleados) throws EmpleadoException: Guarda una lista de empleados regulares en un archivo.
- guardarEmpleadosServicioGeneral(List<ServicioGeneral> empleados) throws EmpleadoException: Guarda una lista de empleados de servicio general en un archivo.
- cargarEmpleadosAtraccionAlto() throws EmpleadoException: Carga empleados de atracción de alto riesgo desde un archivo.
- cargarEmpleadosAtraccionMedio() throws EmpleadoException: Carga empleados de atracción de riesgo medio desde un archivo.
- cargarEmpleadosCajero() throws EmpleadoException: Carga cajeros desde un archivo.
- cargarEmpleadosCocinero() throws EmpleadoException: Carga cocineros desde un archivo.
- cargarEmpleadosRegular() throws EmpleadoException: Carga empleados regulares desde un archivo.
- cargarEmpleadosServicioGeneral() throws EmpleadoException: Carga empleados de servicio general desde un archivo.
- cargarTodosEmpleados() throws EmpleadoException: Carga todos los tipos de empleados en una lista combinada.

Clase: PersistenciaTiquetes

Métodos:

- guardarTiquetesBasicos(List<TiqueteBasico> tiquetes) throws TiqueteException: Guarda una lista de tiquetes básicos en un archivo.
- guardarTiquetesTemporada(List<EnTemporada> tiquetes) throws TiqueteException: Guarda una lista de tiquetes de temporada en un archivo.
- guardarTiquetesIndividuales(List<Individual> tiquetes) throws TiqueteException: Guarda una lista de tiquetes individuales en un archivo.
- guardarFastPasses(List<FastPass> fastPasses) throws TiqueteException: Guarda una lista de FastPasses en un archivo.
- cargarTiquetesBasicos() throws TiqueteException: Carga tiquetes básicos desde un archivo.
- cargarTiquetesTemporada() throws TiqueteException: Carga tiquetes de temporada desde un archivo.
- cargarTiquetesIndividuales() throws TiqueteException: Carga tiquetes individuales desde un archivo.
- cargarFastPasses(List<Tiquete> tiquetes) throws TiqueteException: Carga FastPasses desde un archivo, asociándolos con tiquetes existentes.
- cargarFastPasses() throws TiqueteException: Carga FastPasses desde un archivo sin asociarlos (para uso temporal).
- cargarTodosTiquetes() throws TiqueteException: Carga todos los tipos de tiquetes en una lista combinada.
- asociarAtraccionesATiquetes(Map<String, Atraccion> atraccionesPorNombre) throws TiqueteException: Asocia atracciones a tiquetes individuales según sus nombres.
- asociarTiquetesAFastPasses(List<Tiquete> tiquetes) throws TiqueteException: Asocia tiquetes a FastPasses según sus IDs.

