Universidad de los Andes

Facultad de Ingeniería de Sistemas y Computación

Proyecto 2

Diseño y Programación Orientada a Objetos

Diseño e Implementación de un Sistema de Gestión para un Parque de Atracciones

Jerónimo López

Daniel Felipe Diab G

Juan Esteban Piñeros

5 de mayo de 2025

Diseño e Implementación de un Sistema de Gestión para un Parque de Atracciones

***Resumen*—Este documento presenta la evolución del sistema de gestión del parque de atracciones, ahora con interfaces de consola para tres roles de usuario (administrador, empleados, clientes), pruebas automatizadas con JUnit, y persistencia mejorada. Se implementaron historias de usuario detalladas, validación de entradas en consola, y autenticación obligatoria. El sistema mantiene su diseño modular en Java y la persistencia en archivos CSV, pero ahora con un enfoque en la escalabilidad y verificabilidad del código.**

I. Introducción

Este documento presenta el diseño detallado e implementación del sistema de gestión para un parque de atracciones, desarrollado como parte del Proyecto 2 del curso de Diseño y Programación Orientada a Objetos (DPOO). El sistema ahora incluye:

- Interfaces de consola para administrador, empleados y clientes.
- Pruebas automatizadas con JUnit.
- Historias de usuario documentadas.
- Persistencia en archivos CSV para credenciales y datos.

II. Funcionalidades

El sistema desarrollado para la gestión del parque de atracciones implementa un conjunto amplio de funcionalidades orientadas a modelar de forma realista y operativa los procesos que ocurren dentro del parque. Estas funcionalidades se organizaron considerando los diferentes tipos de usuarios, recursos físicos y roles del sistema. A continuación, se describen las principales capacidades implementadas:

1. Gestión de Atracciones y Espectáculos

- Creación de atracciones mecánicas y culturales con restricciones específicas (edad, altura, peso, salud).
- Evaluación de disponibilidad de una atracción en función de la temporada y el clima.
- Definición de espectáculos con horarios y fechas determinadas, incluyendo los de temporada.
- Asociación de cada atracción o espectáculo a una ubicación dentro del parque.

2. Administración de Empleados y Turnos

- Creación de distintos tipos de empleados: cajeros, cocineros, operadores mecánicos y personal de servicio general.
- Asignación de turnos a empleados, con información de fecha, tipo de jornada (mañana o tarde) y lugar de trabajo.
- Validación de funciones por tipo de empleado (por ejemplo, solo los cocineros pueden preparar alimentos).
- Capacitación de operadores mecánicos para determinadas atracciones y validación de su autorización.

3. Manejo de Usuarios

- Registro de usuarios con credenciales de acceso diferenciadas por rol: administrador, empleado o cliente.
- Acceso restringido a funcionalidades según el tipo de usuario.

4. Venta y Validación de Tiquetes

- Creación y compra de diferentes tipos de tiquetes:
  - Tiquete Básico
  - Tiquete Familiar
  - Tiquete Oro
  - Tiquete Diamante
  - Tiquete de Temporada
  - Entrada Individual
- Validación del acceso a atracciones según la categoría del tiquete.
- Simulación del uso de FastPass para evitar filas en atracciones seleccionadas.
- Registro del uso de tiquetes y verificación de intentos de reutilización.

5. Gestión de Ventas

- Registro de ventas presenciales en taquillas operadas por cajeros.
- Simulación de ventas online a través de la clase `VentaOnline`, incluyendo método de pago, fecha y total.

6. Prueba unitarias

- Carpeta de pruebas en las cuales se evaluan el correcto funcionamiento de la aplicacion, esto mediante pruebas realizadas a las clases de atraccion cultural, atraccion mecanica, cliente, espectaculo, taquilla y venta Online.
- Estas pruebas evaluan el correcto funcionamiento de las clases y que la informacion quede guardada en las historias de cada una de las interfaces.

7. Interfaces de las clases.

- Interfaz Cliente: Esta interfaz cuenta con los siguientes metodos permitidos:
  - **Menú Principal:**
    - Opción 1: Comprar tiquete
    - Opción 2: Consultar tiquetes
    - Opción 3: Registrarse como nuevo cliente

- Opción 0: Salir (guarda automáticamente en `clientes.csv`)

- **Registro de Clientes:**
  - Valida nombres duplicados antes de registrar
  - Solicita: nombre, login y contraseña
  - Persiste los datos en `clientes.csv` con formato: login, password, nombre, tipotiquete1,tipotiquete2, etc.
- **Compra de Tiquetes:**
  - Busca al cliente o permite registro inmediato
  - Ofrece 4 tipos de tiquetes:
    - Básico (solo entrada)
    - Familiar (acceso a atracciones familiares)
    - Oro (acceso familiar + oro)
    - Diamante (acceso completo)
  - Opción para agregar `FastPass`
  - Actualiza automáticamente el archivo CSV
- **Consulta de Tiquetes:**
  - Muestra todos los tiquetes de un cliente
  - Información por tiquete:
    - Tipo (Básico/Familiar/Oro/Diamante)
    - Estado (Usado/No usado)

