Informe práctica 1

i02abhak, i12argoi, i02fujuj, i02topap

Introducción

En esta práctica se ha desarrollado una aplicación en **Java** usando **Eclipse IDE for Java Web Development**. El proyecto se ha trabajado en sistemas **Debian Bookworm**, **macOS** y **Windows 10/11**, garantizando su ejecución multiplataforma. Se ha usado **Javadoc** para generar documentación del código y el seguimiento y control de versiones se ha realizado a través de un repositorio en **GitHub**:
`https://github.com/Kabhad/GM1_i02abhak.git`.

Ejercicio 1

En este ejercicio, se han desarrollado las clases necesarias para representar los conceptos principales del dominio de la aplicación de gestión de reservas de pistas de baloncesto. Estas clases se distribuyen en tres paquetes principales:

- **Paquete jugador**: Contiene la clase `Jugador`, que representa a los usuarios de las instalaciones.
- **Paquete material**: Contiene las clases `Material`, junto con los enumerados públicos `TipoMaterial` y `EstadoMaterial`, que representan los tipos de materiales (pelotas, canastas y conos) y los estados del material (disponible, reservado y en mal estado).
- **Paquete pista**: Contiene la clase `Pista` y la enumeración pública `TamanoPista`, que representa las pistas y sus características.

Cada una de estas clases ha sido implementada siguiendo las especificaciones del enunciado, asegurando que se cumplan los requisitos de los atributos y métodos, tales como:

- Constructores vacío y parametrizado.
- Métodos `get/set` para todos los atributos.
- Métodos específicos como `consultarMaterialesDisponibles` en la clase `Pista` y `calcularAntigüedad` en la clase `Jugador`.

Ejercicio 2

En este ejercicio se implementaron clases para gestionar reservas aplicando un patrón Factoría, con el objetivo de proporcionar una estructura flexible y escalable para la gestión de distintos tipos de reservas.

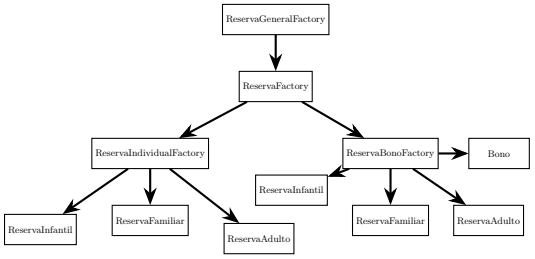
Clases de reserva

Se definieron tres tipos de reservas:

- **ReservaInfantil**: Exclusiva para niños, en pistas tipo minibasket.
- **ReservaFamiliar**: Para adultos y niños, en pistas minibasket o 3vs3.
- **ReservaAdulto**: Exclusiva para adultos, en pistas específicas.

Cada clase incluye su constructor, métodos `get/set` y un método `toString`.

Patrón Factoría



El patrón se implementó en dos niveles:

- **Reserva Individual**: Genera reservas de tipo infantil, familiar o de adulto, aplicando descuentos por antigüedad.
- **Reserva Bono**: Crea reservas de bono, con un 5 % de descuento adicional.
- **Factoría General**: Decide si la reserva es individual o de bono y usa la factoría adecuada.

Beneficios del Doble Patrón Factoría

Este enfoque proporciona:

- **Flexibilidad**: Facilita añadir nuevos tipos de reservas.
- **Separación de responsabilidades**: Cada factoría maneja un tipo específico.
- **Extensibilidad**: Permite incluir nuevas modalidades sin afectar el código.
- **Reutilización de código**: Reduce la duplicación de lógica común.

Ejercicio 3

En este ejercicio, se desarrollaron tres clases gestoras que implementan las funcionalidades necesarias para el programa principal, conforme a las especificaciones del enunciado.

Gestor de usuarios

Esta clase gestiona la información de los usuarios que realizan reservas. Las funcionalidades implementadas incluyen:

- Dar de alta a un usuario, comprobando que no esté registrado previamente.
- Modificar la información de un usuario registrado.
- Listar los usuarios actualmente registrados.

