2024

JTASKS 2.0

Raúl Vicario García

71482159E

Introducción al proyecto

La presente práctica final de Programación III tiene como objetivo desarrollar una aplicación de gestión de tareas llamada jTasks, diseñada para estudiantes universitarios. Esta herramienta busca facilitar la organización y seguimiento de las actividades académicas a lo largo del curso, abordando problemas comunes como la falta de planificación, la procrastinación y el cumplimiento de plazos.

La aplicación, desarrollada en Java, utiliza una arquitectura Modelo-Vista-Controlador (MVC), lo que asegura una separación clara entre la lógica de negocio, la presentación y la interacción con el usuario. Inicialmente se implementa como una aplicación de consola interactiva, con vistas a futuras extensiones que incluyan interfaces gráficas o control por voz.

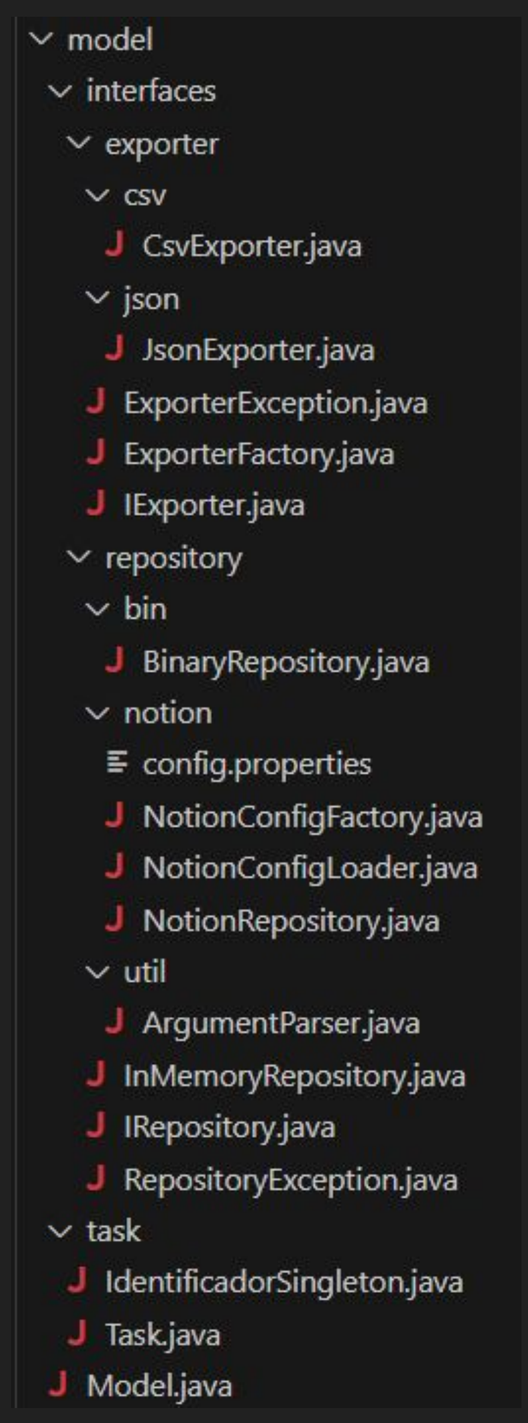
Entre sus funcionalidades principales, jTasks permite realizar operaciones CRUD sobre las tareas (crear, listar, modificar y eliminar), gestionarlas por prioridad o estado, y exportar/importar datos en formatos JSON y CSV. Además, incorpora un sistema de persistencia que, según los parámetros de configuración, puede almacenar los datos localmente mediante serialización o integrarse con bases de datos de Notion, utilizando su API oficial.

Este informe detalla el diseño, desarrollo y funcionamiento de la aplicación, abordando cada uno de los requisitos planteados y demostrando su implementación mediante capturas de pantalla y explicaciones técnicas.

Asimismo, se describen las problemáticas encontradas durante el desarrollo, junto con las posibles mejoras y líneas futuras de evolución. jTasks se presenta como una solución robusta, flexible y extensible para la gestión eficiente del tiempo y las tareas de los estudiantes.

1. MODELO

La estructura de mi modelo es la siguiente:



Con el objetivo de distinguirme del resto, he implementado en el paso de los argumentos una tercera opción.. *CARGAR LAS CLAVES DESDE UN ARCHIVO EXTERNO*

config.properties

PracticaJavaFinal > PracticaEvaluable > jTasks > src > model > interfaces > repository > notion > config.properties

```
notion.api.key=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
notion.database.id=152b2129115080d39a01e8c742c3f7e9
```

Gracias a la clase configloader y configfactory puedo inicializar con ello el programa.

Quiero reseñar que he aplicado el método de la fábrica dado en teoría con objetivo de aplicar y que me sea de utilidad en ejemplos prácticos lo dado en clase.