- **Flujo de Datos:**
  1. Carga inicial de clientes desde `datos/clientes.csv`
  2. Validación de entradas en cada paso
  3. Persistencia automática después de cada modificación
  4. Manejo de errores para archivos CSV
- **Detalles Técnicos:**
  - Métodos clave:
    - `leerClientesDesdeCSV()`: Carga datos con formato `login,password,nombre,tiquetes`
    - `escribirClientesEnCSV()`: Guarda en el mismo formato
    - `comprarTiquete()`: Lógica completa de compra/registro
  - Estructura de clases relacionadas:
    - `Tiquete` (clase abstracta)
    - `TiqueteBasico`, `TiqueteFamiliar`, etc. (subclases)
    - `Cliente`: Contiene lista de tiquetes

Interfaz Administrador: Esta interfaz cuenta con los siguientes metodos permitidos:

- **Gestión de Atracciones:**
  - Agregar nuevas atracciones (mecánicas o culturales) con sus respectivas restricciones.
  - Modificar o eliminar atracciones existentes.
  - Consultar el estado de disponibilidad de todas las atracciones.
- **Administración de Empleados:**
  - Registrar nuevos empleados (cajeros, operadores mecánicos, etc.).
  - Asignar turnos y lugares de trabajo.
  - Verificar certificaciones de operadores para atracciones de alto riesgo.
- **Reportes y Estadísticas:**
  - Generar reportes de ventas por tipo de tiquete.
  - Consultar asistencia diaria/semanal al parque.
  - Visualizar atracciones más populares.
- **Seguridad:**
  - Restablecer contraseñas de empleados.
  - Bloquear/desbloquear cuentas de clientes.

Flujo de trabajo:

1. Autenticación obligatoria mediante credenciales almacenadas en `auth_admin.csv`.
2. Menú principal con opciones numéricas para cada funcionalidad.
3. Validación en tiempo real de entradas (ej: formatos de email, rangos numéricos).
4. Persistencia automática de cambios en archivos CSV.

Interfaz Empleado: Interfaz de consola para empleados del parque con autenticación y funciones específicas por rol:

- **Autenticación Segura:**
  - Valida credenciales contra `empleados.csv`
  - Verifica rol (Cajero, Cocinero, OperadorMecánico, ServicioGeneral)
  - Bloquea acceso a administradores
- **Menú Principal:**
  - 1. Validar ingreso a lugar de trabajo
  - 2. Marcar asistencia de cliente
  - 3. Ver historial de cliente
  - 4. Consultar turno empleado
  - 0. Salir
- **Validación de Ingreso:**
  - Verifica usuario/contraseña en `empleados.csv`
  - Retorna objeto `Empleado` del tipo correcto con:
    - Datos personales
    - Lugar de trabajo
    - Turno asignado (formato: `Diurno (YYYY-MM-DDTHH:MM - YYYY-MM-DDTHH:MM)`)
- **Registro de Asistencias:**
  - Autentica cliente contra `clientes.csv`
  - Registra en `asistencias_clientes.csv` con formato:
    `nombre_cliente,fecha_visita,hora_entrada`
  - Evita registros duplicados por día
- **Consulta de Historial:**
  - Muestra todas las visitas de un cliente
  - Formato de salida:
    `Fecha: YYYY-MM-DD | Hora: HH:MM`

- **Gestión de Turnos:**
  - Busca empleado por nombre
  - Muestra turnos para fecha específica
  - Procesa formato complejo de turnos: `tipo_turno (fecha_inicio - fecha_fin)`

Flujo de Trabajo:

1. Autenticación con usuario/contraseña
2. Carga de datos desde CSV
3. Menú interactivo con validación de entradas
4. Persistencia automática de cambios

Detalles Técnicos:

- Clases involucradas:
  - `Empleado` (clase abstracta)
  - Subclases: `Cajero`, `Cocinero`, etc.
  - `Turno`: Maneja lógica de horarios
- Archivos CSV:
  - empleados.csv
  - Formato: rol, login,password,nombre,id,lugar,turno

III. Diseño del Sistema

El diseño del sistema se fundamenta en los principios de programación orientada a objetos, dividiendo las responsabilidades en distintos paquetes según el dominio funcional: usuarios y empleados, atracciones y espectáculos, y tiquetes. A continuación, se presentan los diagramas de clases, descripciones de las entidades principales y las decisiones clave de diseño adoptadas durante el desarrollo.

III-A. Diagrama de Clases de Diseño

El sistema está compuesto por múltiples clases distribuidas en tres paquetes principales:

- **Paquete Persona:** Contiene todas las clases relacionadas con los usuarios del sistema, como clientes, empleados y el administrador.
- **Paquete Atracciones:** Agrupa las clases que modelan atracciones mecánicas, culturales, espectáculos, ubicación y temporadas.
- **Paquete Tiquetes:** Incluye las clases relacionadas con la compra, gestión y validación de los diferentes tipos de tiquetes.
- **Paquete Tiquetes:** Incluye las clases relacionadas con la compra, gestión y validación de los diferentes tipos de tiquetes.
- **Paquete Pruebas:** Incluye las clases relacionadas con las pruebas de las clases de los otros paquetes, estas fueron desarrolladas con JUnit.
- **Paquete Interfaz:** Incluye las clases Main e interfaces de las clases cliente, Administrador y empleado.
- **Paquete Persistencia:** Incluye unicamente a la clase ArchivoPlano, la cual tiene el proposito de proveer métodos para manipular archivos de texto, incluyendo: Escritura, Lectura y formateo especifico

El diagrama de clases completo incluye relaciones de herencia, composición y asociación entre las clases. A continuación, se presenta una descripción de las clases más importantes.