Gestor de pistas

El gestor de pistas maneja las instalaciones y el material asociado. Las funcionalidades implementadas son:

- Crear pistas y materiales.
- Asociar materiales a pistas disponibles, verificando que no estén asignados a otras pistas o en mantenimiento. Además, se respetan las restricciones entre tipos de pistas (interior y exterior) y materiales.
- Listar las pistas no disponibles.
- Dado un número de jugadores y un tipo de pista, devolver las pistas que estén libres y que puedan soportar ese número de jugadores.

Gestor de reservas

El gestor de reservas permite a los usuarios gestionar sus reservas de pistas. Las funcionalidades incluyen:

- Hacer reservas individuales para usuarios registrados, aplicando un 10 % de descuento si tienen más de 2 años de antigüedad.
- Hacer reservas dentro de un bono con 5 sesiones, aplicando un 5 % de descuento respecto al precio individual. Los bonos caducan un año después de la primera reserva.
- El precio varía según la duración: 60 minutos (20€), 90 minutos (30€) y 120 minutos (40€).
- Modificar o cancelar reservas hasta 24 horas antes.
- Consultar reservas futuras.
- Consultar reservas para un día y pista específicos.

Implementación

Se implementaron las clases gestoras con todas las funcionalidades requeridas, junto con la clase `Bono` para gestionar la información de los bonos.

Se desarrollaron menús interactivos (*displays*) para la gestión por consola, usando una única instancia de `Scanner` en el `mainPrincipal`, compartida con los demás *displays*.

El patrón Singleton se aplicó en las instancias y listas de los gestores, garantizando una única instancia que mantenga la integridad de los datos. Los métodos para cargar y guardar datos usan ficheros de texto, con rutas gestionadas mediante un fichero de propiedades accesible con `java.util.Properties`, trabajando con rutas relativas.

La información se carga al inicio, se actualiza durante la ejecución, y se guarda nuevamente al finalizar desde el menú principal.

Implementaciones adicionales

Se ha añadido una funcionalidad en el gestor de jugadores que permite dar de baja a un usuario de manera lógica, sin eliminarlo del sistema. Esto se realiza mediante un nuevo atributo en la clase `Jugador`, que desactiva la cuenta del usuario, permitiendo que sus datos permanezcan en el sistema sin que pueda realizar nuevas reservas. De esta forma, se conserva la información histórica de las reservas sin perder datos importantes.

Dificultades durante la realización de la práctica

Durante el desarrollo, surgieron varias dificultades que se resolvieron de la siguiente manera:

- **Importación del proyecto**: Al clonar el proyecto en macOS, el JDK no se cargaba automáticamente. Se resolvió seleccionando manualmente la versión correcta del JDK en las propiedades del proyecto.
- **Patrón Factoría**: Al principio, el patrón Factoría no gestionaba correctamente los tipos de reservas. Esto llevó a implementar un patrón de Factoría doble, permitiendo una gestión eficiente de las reservas individuales y de bono.
- **Manejo del objeto `Scanner`**: El uso de múltiples instancias de `Scanner` en diferentes menús causaba problemas. Se solucionó creando una única instancia en el `mainPrincipal`, que se pasó a los demás menús para asegurar un control correcto de la entrada por consola.

Referencias

[1] Oracle. Appendable class documentation, 2023. Accessed: 2024-10-15.

[2] Oracle. Ejb singleton class documentation, 2023. Accessed: 2024-10-08.

[3] Oracle. File class documentation, 2023. Accessed: 2024-10-8.

[4] Oracle. Java language specification se7, 2023. Accessed: 2024-10-20.

[5] Oracle. Rmi factory documentation, 2023. Accessed: 2024-10-22.

[6] Oracle. Scanner class documentation, 2023. Accessed: 2024-10-13.