INFORME DE TRABAJO
PRÁCTICA FINAL
PROGRAMACIÓN III

Jesús Parejo Pineda - DNI: 53737163D

Universidad de Salamanca

12 de diciembre, 2024

Introducción

El objetivo fundamental de este proyecto es el desarrollo de una aplicación Java que permita a los usuarios (estudiantes) gestionar sus tareas. La aplicación debe cumplir una serie de requisitos (multiplataforma, enfoque modular, escalabilidad) y las funcionalidades clave son las operaciones C.R.U.D. de tareas, la importación/exportación de datos en formato CSV y JSON y la persistencia de datos mediante serialización.

Procedimiento

He llevado a cabo el desarrollo de la aplicación basándome en la teoría estudiada y los ejemplos vistos en clase, además he buscado información extra para facilitar la implementación de algunas funcionalidades (por ejemplo en el uso de la librería Date y SimpleDateFormat). He seguido el patrón M.V.C. y dividido el proyecto en varias etapas de desarrollo, a continuación mostraremos una por una las distintas funcionalidades del programa y cómo las he implementado siguiendo como criterio el orden dado por el ciclo de vida de la aplicación.

1. Clase App

La clase App es el punto de entrada de la aplicación, en ella se inicializan los componentes del MVC. Su ejecución comienza con el método main(String[] args) cuyos argumentos se interpretan para determinar la vista y el repositorio que se deben utilizar (en mi caso las únicas opciones desarrolladas por ahora son la vista por consola “InteractiveView” y el repositorio “BinaryRepository”. Por defecto se configuran las opciones predeterminadas.
A continuación se crean las instancias de los elementos Model, BaseView y Controller y finalmente se inicia el programa llamando al método init() del controlador.

2. Paquete view

2.1. Clase BaseView

Se trata de una clase abstracta en la que se definen los métodos básicos que deben ser implementados por cualquier tipo de vista en la aplicación indistintamente de si es por consola, con submenús, narrando las opciones, etc. Su propósito es, por tanto, proveer una interfaz común a cualquier vista que se implemente.

2.2. Clase BaseView

La clase InteractiveView es una implementación concreta de la clase abstracta BaseView que. El primer método en ejecutarse es init() que muestra un mensaje de bienvenida y a continuación invoca el método showMainMenu() en el que se procede a la interacción con el usuario hasta terminar con el método end(). Profundicemos en el menú; este muestra al usuario tres opciones:

```
Bienvenido a la aplicación de gestión de tareas.

 --- Menú principal (submenús) ---
1. Operaciones CRUD
2. Importación/Exportación
3. Salir
```

2.2.1. Operaciones CRUD

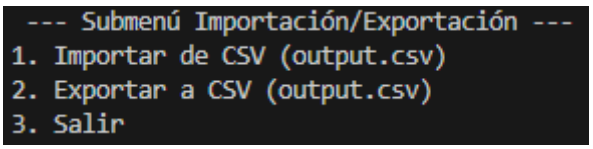
```
 --- Submenú CRUD ---
1. Agregar tarea
2. Mostrar tareas sin completar por prioridad (mayor a menor)
3. Mostrar todas las tareas (completadas o no)
4. Marcar tarea como completa/pendiente (cambia su estado)
5. Modificar tarea
6. Eliminar tarea
7. Volver al menú principal
```

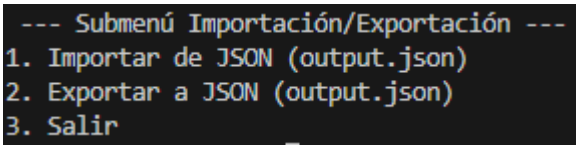
Remite a un submenú con las siguientes opciones:

1. Agregar tarea: Pide al usuario los datos de la nueva tarea y llama al método agregarTarea() del controlador pasándole como argumento una Task con los parámetros dados por el usuario, si todo va bien devuelve un mensaje de “éxito” y si se lanza la excepción RepositoryException comunica el problema al usuario.
2. Mostrar tareas sin completar: Llama al controlador que le proporciona la lista de tareas ordenada por prioridad y con el atributo “Completada = false” y lo muestra por pantalla.
3. Mostrar todas las tareas: Muestra por pantalla todas las tareas obtenidas, en última instancia, del repositorio.

4. Marcar tarea como completa/pendiente: Pide al usuario el título de la tarea a modificar su estado y llama al método cambiarEstadoTarea() del controlador pasando como argumento una tarea vacía con el título proporcionado por el usuario.
5. Modificar tarea: Pide al usuario el identificador de la tarea que quiere modificar y luego, de manera análoga a agregarTarea(), pide el resto de atributos para modificar, finalmente llama al controlador y muestra por pantalla el éxito o fracaso de la modificación.
6. Eliminar tarea: De manera similar pide al usuario el identificador de la tarea que quiere eliminar y le pasa el controlador una tarea vacía con ese identificador.
7. Volver al menú principal: Se sale del bucle do-while.

2.2.2. Importación/Exportación





El usuario debe elegir entre dos tipos de Importación/Exportación, o bien a través de ficheros .csv o bien a través de ficheros .json, en un caso u otro redirigimos a un submenú con métodos análogos en cada caso que se unen en el controlador, es decir, nuestro controlador no tiene conocimiento sobre si se ha elegido .csv o .json, el modelo sí tendrá el exportador inicializado. Cabe destacar que los modelos de importación y exportación del modelo son propios de la interfaz IExporter, no son particulares del CSVExporter o el JSONExporter, tal y como se pedía en la guía.

3. Paquete controller

Únicamente hace de intermediario entre la vista y el modelo y sus funcionalidades son mayormente sencillas.

4. Paquete model

4.1. Clase Model

Esta clase instancia el repositorio y el exportador, será la encargada de manejar la lógica de negocio del programa, o más bien de dirigir las órdenes procedentes del controlador al repositorio y al exportador, pues en sí mismo, el modelo no tiene ni los métodos CRUD ni los métodos de importación/exportación.
Cabe destacar el método obtenerTareasSinCompletar(), que funciona de la siguiente manera:

- Obtiene las tareas del repositorio.
- Elimina mediante iterator las tareas completas
- Ordena las tareas mediante comparator según su prioridad en orden descendente.

4.2. Clase BinaryRepository

La clase BinaryRepository implementa la interfaz IRepository y actúa como un repositorio para gestionar tareas (Task) a través del almacenamiento en un archivo binario.
A continuación, se detallan sus principales características:

Atributos:

- tareas: Lista de tareas (ArrayList<Task>) que se gestiona en memoria.
- ficheroEstadoSerializado: Archivo (File) donde se guardan y cargan las tareas en formato binario (tasks.bin).

Métodos principales:

1. CRUD (Crear, Leer, Actualizar, Eliminar)

- addTask(Task t): Agrega una tarea a la lista. Lanza una excepción si la tarea ya existe.
- modifyTask(Task task): Modifica una tarea existente basándose en su identificador. Lanza una excepción si no se encuentra.
- removeTask(Task task): Elimina una tarea según su identificador. Lanza una excepción si no se encuentra.
- getAllTasks(): Devuelve una copia de todas las tareas en la lista.
- cambiarEstadoTarea(Task task): Cambia el estado de completado de una tarea (true/false) basada en su título.

2. Persistencia de datos

- saveData(): Guarda la lista de tareas en un archivo binario (tasks.bin). Devuelve true si tiene éxito, o false en caso de error.
- loadData(): Carga la lista de tareas desde el archivo binario si este existe. Devuelve la lista cargada o null si ocurre un error.