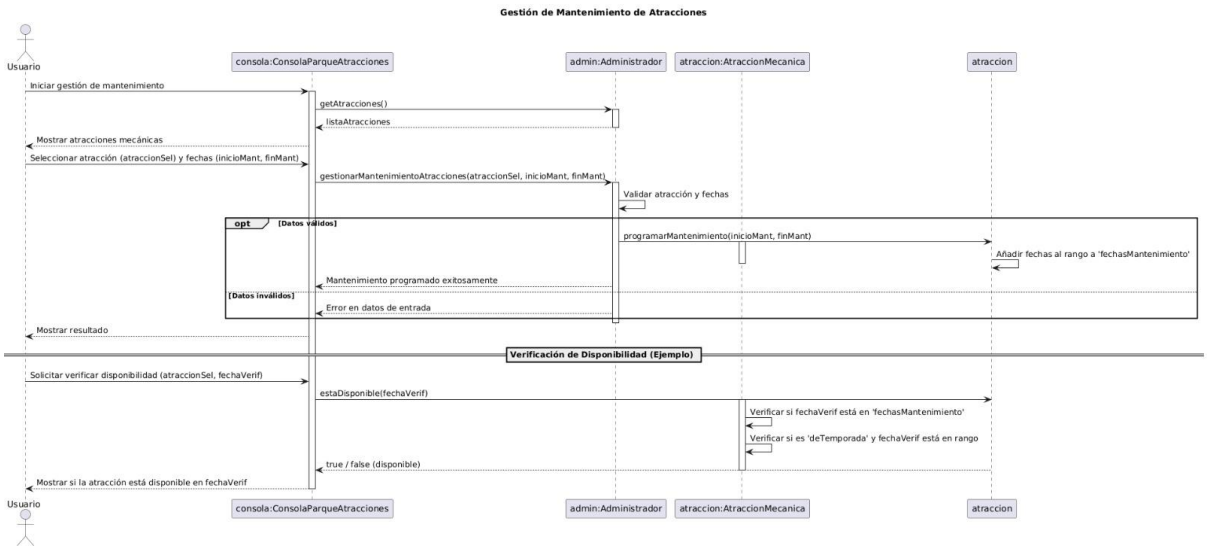
Clase: PersistenciaUsuarios

Métodos:

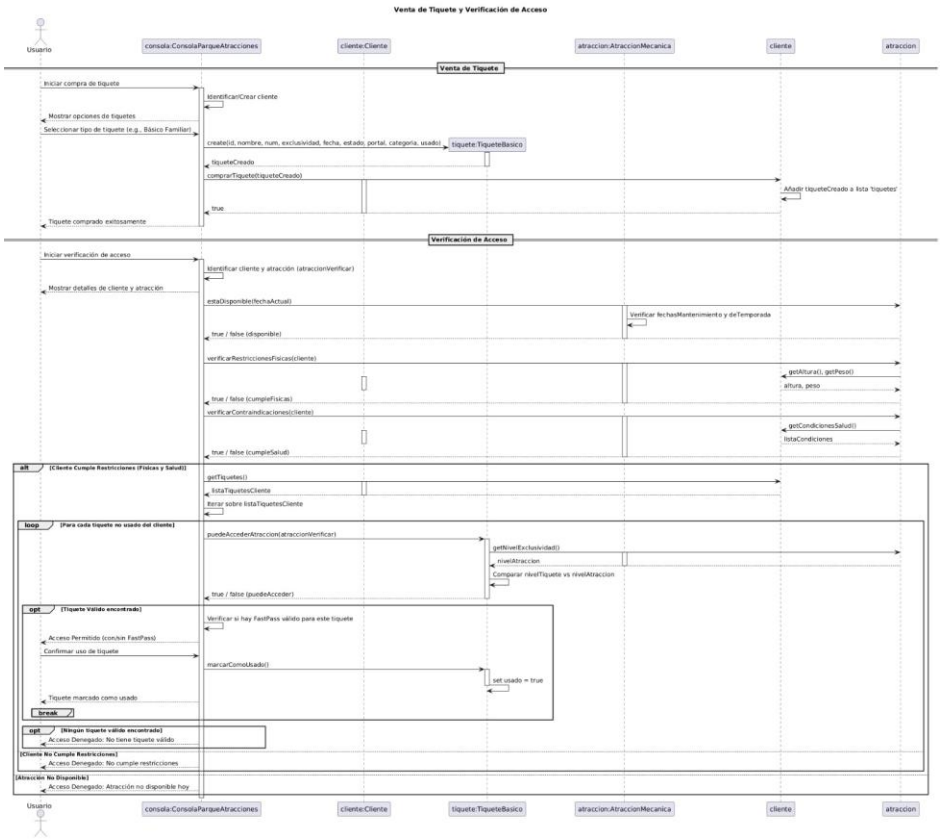
- guardarClientes(List<Cliente> clientes) throws UsuarioException: Guarda una lista de clientes en un archivo.
- guardarAdministradores(List<Administrador> administradores) throws UsuarioException: Guarda una lista de administradores en un archivo.
- cargarClientes() throws UsuarioException: Carga clientes desde un archivo.
- cargarAdministradores() throws UsuarioException: Carga administradores desde un archivo.
- cargarTodosUsuarios() throws UsuarioException: Carga todos los tipos de usuarios en una lista combinada.
- buscarUsuarioPorEmail(String email) throws UsuarioException: Busca un usuario por su email entre todos los usuarios.
- autenticarUsuario(String email, String password) throws UsuarioException: Autentica un usuario verificando sus credenciales.