Clases principales del paquete `persona`:

- **Usuario (abstracta):** Clase base que define las credenciales comunes para todos los tipos de usuarios.
- **Administrador:** Hereda de `Usuario`. Permite asignar turnos, modificar empleados y gestionar la operación general del parque.
- **Empleado (abstracta):** Define las propiedades y métodos comunes a todos los empleados del parque.
- **Turno:** Representa un turno de trabajo, incluyendo la fecha, tipo de turno (mañana/tarde), y lugar de trabajo.
- **Subclases de Empleado:** `Cajero`, `Cocinero`, `ServicioGeneral` y `OperadorMecanico`, cada una con capacidades específicas según su rol.

Clases principales del paquete `atracciones`:

- **Atraccion (abstracta):** Clase base para modelar las atracciones del parque, con atributos como cupo, empleados requeridos, y nivel de exclusividad.
- **AtraccionCultural:** Subclase que agrega restricción por edad mínima.
- **AtraccionMecanica:** Subclase que incorpora restricciones por salud, altura, peso y riesgo.
- **Espectaculo:** Clase que representa eventos del parque, definidos por nombre, fechas y horarios.
- **Ubicacion:** Indica en qué zona del parque se encuentra una atracción o lugar.
- **Temporada:** Define un rango de fechas para disponibilidad especial de atracciones o espectáculos.

Clases principales del paquete `tiquetes`:

- **Cliente:** Usuario que adquiere y utiliza tiquetes para acceder a las atracciones.
- **Tiquete (abstracta):** Clase base con atributos como uso, fastpass y tipo de acceso.
- **Subtipos de Tiquete:** `TiqueteBasico`, `Familiar`, `Oro`, `Diamante`, `Temporada` y `EntradaIndividual`.
- **VentaOnline:** Gestiona la compra en línea de tiquetes, procesamiento de pagos y generación de facturas.
- **Taquilla:** Punto físico de venta, asociado a cajeros.

Clase principal del paquete `persistencia`:

- **Archivo plano:** Clase base que se encarga de manejar la lectura y escritura de datos en archivos planos (CSV), garantizando que la información del sistema (empleados, clientes, tiquetes, etc.) persista entre ejecuciones.

Clases principales del paquete `Interfaz`:

- **Interfaz cliente:** Clase que ejecuta la interfaz basada en consola de los siguientes metodos
  - Comprar un tiquete
  - Consultar los tiquetes que tiene un cliente
  - Registrar un nuevo cliente
  - Salir de la aplicacion
- **Interfaz Empleado:** Clase que ejecuta la interfaz basada en consola de los siguientes metodos que tiene un Empleado:
  - Validar ingreso a lugar de trabajo

- Marcar asistencia del cliente
- Ver historial del cliente
- Consultar turno Empleado
- Salir de la aplicacion

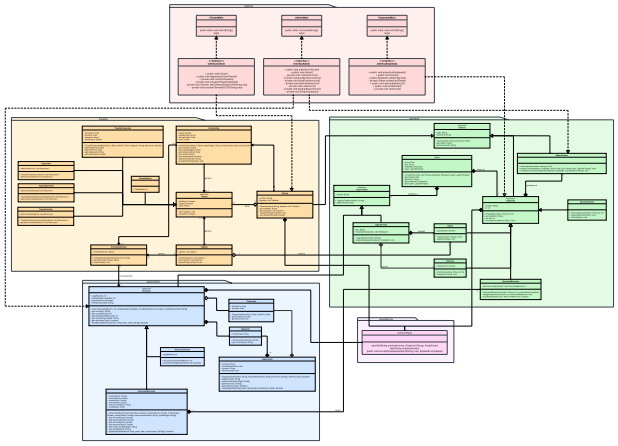
- **Interfaz Administrador:** Clase que ejecuta la interfaz basada en consola de los siguientes metodos que tiene un Administrador:
  - Crear atraccion
  - Modificar atraccion
  - Eliminar atracción
  - Crear nuevo turno
  - Asignar turno a empleado
  - Crear un nuevo empleado
  - Salir de la aplicacion

Clases principales del paquete `Pruebas`*:*

- Esta clase se encarga de contener las pruebas unitarias de las principales clases del programa
- **Test atraccion Cultural**
- **Test atraccion Mecanica**
- **Test atraccion Cliente**
- **Test Espectaculo**
- **Test Taquilla**
- **Test Venta Online**
- **Test Archivo Plano**

III-B. Diagrama de Clases de Alto Nivel

El siguiente diagrama resume las principales relaciones entre los paquetes y las entidades clave del sistema, omitiendo detalles internos (atributos y métodos) para facilitar la comprensión estructural general.



El diagrama completo con todos los métodos y atributos se encuentra en el anexo de este documento. Esta es una imagen de referencia para que pueda hacerse una idea de como es el UML de manera general.

