Informe de Práctica: Sistema de Gestión de Reservas Deportivas

Asignatura: Programación Web
Universidad: Universidad de Córdoba
Integrantes del Equipo: [Belén María Estepa Reboso, Manuel Eddy Ruiz Rivera, Juan Francisco Luna García, Pablo García Izquierdo]
Fecha de Entrega: [28/10/2024]

1. Introducción

El objetivo de esta práctica ha sido desarrollar un sistema en Java para la gestión de reservas deportivas, enfocado en tres tipos de reservas: infantiles, familiares y para adultos. A lo largo del desarrollo, hemos aplicado conceptos avanzados de programación orientada a objetos (POO), implementado patrones de diseño y seguido buenas prácticas de desarrollo. El sistema contempla la gestión de usuarios, pistas y reservas mediante la integración de múltiples clases y patrones que permiten una estructura modular y escalable.

2. Estructura del Proyecto y Decisiones de Diseño

El sistema se organiza en varias clases principales y utiliza un patrón Factoría para la creación de reservas de diferentes tipos. Las decisiones de diseño adoptadas están fundamentadas en la búsqueda de una estructura modular, reutilizable y que permita la extensión futura de funcionalidades.

- **Gestor de Usuarios (GestorUsuarios)**: Se encarga de almacenar y gestionar la información de los usuarios que podrán realizar reservas. Esta clase mantiene una lista de objetos de tipo Jugador y permite operaciones de búsqueda y registro. Su implementación permite validar que los usuarios estén registrados previamente antes de hacer una reserva, asegurando integridad en el sistema.

- **Gestor de Pistas (GestorDePistas)**: Maneja la información sobre las pistas deportivas disponibles. Cada pista tiene atributos como su tamaño, capacidad máxima y tipo de uso permitido. El diseño modular de esta clase facilita la consulta de disponibilidad de las pistas y la verificación de su compatibilidad con el tipo de reserva solicitado.

- **Gestor de Reservas (GestorReservas)**: La clase central de la aplicación. Este gestor controla la creación, modificación y cancelación de reservas. Aquí, se ha utilizado el **Patrón de Diseño Factoría** mediante la clase ReservaFactory, permitiendo instanciar distintos tipos de reservas (infantil, familiar y adultos) de acuerdo con las necesidades del usuario.

- **Tipos de Reserva**: Se definieron clases especializadas (ReservaAdultos, ReservaInfantil, ReservaFamiliar), que heredan de la clase base abstracta Reserva. Cada subclase incluye la información y lógica adicional específica para el tipo de reserva que representa, lo que permite que la lógica y atributos adicionales para cada tipo de reserva se manejen de forma clara y extendible.

- **Clases Auxiliares y Enumeraciones**:
    - **Material**: Representa los materiales deportivos (como pelotas, aros, etc.) que pueden asociarse a una reserva.
    - **TipoReserva y TamanioPista**: Se utilizaron enumeraciones para definir los diferentes tipos de reserva y tamaños de pista, permitiendo una tipificación y validación sencilla en el código.
    - **Bono**: Clase que permite manejar las reservas mediante bonos, implementando descuentos adicionales por antigüedad y sesiones acumuladas.

3. Fuentes Consultadas

Para la realización de este proyecto, el equipo consultó diversas fuentes:

- **Repositorio GitHub**: Herramienta principal para buscar ejemplos de implementación en Java, analizar soluciones similares y revisar buenas prácticas en programación orientada a objetos y patrones de diseño.
- **Documentación de Java**: La documentación oficial de Java fue fundamental para revisar métodos específicos de clases como LocalDateTime, Date, List, y Stream, entre otros. Documentación Java
- **Moodle de la Universidad de Córdoba**: Consultamos apuntes, ejemplos de la asignatura y recomendaciones del profesorado en la plataforma Moodle, que fueron de gran ayuda para guiar la estructura y modularidad del proyecto.
- **Blogs y foros de desarrolladores (Stack Overflow, Baeldung)**: Utilizamos recursos como Stack Overflow para resolver problemas puntuales de implementación y Baeldung para mejorar nuestra comprensión de patrones de diseño en Java y de la API de tiempo (java.time).

4. Dificultades Encontradas y Soluciones

Durante el desarrollo del proyecto, el equipo se encontró con diversos desafíos. A continuación, se detallan algunos de los problemas más importantes y las soluciones adoptadas:

1. **Gestión de Fechas**: La transición entre Date y LocalDateTime fue compleja, debido a la compatibilidad limitada entre ambas clases. Optamos por convertir todas las operaciones a LocalDateTime, lo que simplificó las comparaciones de fechas y horas y permitió el uso de métodos como isAfter().
2. **Aplicación del Patrón Factoría**: Inicialmente, la creación de los diferentes tipos de reservas estaba dispersa en la clase GestorReservas. Esto complicaba el mantenimiento y la legibilidad. Decidimos aplicar el patrón Factoría a través de ReservaFactory, centralizando la lógica de creación y aumentando la extensibilidad del sistema.
3. **Validaciones y Consistencia de Datos**: Una de las tareas más complejas fue asegurar que el sistema verificara correctamente la disponibilidad de las pistas, la existencia de usuarios registrados y los descuentos aplicables. Implementamos métodos de validación dedicados, como puedeRealizarReserva() y esPistaAdecuada(), para asegurar que cada reserva cumpliera con las condiciones requeridas.
4. **Herencia y Polimorfismo en las Clases de Reservas**: La necesidad de agregar atributos específicos a cada tipo de reserva, como el número de niños o adultos, hizo que la herencia fuera la solución ideal. Sin embargo, la implementación inicial sobrecargaba el constructor de Reserva. Refinamos el diseño distribuyendo los atributos específicos en subclases y creando un método toString() en cada una para imprimir la información de cada reserva de manera detallada y personalizada.

5. Conclusión

El desarrollo de este sistema nos permitió aplicar múltiples conceptos de programación orientada a objetos, como herencia, polimorfismo y encapsulación, en un entorno modular. La implementación del patrón Factoría aportó flexibilidad y claridad en la creación de reservas, y el uso de clases especializadas para cada tipo de reserva hizo que el sistema fuera más estructurado y fácilmente extensible. A través de este proyecto, el equipo consolidó su conocimiento en Java y en el diseño de sistemas escalables, reforzando la importancia de buenas prácticas y la planificación previa en la implementación de software.

Este informe resume el proceso de desarrollo y las decisiones de diseño de nuestro sistema de gestión de reservas deportivas, detallando tanto las soluciones técnicas adoptadas como las dificultades encontradas y las fuentes consultadas.