- asociarTiquetesAClientes(List<Cliente> clientes, List<modelo.tiquetes.Tiquete> tiquetes) throws UsuarioException: Método preparado para asociar tiquetes a clientes.

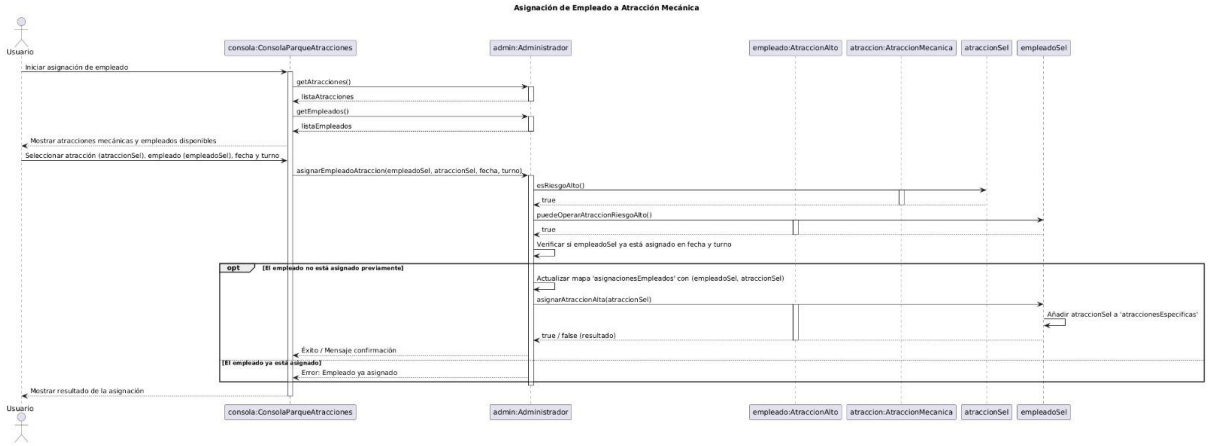
3. Diagramas de secuencia



Este diagrama muestra cómo colaboran las clases para permitir que un administrador programe mantenimientos, consulte la disponibilidad de una atracción. El usuario comienza la operación desde la consola, la consola solicita las atracciones al administrador, se muestra la lista de atracciones mecánicas al usuario, el usuario selecciona una atracción y un rango de fechas y se invoca gestionarMantenimientoAtracciones(...) desde la consola. El administrador valida si la atracción y las fechas ingresadas son válidas. Si los datos son válidos, el administrador llama a programarMantenimiento(...) en la atracción mecánica. Se añaden las fechas al atributo fechasMantenimiento. Se notifica que el mantenimiento fue programado con éxito. Si hay error en los datos, se devuelve un mensaje de error.



Este diagrama de secuencia describe el proceso completo de venta de tiquetes y verificación de acceso a una atracción. Primero, el usuario compra un tiquete desde la consola, elige un tipo (como básico o familiar), y se crea un objeto Tiquete que se agrega al cliente. Luego, para ingresar a una atracción, se verifica su disponibilidad, condiciones físicas del cliente (altura, peso, salud) y si el tiquete es válido y exclusivo para esa atracción. Si cumple todo, el acceso es permitido y el tiquete se marca como usado; si no, se rechaza el acceso.