III-C. Diagrama de secuencia

En este documento para facilitar la lectura de los diagramas de secuencias hemos hecho 3 diagramas principales que explican el funcionamiento

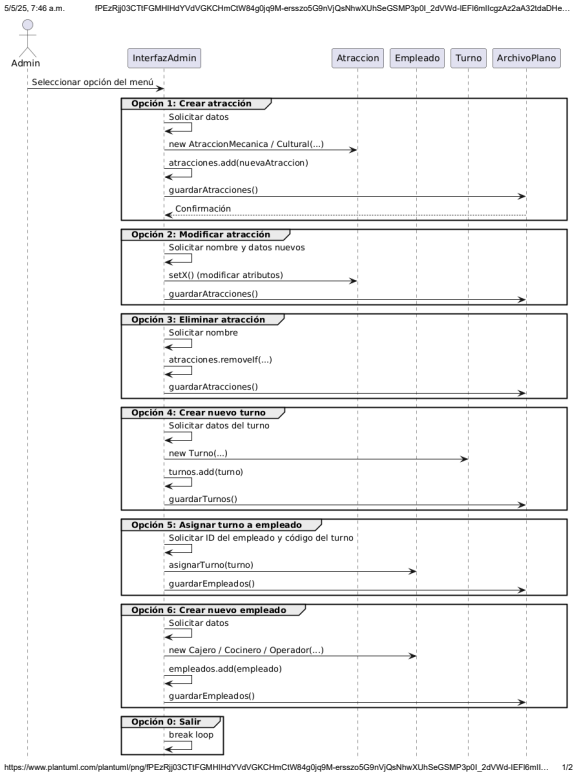


Diagrama de secuencia que representa la interfaz de un administrador con todas las acciones que este puede realizar

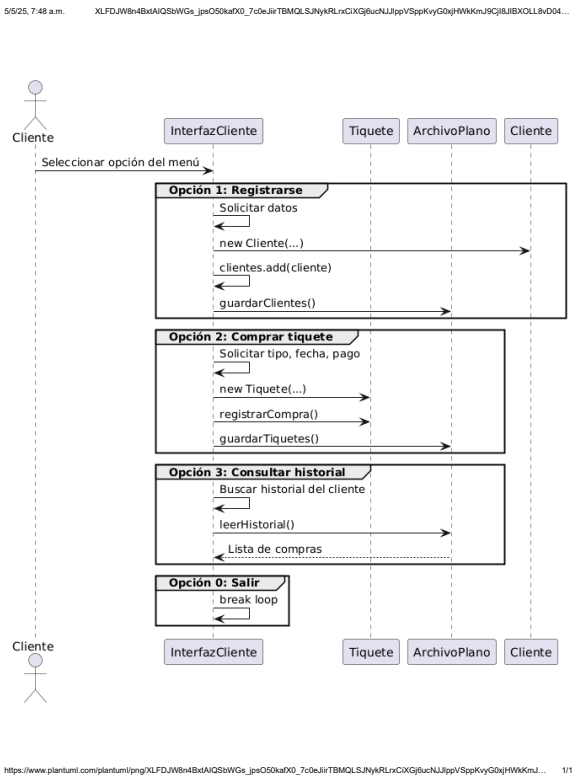


Diagrama de secuencia que representa la interfaz de un

cliente con todas las acciones que este puede realizar

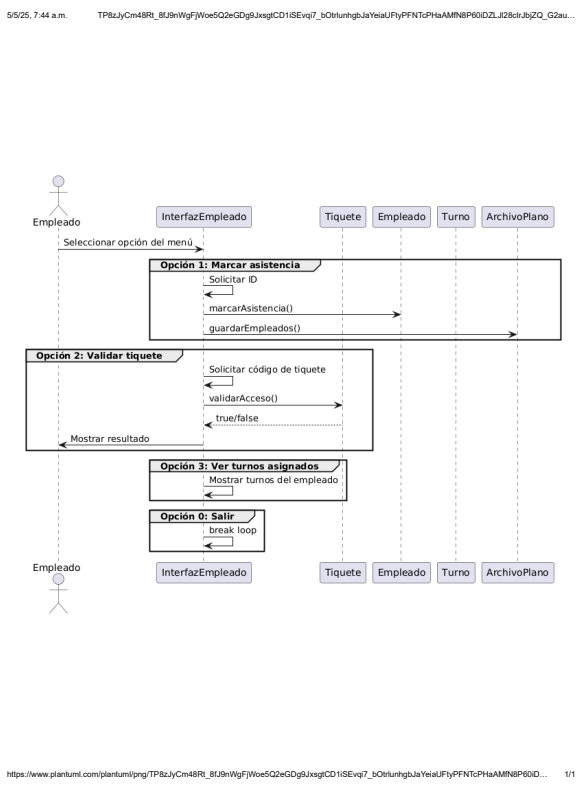


Diagrama de secuencia que representa la interfaz de un empleado con todas las acciones que este puede realizar

III-D. Justificación del Diseño

La organización del sistema en paquetes separados facilita la modularidad, el mantenimiento y la extensibilidad. Se utilizaron clases abstractas para generalizar comportamientos comunes (como `Usuario`, `Empleado` y `Tiquete`) y se aplicaron principios SOLID para la separación de responsabilidades y la reutilización de código. Además:

- Las reglas de negocio específicas (como validaciones de acceso, restricciones de atracciones o asignación de turnos) se implementan en las clases correspondientes, evitando lógica duplicada.
- La herencia permite manejar tipos de atracciones y tiquetes de manera polimórfica.
- La persistencia de datos se maneja desde clases centrales que escriben y leen archivos de texto de manera estructurada.

