Samuel Ramirez

Julián Ramos

ISIS 1226

EXPLICACION CODIGO

El sistema desarrollado tiene como objetivo gestionar de manera integral las operaciones de un parque de atracciones, abarcando desde la administración de personas, atracciones, espectáculos, tiquetes y turnos de trabajo, hasta la venta y validación de accesos. Todo el proceso de desarrollo se llevó a cabo aplicando los principios de la programación orientada a objetos (POO), con un enfoque riguroso en conceptos clave como herencia, polimorfismo, encapsulamiento y abstracción. Se buscó construir una arquitectura robusta, organizada y lo suficientemente flexible como para adaptarse a futuras necesidades, cumpliendo a la vez con las directrices establecidas en el enunciado del proyecto.

1. Estructura general del sistema

Desde el inicio, la estructura del sistema se diseñó pensando en una separación clara de responsabilidades. La lógica se organizó en paquetes funcionales que encapsulan distintos componentes del sistema. Esta división favorece la escalabilidad, facilita el mantenimiento y mejora la comprensión del código en general.

Los paquetes principales son los siguientes:

1.1. logica.personas

Este paquete agrupa todas las clases relacionadas con los distintos perfiles humanos dentro del parque. Se parte de una clase abstracta Persona, que define los atributos comunes a todos los actores: nombre, login, contraseña, fecha de nacimiento, altura, peso, enfermedades o discapacidades y turnos asignados. A partir de esta base, se definen las siguientes clases concretas:

- **Cliente**: almacena la lista de tiquetes comprados y dispone de métodos para verificar el acceso a atracciones.
- **Cajero**: lleva registro de los tiquetes vendidos y posee funciones para su validación.
- **OperadorAtracciones**: incluye un atributo nivelRiesgoCapacitado y puede ser asignado solo a atracciones compatibles con su nivel de formación.
- **Cocinero**: posee un atributo booleano que indica si está capacitado para manipular alimentos.
- **ServicioGeneral**: representa al personal encargado de limpieza y mantenimiento, con asignaciones rotativas.

- **Administrador**: actúa como controlador principal del sistema, con capacidad para crear empleados, registrar atracciones, asignar turnos y manejar la persistencia de datos.

También se incluye la clase Turno, que contiene información sobre el lugar de trabajo, horario (apertura y cierre) y la tarea asignada al empleado.

1.2. logica.atracciones

Este paquete reúne todo lo relacionado con las atracciones del parque. Se construyó una jerarquía de clases que permite representar distintos tipos de atracciones:

- **Atraccion (abstracta)**: define atributos generales como nombre, ubicación, capacidad máxima, número mínimo de empleados, nivel de exclusividad, restricciones, temporada y operadores asignados.
- **AtraccionMecanica**: incorpora restricciones de altura, peso y nivel de riesgo (bajo, medio o alto).
- **AtraccionCultural**: añade una restricción basada en edad mínima.
- **Espectaculo**: es una clase aparte que modela eventos específicos con horario y temporada, sin heredar de Atraccion.

También se encuentra la clase Temporada, que sirve para controlar la vigencia tanto de atracciones como de espectáculos, permitiendo modelar comportamientos estacionales del parque.

1.3. logica.tiquetes

Este paquete se encarga de representar los distintos tipos de tiquetes disponibles. Todos heredan de la clase abstracta Tiquete, que define los atributos y métodos comunes: identificador único, estado (activo o usado), y métodos como usar(), permiteAcceso() y estaActivo().

Los tipos de tiquetes implementados son:

- **TiqueteRegular**: incluye un nivel de exclusividad (familiar, oro, diamante).
- **TiqueteTemporada**: permite el ingreso durante un período determinado.
- **TiqueteIndividual**: habilita el acceso a una atracción específica, una única vez.
- **FastPass**: otorga ingreso prioritario a determinadas atracciones en una fecha concreta.