Este diagrama muestra el proceso de asignación de un empleado a una atracción mecánica. El usuario selecciona una atracción, un empleado y un turno; el sistema valida si la atracción implica riesgo alto y si el empleado tiene la capacidad para operarla. Luego verifica que no esté previamente asignado en ese turno. Si todo

está correcto, se actualizan los registros del sistema y se asocia el empleado a la atracción; de lo contrario, se muestra un mensaje de error.

Aspectos Notables de la Implementación

- Manejo de Fechas: El sistema implementa un manejo cuidadoso de fechas, utilizando: Comparación de fechas por día, ignorando la hora (método mismodia()) y rangos de fechas para periodos de temporada y mantenimiento
- Validación Exhaustiva: Se realiza validación extensiva de parámetros en prácticamente todos los métodos, verificando:

Valores nulos

Identificadores existentes

Capacitaciones vigentes

Compatibilidad entre tipos (empleado-atracción, tiquete-atracción)

- Consistencia en la Persistencia

La persistencia mantiene un formato consistente usando separadores de campo (|) y manejo de valores nulos explícitos ("null"). Se implementa serialización/deserialización personalizada para cada tipo de entidad.

- Excepciones Específicas

El sistema utiliza excepciones personalizadas para cada dominio, permitiendo un manejo de los errores y proporcionando mensajes claros sobre los problemas detectados.

Respecto a los requerimientos funcionales

1. Gestión de Atracciones y Espectáculos
- El sistema debe permitir crear, modificar y eliminar espectáculos y programar sus funciones en fechas específicas.
- El sistema debe permitir programar mantenimientos para atracciones en rangos de fechas específicos.
- El sistema debe verificar y controlar la disponibilidad de atracciones según su mantenimiento programado.
- El sistema debe permitir configurar atracciones como "de temporada" con fechas de inicio y fin de disponibilidad.
- El sistema debe permitir asignar niveles de exclusividad (Familiar, Oro o Diamante) a cada atracción.
2. Gestión de Empleados
- El sistema debe permitir asignar empleados a atracciones según su nivel de capacitación y el nivel de riesgo de la atracción.
- El sistema debe verificar que los empleados asignados a atracciones de alto riesgo tengan capacitación específica para esa atracción.

- El sistema debe permitir asignar cajeros a lugares de servicio como taquillas, tiendas y cafeterías.
- El sistema debe permitir asignar cocineros a cafeterías verificando que estén debidamente capacitados.
- El sistema debe verificar el personal mínimo requerido para que una atracción o lugar de servicio pueda operar.

3. Gestión de Tiquetes y Acceso

- El sistema debe permitir vender tiquetes básicos con diferentes niveles de exclusividad (Familiar, Oro, Diamante).
- El sistema debe permitir vender tiquetes individuales para atracciones específicas.
- El sistema debe permitir vender FastPass asociados a tiquetes existentes para fechas específicas.
- El sistema debe verificar la validez de un tiquete al intentar acceder a una atracción.
- El sistema debe verificar que el cliente cumpla con las restricciones físicas al intentar acceder a una atracción mecánica.
- El sistema debe aplicar descuentos especiales en tiquetes para empleados

4. Gestión de Usuarios y Autenticación

- El sistema debe permitir que los usuarios inicien sesión con email y contraseña.

5. Gestión de Lugares de Servicio

- El sistema debe verificar el personal mínimo necesario para la operación de cada lugar de servicio.

6. Persistencia de Datos

- El sistema debe guardar toda la información en archivos de texto estructurados.
- El sistema debe cargar la información almacenada al iniciar la aplicación.
- El sistema debe mantener la integridad de los datos entre diferentes entidades relacionadas.