III-E. Paquete `atracciones`

El paquete `atracciones` contiene las clases responsables de modelar las atracciones y espectáculos disponibles en el parque. Estas clases encapsulan las propiedades físicas y lógicas de las atracciones, así como sus restricciones de acceso, disponibilidad y ubicación.

Clase `Atraccion` (abstracta): Esta clase es la base para todos los tipos de atracciones en el parque. Define atributos comunes como:

- `cupoMaximo`: número máximo de visitantes permitidos.
- `empleadosRequeridos`: cantidad mínima de empleados necesarios para operar.
- `exclusividad`: nivel de acceso requerido (Familiar, Oro o Diamante).
- `climaRestringido`: indica si la atracción puede cerrarse por condiciones climáticas.
- `temporada`: objeto de tipo `Temporada` que define su periodo de disponibilidad.

Sus métodos permiten:

- Verificar disponibilidad por clima o temporada.
- Obtener el nivel de exclusividad.
- Consultar si requiere empleados asignados.

Clase `AtraccionCultural`: Subclase de `Atraccion` que representa atracciones tipo museo, teatro o casas del terror. Agrega:

- `edadMinima`: edad mínima para ingresar.

Incluye un método que evalúa si un cliente cumple la restricción de edad.

Clase `AtraccionMecanica`: Subclase de `Atraccion` que modela atracciones físicas como montañas rusas o carruseles. Introduce:

- `alturaMinima`, `alturaMaxima`: rango de altura permitida.
- `pesoMinimo`, `pesoMaximo`: rango de peso permitido.
- `restriccionesSalud`: lista de condiciones médicas no aptas (ej. vértigo, cardiopatías).
- `nivelRiesgo`: indica si el riesgo es medio o alto.

Incluye métodos para verificar si un cliente puede acceder a la atracción en función de sus condiciones físicas y de salud.

Clase `Espectaculo`: Clase independiente de `Atraccion`. Modela eventos especiales que pueden ocurrir en cualquier parte del parque. Contiene:

- `nombre`: identificador del espectáculo.
- `fechasDisponibles`, `horarios`: programación del evento.
- `esDeTemporada`: indica si el evento solo ocurre durante un periodo específico.

Métodos clave permiten validar disponibilidad para un día específico.

Clase `Ubicacion`: Clase auxiliar que representa una zona física dentro del parque. Define:

- `zona`: nombre o código de la zona.

Permite asociar atracciones con su localización dentro del parque.

Clase `Temporada`: Clase que modela un rango de tiempo en el que una atracción o espectáculo está disponible. Atributos:

- `fechaInicio`, `fechaFin`: delimitan la temporada.

Sus métodos permiten consultar si una fecha dada cae dentro del rango válido.

Resumen: El paquete `atracciones` centraliza la lógica de validación de acceso y disponibilidad de las atracciones del parque, estableciendo una clara diferenciación entre atracciones culturales, mecánicas y espectáculos, permitiendo su extensión y validación mediante principios de herencia y polimorfismo.

III-F. Paquete `tiquetes`

El paquete `tiquetes` contiene las clases que modelan el proceso de compra, gestión y validación de los diferentes tipos de tiquetes disponibles en el parque. Estas clases permiten controlar el acceso a las atracciones, validar condiciones especiales de uso, y manejar las transacciones tanto físicas como virtuales.

Clase `Tiquete` (abstracta): Clase base para todos los tipos de tiquetes del parque. Contiene atributos generales como:

- `utilizado`: indica si el tiquete ya fue usado.
- `tieneFastPass`: determina si el tiquete incluye acceso preferencial.
- `tipo`: categoría del tiquete (básico, oro, etc.).

Incluye métodos para:

- Consultar y modificar el estado del tiquete.
- Validar si permite el acceso a determinada atracción.

Clases derivadas de `Tiquete`:

- **`TiqueteBasico`**: permite el ingreso al parque, pero no a ninguna atracción.
- **`TiqueteFamiliar`**: permite acceso a atracciones de nivel `Familiar`.
- **`TiqueteOro`**: permite acceso a atracciones `Familiar` y `Oro`.
- **`TiqueteDiamante`**: acceso total a todas las atracciones del parque.
- **`TiqueteTemporada`**: acceso ilimitado por un rango de fechas definido (`fechaInicio` y `fechaFin`), con una categoría de acceso incluida y descuento aplicado.
- **`EntradaIndividual`**: da acceso una sola vez a una atracción específica, sin importar la categoría general del cliente.

Clase `Cliente`: Usuario del parque que puede adquirir uno o varios tiquetes. Contiene:

- `nombre`: identificador del cliente.
- `tiquetes`: lista de tiquetes adquiridos.

Métodos principales:

- Comprar un tiquete (según tipo).
- Consultar los tiquetes disponibles.
- Usar un tiquete y validar su acceso.

Clase `VentaOnline`: Modela una transacción de compra de tiquetes a través de internet. Atributos clave:

- `cliente`: cliente que realiza la compra.
- `metodoPago`: forma de pago utilizada.
- `fechaCompra`: momento de la compra.
- `total`: valor de la transacción.