Además, se incluyen las clases VentaOnline y VentaTaquilla, que modelan las dos modalidades de compra, guardando detalles del cliente y el medio de pago utilizado. La clase GeneradorId se encarga de generar identificadores únicos para los tiquetes, asegurando su trazabilidad.

1.4. persistencia

Este paquete aloja la clase ArchivoPlano, cuya responsabilidad es gestionar la lectura y escritura de archivos .csv. Gracias a esta clase, es posible mantener la persistencia de datos entre sesiones. Se encarga de almacenar y recuperar información relacionada con:

- Personas (clientes y empleados)
- Tiquetes emitidos
- Atracciones y espectáculos
- Turnos asignados

Este enfoque permite que el sistema conserve su estado, incluso después de ser cerrado, lo cual resulta clave para su funcionamiento en el mundo real.

1.5. presentacion

Aquí se encuentra la clase principal que ejecuta las pruebas del sistema. A través de esta clase se simulan diversos flujos funcionales como la creación de empleados, registro de atracciones, compra de tiquetes y validación de accesos. Aunque no hay una interfaz gráfica, esta clase permite verificar el correcto funcionamiento del sistema de forma integral.

2. Cumplimiento de requisitos funcionales

El sistema implementado cubre por completo los requisitos establecidos tanto a nivel funcional como no funcional. Algunos de los puntos clave incluidos son:

- Modelado completo de todos los perfiles de personas, con sus atributos y comportamientos definidos.
- Validación del acceso a las atracciones, considerando condiciones físicas, médicas, restricciones por temporada o incluso por clima.
- Representación adecuada de las atracciones, distinguiendo correctamente entre tipos culturales y mecánicos.
- Implementación de todos los tipos de tiquetes, con lógica asociada para el control de accesos.
- Integración de mecanismos de venta tanto en línea como en taquilla.
- Gestión y validación de turnos de trabajo, respetando las compatibilidades por tipo de rol.
- Persistencia bien organizada mediante archivos externos, permitiendo que toda la información crítica se conserve adecuadamente.

3. Resolución de problemas y decisiones de implementación

Durante el desarrollo del sistema se presentaron varios retos técnicos. Algunas decisiones clave que marcaron la diferencia fueron:

- **Modularización del sistema**: se definieron paquetes con responsabilidades bien delimitadas, reduciendo el acoplamiento entre módulos y facilitando el mantenimiento y pruebas.
- **Uso efectivo de herencia y polimorfismo**: se evitó la duplicación de código reutilizando atributos y comportamientos comunes en clases abstractas como Tiquete, Persona y Atraccion.
- **Restricciones flexibles en atracciones**: las condiciones para acceder a una atracción se modelaron como una lista dinámica, lo que permite aplicar múltiples reglas sin necesidad de codificar lógicas fijas.
- **Encapsulamiento de la persistencia**: se mantuvo la lógica de negocio completamente separada de los procesos de lectura y escritura, centralizando esta funcionalidad en la clase ArchivoPlano.
- **Pruebas incrementales**: se aplicó un enfoque iterativo, probando cada módulo por separado y luego en conjunto, lo que permitió detectar errores a tiempo y asegurar un comportamiento estable en cada fase del desarrollo.

Conclusión

En resumen, el sistema desarrollado representa una solución completa y bien fundamentada para el problema planteado. No solo se respetaron los principios fundamentales de diseño orientado a objetos, sino que también se adoptaron buenas prácticas de ingeniería de software que garantizan su mantenibilidad y capacidad de evolución.

El diseño modular, el uso adecuado de herencia y polimorfismo, y la separación entre lógica y persistencia dan como resultado un sistema preparado para escalar, al que fácilmente podrían añadirse funcionalidades futuras como generación de reportes, control de inventarios, historial de accesos o integración con bases de datos más complejas.

Además, la experiencia adquirida a lo largo del desarrollo contribuyó al fortalecimiento de habilidades clave en análisis, modelado de dominios, representación de restricciones y control de operaciones. Todo esto se ve reflejado en una solución sólida, coherente y funcional, que simula fielmente las dinámicas reales de un parque de atracciones digitalizado.