Sobre el paquete tests

En el paquete tests se incluyen diversas clases de prueba unitaria que verifican el funcionamiento correcto de los principales componentes del sistema de gestión del parque de atracciones. A continuación, se detalla la función de cada uno de estos programas:

TestAdministrador

Este programa verifica la funcionalidad de la clase Administrador, que es central para la gestión del parque. Demuestra:

-La correcta gestión de atracciones (agregar, modificar, eliminar)

-La correcta gestión de empleados (agregar, modificar, eliminar)

-La gestión de espectáculos (crear, modificar, eliminar)

-La asignación de empleados a atracciones según su capacitación

-La asignación de cocineros y cajeros a lugares de servicio

-La asignación de empleados a servicio general

-La gestión de atracciones de temporada

-La programación de mantenimiento para atracciones

-El cambio de nivel de exclusividad de atracciones

-La aplicación de descuentos de empleado a tiquetes

TestAtraccion

Este programa verifica el funcionamiento de las clases AtraccionMecanica y AtraccionCultural. Demuestra:

-La correcta verificación de restricciones físicas en atracciones mecánicas

-La correcta verificación de contraindicaciones de salud

-La correcta verificación de restricciones de edad en atracciones culturales

-La disponibilidad de atracciones según fechas de mantenimiento y temporada

-La correcta identificación del nivel de riesgo (alto o medio) en atracciones mecánicas

-La generación de información detallada sobre atracciones

TestCliente

Este programa verifica la funcionalidad de la clase Cliente. Demuestra:

-La correcta gestión de propiedades del cliente (nombre, ID, email, características físicas)

-La gestión de condiciones de salud

-La compra de tiquetes y su asociación con el cliente

-La validación de credenciales (email y contraseña)

TestEmpleado

Este programa verifica el funcionamiento de las diferentes clases de empleados. Demuestra:

-Las capacidades específicas de empleados de atracciones de alto riesgo

-Las capacidades específicas de empleados de atracciones de riesgo medio

-La venta de tiquetes por parte de cajeros

-La asignación de cocineros a cafeterías

-La asignación de empleados regulares a diferentes funciones

-La gestión de zonas asignadas a empleados de servicio general

TestEspectaculo

Este programa verifica la funcionalidad de la clase Espectaculo. Demuestra:

-La programación de funciones en fechas específicas

-La cancelación de funciones

-La verificación de disponibilidad según fechas de función y temporada

-La generación de información detallada sobre espectáculos

-El manejo de espectáculos que no son de temporada

TestLugarTrabajo

Este programa verifica el funcionamiento de las clases relacionadas con lugares de trabajo. Demuestra:

-La asignación de empleados a lugares de trabajo genéricos

-La verificación de personal mínimo en cafeterías (cajero y cocinero)

-La gestión del menú en cafeterías

-La venta de tiquetes en taquillas

-La gestión de inventario en tiendas

-La verificación de personal mínimo en diferentes lugares de servicio

TestPersistencia

Este programa verifica la correcta persistencia de datos del sistema. Demuestra:

-El almacenamiento y carga de atracciones mecánicas y culturales

-El almacenamiento y carga de espectáculos

-El almacenamiento y carga de diferentes tipos de empleados

-El almacenamiento y carga de tiquetes

-El almacenamiento y carga de clientes y administradores

-La búsqueda de usuarios por email

-La autenticación de usuarios

-La carga combinada de todas las atracciones

-La utilidad de la clase GuardarCargar para operaciones de archivo

TestTiquete

Este programa verifica el funcionamiento de las diferentes clases de tiquetes. Demuestra:

-El acceso a atracciones según nivel de exclusividad del tiquete básico

-La validación de fechas de vigencia en tiquetes de temporada

-El acceso a la atracción específica en tiquetes individuales

-La validación de FastPass según fecha y uso

-La implementación de los niveles de exclusividad (Familiar, Oro, Diamante)

-La aplicación de descuentos para empleados en tiquetes

Sobre la persistencia

El sistema implementa un mecanismo de persistencia claro y coherente:

Estructura de clases dedicadas a persistencia:

1. La clase GuardarCargar proporciona métodos básicos para manipular archivos. Clases especializadas como PersistenciaAtracciones, PersistenciaEmpleados, PersistenciaTiquetes y PersistenciaUsuarios manejan la persistencia de cada tipo de entidad.
2. Almacenamiento en archivos de texto: Cada tipo de entidad se almacena en archivos específicos (ej: "atracciones_mecanicas.txt", "empleados_cajero.txt"). Se utiliza un formato de texto delimitado por "|" que es fácil de leer y mantener

3. Directorio específico: Todos los archivos se almacenan en un directorio "data", separado del código fuente como requiere el enunciado.
4. El sistema mantiene las relaciones entre entidades a través de métodos como:

- asociarAtraccionesATiquetes(): Vincula tiquetes individuales con atracciones específicas
- asociarTiquetesAClientes(): Asigna tiquetes a los clientes correspondientes
- asociarTiquetesAFastPasses(): Conecta FastPasses con sus tiquetes asociados

5. Todos los usuarios del sistema deben tener un login y un password, la persistencia maneja esta información correctamente y proporciona métodos para autenticación en PersistenciaUsuarios.

Restricciones del Proyecto

- Las atracciones mecánicas tienen un nivel de riesgo (medio o alto) que determina qué tipo de empleado puede operarlas.
- Los empleados que operan atracciones de alto riesgo deben estar capacitados específicamente para esa atracción en particular, no siendo válido que operen otras atracciones de alto riesgo para las que no tienen capacitación.
- Cada atracción requiere un número mínimo de empleados encargados para poder funcionar. Si no se alcanza este número, la atracción no prestará servicio ese día.
- Algunas atracciones solo están disponibles en períodos específicos del año y deben estar correctamente configuradas con sus fechas de inicio y fin.
- Cada lugar de servicio (cafeterías, tiendas, taquillas) debe tener siempre al menos un cajero asignado.
- Las cafeterías requieren cocineros capacitados para el manejo y preparación de alimentos. Un cocinero puede cubrir eventualmente un puesto de cajero, pero un empleado regular no puede trabajar en la cocina.
- Los empleados del parque (y el administrador) tienen derecho a un descuento porcentual adicional cuando compran tiquetes para sí mismos.
- Algunas atracciones y espectáculos pueden tener operación restringida cuando el clima (tormentas, frío o calor extremo) no lo permita.
- Cada día hay dos turnos de trabajo (apertura y cierre), y un mismo empleado puede estar asignado a ambos turnos para hacer horas extras.
- Los tiquetes solo son válidos si están activos y dentro de la fecha.
- Cada cliente debe autenticar su identidad mediante contraseña y email para realizar compras.

Justificaciones

El diseño se pensó para separar las clases en paquetes por tema (como modelo.atracciones, modelo.empleados, persistencia, etc.). También usamos

herencia con clases base como Atraccion y Empleado; esto nos ayudó a que el código fuera más eficiente y a poder tratar diferentes tipos de objetos (como AtraccionMecanica y AtraccionCultural) de forma parecida usando polimorfismo.

Para la persistencia de datos, se eligió un mecanismo basado en archivos de texto plano con campos delimitados por un carácter específico (|), gestionado a través de clases dedicadas en el paquete persistencia. Esta decisión se priorizó por su simplicidad conceptual y de implementación. Una implicación directa de este enfoque es la necesidad de almacenar identificadores (IDs) para representar relaciones entre objetos en los archivos, lo que requiere un paso explícito de asociación en memoria después de la carga inicial de datos para reconstruir correctamente las referencias entre objetos (ej. Cliente y Tiquete).

Finalmente, se definieron excepciones personalizadas y específicas del dominio (AtraccionException, EmpleadoException, etc.) para mejorar la robustez y claridad del manejo de errores, proporcionando información más precisa sobre las fallas y permitiendo un tratamiento diferenciado de distintas condiciones excepcionales.