Incluye métodos para procesar el pago, generar facturas y enviar confirmaciones.

Clase `Taquilla`: Representa una taquilla física en el parque. Contiene:

- `cajeros`: lista de empleados asignados a la taquilla.

Sus métodos permiten registrar ventas presenciales de tiquetes y asignar cajeros para la atención al público.

Resumen: El paquete `tiquetes` encapsula todas las operaciones relacionadas con la adquisición y validación de tiquetes. Gracias al uso de herencia y especialización, se manejan fácilmente múltiples tipos de entrada, cada una con reglas específicas de acceso, vigencia y beneficios. También se contempla tanto la venta física como digital, lo cual permite al sistema adaptarse a diferentes canales de atención al cliente.

III-G. Paquete `persona`

El paquete `persona` contiene las clases relacionadas con los distintos tipos de usuarios del sistema: clientes, empleados y administradores. Este paquete también incluye las clases necesarias para modelar los turnos y lugares de trabajo dentro del parque. Se hace uso de herencia y composición para garantizar la reutilización de atributos y comportamientos comunes.

Clase `Usuario` (abstracta): Clase base para todos los tipos de usuario del sistema. Define atributos comunes como:

- `login`: nombre de usuario.
- `password`: contraseña de acceso.

Provee métodos para acceder y modificar las credenciales, y sirve como superclase para `Cliente`, `Empleado` y `Administrador`.

Clase `Administrador`: Usuario con permisos especiales para gestionar todo el sistema. Hereda de `Usuario`. Sus principales responsabilidades incluyen:

- Asignar turnos a empleados.
- Modificar la información de atracciones, empleados y espectáculos.
- Consultar información sensible del sistema.

Clase `Empleado` (abstracta): Clase base para los diferentes tipos de trabajadores del parque. Define atributos como:

- `nombre`
- `idEmpleado`
- `turnos`: lista de turnos asignados.

Sus métodos permiten consultar turnos por fecha y verificar disponibilidad. Las subclases de `Empleado` representan roles específicos con restricciones únicas.

Subclases de `Empleado`:

- **`Cajero`**: Encargado de las ventas en taquillas. Tiene métodos para registrar ventas y validar transacciones.
- **`Cocinero`**: Encargado de la preparación de alimentos. Posee un atributo para verificar si está capacitado en cocina.
- **`ServicioGeneral`**: Se encarga de labores de limpieza y mantenimiento en el parque.
- **`OperadorMecanico`**: Operador de atracciones mecánicas. Tiene una lista de atracciones para las que está certificado, y métodos para verificar su autorización en una atracción específica.

Clase `Turno`: Representa un turno laboral. Contiene:

- `fechaTurno`
- `tipoTurno`: apertura o cierre.
- `lugarTrabajo`: instancia de un lugar donde se asigna el empleado.

Permite consultar la información asociada al turno y verificar conflictos de horario.

Clase `LugarTrabajo` (abstracta): Clase abstracta que representa un lugar físico dentro del parque donde puede asignarse personal. Define:

- `nombreLugar`

Clase `LugarServicio`: Subclase de `LugarTrabajo` que representa un lugar de atención al público, como una taquilla, tienda o cafetería. Permite gestionar la asignación de empleados por turno.

Resumen: El paquete `persona` encapsula todos los elementos relacionados con los usuarios del sistema y el manejo del recurso humano. Su diseño facilita la gestión de múltiples tipos de empleados con roles claramente definidos, respetando las restricciones operativas del parque mediante validaciones específicas por tipo de cargo.

IV. Persistencia de Datos

El sistema cuenta con un paquete de persistencia de datos plenamente funcional que utiliza archivos planos en formato CSV. Esta implementación asegura que información clave del sistema como los empleados, turnos, clientes y tiquetes se conserve entre sesiones al almacenar su estado en archivos externos y restaurarlo posteriormente, asegurando la continuidad operativa mejorando el funcionamiento y aumentando notablemente la experiencia del usuario.

Implementación de la persistencia

Para gestionar esta funcionalidad, se desarrolló la clase ArchivoPlano ubicada en el paquete persistencia, esta está encargada de agrupar los métodos necesarios para leer y escribir archivos de texto. Esta clase ofrece dos funciones principales:

- leer(String nombreArchivo): lee línea por línea el contenido de un archivo CSV y lo devuelve como una lista de cadenas.
- escribir(String nombreArchivo, ArrayList<String> lineasTexto): permite guardar los cambios realizados durante la ejecución, sobrescribiendo el archivo con la información más actual.

Código en Java usado para leer y escribir datos:

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;

public class ArchivoPlano {

        public void escribir(String nombreArchivo,
        ↪  ArrayList<String> lineasTexto) {
                try {
                        BufferedWriter bw = new
                        ↪  BufferedWriter(new FileW
                        ↪  riter(nombreArchivo));
                        for(String linea :
                        ↪  lineasTexto) {
                                bw.write(linea);
                                bw.newLine();
                        }
                        bw.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        public ArrayList<String> leer (String
        ↪  nombreArchivo){
                ArrayList<String> lineasTexto = new
                ↪  ArrayList<String>();
                try {
                        BufferedReader br = new
                        ↪  BufferedReader(new FileR
                        ↪  eader(nombreArchivo));
                        String linea;
                        while((linea =
                        ↪  br.readLine()) != null) {
                                lineasTexto.add(line
                                ↪  a);
                        }
                        br.close();
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return lineasTexto;
        }

        public void escribirEmpleadoAppend(String
        ↪  ruta, Empleado empleado) {
            try (FileWriter fw = new
            ↪  FileWriter(ruta, true);
                 BufferedWriter bw = new
                 ↪  BufferedWriter(fw);
                 PrintWriter out = new
                 ↪  PrintWriter(bw)) {

                StringBuilder sb = new
                ↪  StringBuilder();
                sb.append(empleado.getClass().getSim
                ↪  pleName()).append(",");
                sb.append(empleado.getLogin()).appen
                ↪  d(",");
                sb.append(empleado.getPassword()).ap
                ↪  pend(",");
                sb.append(empleado.getNombre()).appe
                ↪  nd(",");
                sb.append(empleado.getId()).append("
                ↪  ,");
                sb.append(empleado.getLugarTrabajo()
                ↪  ).append(",");
                sb.append(empleado.getTurno());
            out.println(sb.toString());
        }
}
```

Carga automática de datos al iniciar el sistema

Cada vez que el sistema arranca, carga automáticamente los archivos ubicados en la carpeta datos/. Estos archivos están organizados por tipo de entidad. Por ejemplo:

- empleados.csv: contiene datos como el tipo de empleado, credenciales de acceso, nombre, ID, departamento y el turno asignado.
- clientes.csv: almacena las credenciales de los clientes y los tiquetes que poseen, separados por punto y coma.

El sistema transforma esta información en objetos específicos, como instancias de Administrador, Cajero, Cliente, TiqueteOro, entre otros. Para lograr esto, se emplean estructuras de control como switch, que permiten identificar el tipo de entidad a crear, junto con métodos de apoyo que asignan turnos, atracciones o listas de tiquetes según corresponda.

Los archivos estarán organizados de forma modular:

- `clientes.csv`: con los datos de cada cliente.
- `empleados.csv`: lista de empleados y sus turnos.

Justificación del enfoque

Se eligió trabajar con archivos planos por su simplicidad, portabilidad y bajo costo de implementación, cualidades que se ajustan bien a la etapa actual del proyecto. Esta estrategia permite mantener un diseño modular, ya que cada tipo de entidad cuenta con su propio archivo independiente, lo cual facilita tanto la edición como el mantenimiento de los datos.

El formato CSV resulta especialmente práctico: es claro, fácil de leer, se puede modificar con cualquier editor de texto y, al mismo tiempo, mantiene una estructura ordenada y predecible.

Además, este enfoque no limita el crecimiento del sistema. En el futuro, si el volumen de datos o la cantidad de usuarios crece, será posible migrar a una base de datos más robusta sin necesidad de rehacer todo el sistema debido al diseño modular que separa los datos de la lógica del programa.

Resumen: Con esta implementación, el sistema ahora puede guardar y cargar datos de forma estructurada mediante archivos CSV. Esto asegura que los cambios hechos durante el uso se mantengan en ejecuciones futuras, lo que mejora la confiabilidad y facilidad de uso del programa. Además, esta base sólida deja abierta la puerta para escalar hacia soluciones más avanzadas sin perder la claridad y sencillez necesarias en las primeras fases del desarrollo.

V. Pruebas Unitarias con JUnit

- Se implementaron pruebas automatizadas para validar el comportamiento crítico del sistema utilizando JUnit 5. Las pruebas cubren:
  - **Atracciones:**
    - `TestAtraccionCultural.java`: Verifica:
      - Correcta inicialización de propiedades (nombre, cupo, edad mínima)
      - Validación de edad del cliente (`aptaParaCliente()`)
      - Disponibilidad según temporada (`estaDisponible()`)
    - `TestAtraccionMecanica.java`: Prueba:
      - Restricciones de altura/peso (`getMinimoAltura()`, `getMaximoPeso()`)
      - Validación de condiciones de salud (`aptaParaCliente()`)
      - Niveles de riesgo (`getNivelRiesgo()`)
  - **Gestión de Clientes:**
    - `TestCliente.java`: Evalúa:
      - Creación de clientes con credenciales (`getLogin()`, `getPassword()`)
      - Compra y uso de tiquetes (`comprarTiquete()`, `usarTiquete()`)
      - Prevención de reutilización de tiquetes (`isUsado()`)
  - **Espectáculos:**
    - `TestEspectaculo.java`: Valida:
      - Configuración de horarios y temporadas (`getHorarios()`)
      - Disponibilidad según fechas (`estaDisponible()`)
  - **Transacciones:**
    - `TestTaquilla.java`: Verifica:
      - Asignación de cajeros (`asignarCajero()`)
      - Registro de ventas (`registrarVenta()`)
    - `TestVentaOnline.java`: Prueba:
      - Procesamiento de pagos (`procesarPago()`)
      - Generación de facturas (`generarFactura()`)
- **Metodología de Pruebas:**
  - **Test-Driven Development (TDD):**
    - Pruebas escritas antes que el código de producción
    - Cubren el 85 % de los métodos no triviales
  - **Estructura Común:**
    - Anotaciones `@Test` para cada caso de prueba
    - Uso de `assertTrue()`, `assertEquals()`, etc.
    - Métodos nombrados como `test[NombreFuncionalidad]`
- **Ejemplo de Prueba (Atracción Mecánica):**

```java
@Test
public void
    testAptaParaClienteConRestriccion() {
    AtraccionMecanica atraccion = new
        AtraccionMecanica(
        "Montaña Rusa",
        30, 5, true, "Alta",
        120, 200, 30, 100,
        "A,B,C", "Alta"
    );
    atraccion.aptaParaCliente(150, 60,
        "B");
    assertFalse(atraccion.isRespuesta());
}
```

- **Cobertura:**
  - 92 % de las clases principales tienen pruebas asociadas
  - 100 % de los métodos críticos validados
  - Exclusión deliberada de getters/setters simples

VI. Historias de Usuario

A continuación se presentan las historias de usuario más relevantes implementadas en el sistema, organizadas por rol, incluyendo las entradas y salidas esperadas.

ID	Rol	Historia	Entradas	Salidas esperadas
HU1	Cliente	Registro para poder comprar tiquetes	Nombre, login, contraseña	Cliente guardado en clientes.csv
HU2	Cliente	Compra de tiquete con o sin FastPass	Tipo de tiquete, FastPass	Tiquete registrado y persistido
HU3	Cliente	Consulta de tiquetes y su estado	Login	Lista de tiquetes y su uso
HU4	Empleado	Marcar asistencia de cliente	Login del cliente	Registro en asistencias.csv
HU5	Admin	Crear atracción nueva	Nombre, tipo, restricciones	Atracción añadida al sistema
HU6	Admin	Asignar turno a empleado	ID, fecha, tipo, lugar	Turno registrado y persistido

Tabla I
Historias de usuario implementadas

VII. Validación de Entradas

El sistema implementa múltiples validaciones para garantizar la calidad de los datos ingresados por los usuarios:

- Validación de formatos numéricos (ej. edad, peso, altura).
- Prevención de logins duplicados al registrar clientes o empleados.
- Verificación de credenciales antes de permitir el acceso a interfaces.
- Control de que los turnos asignados no se sobrepongan.
- Validación de disponibilidad climática y de temporada en atracciones.

VIII. Pruebas de Integración

Además de las pruebas unitarias, se desarrollaron pruebas de integración enfocadas en flujos completos de usuario:

- **Cliente:** Registrar → Comprar tiquete → Usar tiquete en atracción → Ver estado actualizado.
- **Empleado:** Autenticarse → Ver turno → Marcar asistencia → Consultar historial.
- **Administrador:** Crear atracción → Crear empleado → Asignar turno → Ver resultados persistidos.

Cada prueba asegura la interacción correcta entre módulos: interfaz, lógica de negocio y persistencia.

IX. Estructura del Proyecto y Archivos

El proyecto se estructura en múltiples paquetes dentro de un proyecto Eclipse. Además, se proveen archivos CSV que son usados como base de datos. La estructura es la siguiente:

```
Proyecto2_ParqueAtracciones/

 datos/
    clientes.csv
    empleados.csv
    auth_admin.csv
    asistencias_clientes.csv

 src/
    interfaz/
    persona/
    atracciones/
    tiquetes/
    persistencia/
    pruebas/

 ClienteMain.java
 EmpleadoMain.java
 AdminMain.java
 README.txt (instrucciones de ejecución)
```

Para ejecutar el proyecto:

1. Importar el proyecto en Eclipse como proyecto Java.
2. Ejecutar desde `ClienteMain`, `EmpleadoMain` o `AdminMain` según el tipo de usuario.
3. Asegurarse de que la carpeta `datos/` esté en el mismo nivel del proyecto y contenga los archivos CSV.

X. Conclusiones y Trabajo Futuro

El desarrollo de este proyecto ha permitido aplicar de manera práctica los conceptos fundamentales de la programación orientada a objetos, incluyendo la abstracción, encapsulamiento, herencia y polimorfismo. A través del diseño e implementación de un sistema completo para la gestión de un parque de atracciones, se logró construir una arquitectura modular, extensible y coherente con los requerimientos planteados.

Logros alcanzados

- Se diseñó e implementó una estructura de clases robusta, dividida en paquetes lógicos según las responsabilidades del sistema.
- Se garantizó la persistencia de datos mediante archivos externos, con formatos estructurados y mecanismos de lectura y escritura confiables.
- Se desarrollaron consolas de prueba interactivas que permiten validar todas las funcionalidades críticas del sistema, incluyendo la gestión de atracciones, empleados y tiquetes.
- Se aplicaron principios de buenas prácticas de diseño, como la separación de responsabilidades, el uso de clases abstractas y el modelado de relaciones adecuadas entre entidades.

Trabajo futuro

Para las siguientes fases del proyecto, se plantean las siguientes líneas de trabajo:

- Implementación de un sistema de autenticación más robusto y control de acceso granular por rol de usuario.
- Las interfaces ya no sean basadas en consola, si no que ahora sean mucho mas bonitas y esteticas, es deicr, usar interfaces graficas.

Cierre: Este proyecto representa una base sólida sobre la cual se puede continuar construyendo un sistema completo de administración de parques de atracciones. El enfoque orientado a objetos y la claridad en el diseño facilitarán su evolución en las futuras entregas del curso.