NotionConfigFactory.java

PracticaJavaFinal > PracticaEvaluable > jTasks > src > model > interfaces > repository > notion > NotionConfigFactory.java > {} model.interfaces.repository.notion

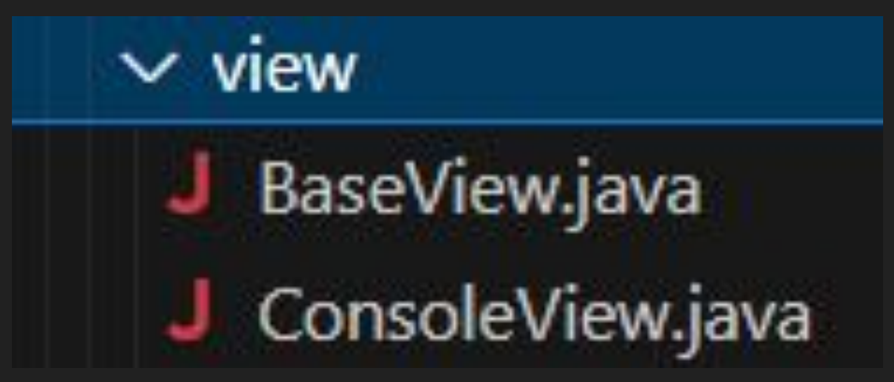
```java
package model.interfaces.repository.notion;

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public class NotionConfigFactory {
    public static Properties cargarPropiedades() {
        Properties propiedades = new Properties();
        try (InputStream input = NotionConfigFactory.class.getClassLoader()
                .getResourceAsStream(name:"config.properties")) {
            if (input == null) {
                throw new RuntimeException(message:"No se puede encontrar el archivo config.properties en el classpath.");
            }
            propiedades.load(input);
        } catch (IOException ex) {
            throw new RuntimeException(message:"Error al cargar config.properties.", ex);
        }
        return propiedades;
    }
}
```

2. VISTA

La estructura de mi vista es la siguiente:



Defino BaseView como superclase y abstraigo los métodos.

BaseView.java

PracticaJavaFinal > PracticaEvaluable > jTasks > src > view > BaseView.java > {} view

```java
package view;

import java.util.List;

import controller.Controller;
import model.task.Task;

public abstract class BaseView {
    protected Controller controller; // Atributo común para todas las vistas.

    // Constructor
    public BaseView(Controller controller) {
        this.controller = controller;
    }

    // Métodos abstractos

    public abstract void displayTasks(List<Task> tasks);
    public abstract void init(); // Inicia la vista.
    public abstract void showMessage(String message); // Muestra un mensaje.
    public abstract void showErrorMessage(String errorMessage); // Muestra un mensaje de error.
    public abstract void end(); // Finaliza la vista.
    public abstract String showMenuAndGetOption(); // Método que define cómo se muestra el menú y se obtiene la opción.
}

```

La clase ConsoleView es heredada y compleja.

Por empezar sobreescribo los métodos para dotarlo de profesionalidad y que salte como avisos de sistema.

Se pueden apreciar inyecciones de controlador y gestiones de menús.

```java
int choice = Esdia.readInt(prompt:"Seleccione una opción (1-7):", min:1, max:7);

switch (choice) {
    case 1 -> controller.addTask();
    case 2 -> controller.listTasks();
    case 3 -> controller.listPendingTasksByPriority();
    case 4 -> controller.modifyTask();
    case 5 -> controller.toggleTaskCompletion(); // Llama al método para cambiar el estado de completado
    case 6 -> controller.deleteTask();
    case 7 -> exit = true;
    default -> showErrorMessage(errorMessage:"Opción no válida. Intente de nuevo.");
```

El POJO del task es básico y sencillo y la manera con la que aseguro la unicidad del identificador es con un Singleton dado en clases de teoría y práctica.

IdentificadorSingleton.java

PracticaJavaFinal > PracticaEvaluable > jTasks > src > model > task > IdentificadorSingleton.java > {} model.task

```java
package model.task;

public class IdentificadorSingleton {
    private static IdentificadorSingleton instancia;
    private int ultimoId;

    private IdentificadorSingleton() {
        this.ultimoId = 0; // Inicia en 0 por defecto
    }

    public static IdentificadorSingleton getInstancia() {
        if (instancia == null) {
            instancia = new IdentificadorSingleton();
        }
        return instancia;
    }

    public synchronized int generarNuevoIdentificador() {
        return ++ultimoId;
    }

    public synchronized void actualizarUltimoIdentificador(int id) {
        if (id > ultimoId) {
            ultimoId = id;
        }
    }
}

```

El resto de clases no tienen ninguna diferencia reseñable, de hecho sus implementaciones aparecen en el diagrama del enunciado, como mucho InMemoryRepository que implementa la novedad que he presentado antes.

He implementado lo máximo posible el paquete de esDia, para entrada / salida y también para las rutas.

Las excepciones son personalizadas y adaptadas a cada situación.

ExporterException.java

PracticaJavaFinal > PracticaEvaluable > jTasks > src > model > interfaces > exporter > ExporterException.java > ...

```java
package model.interfaces.exporter;

// Excepción personalizada para manejo de errores en exportación/importación.

public class ExporterException extends Exception {

    public ExporterException(String mensaje, Throwable causa) {

        super(mensaje, causa);

    }

    public ExporterException(String message) {
        super(message);
    }

}
```

3. CONTROLADOR

El controlador junto a NotionRepository son las dos clases más complejas del proyecto.

Incluyo todos los métodos de gestión de tareas que me asegura el CRUD.

```java
// Método para eliminar una tarea por ID
public void deleteTask() {
    try {
        int id = Esdia.readInt(prompt:"Introduce el ID de la tarea a eliminar:");
        try {
            // Cargar todas las tareas
            List<Task> tasks = model.loadData();
            // Buscar la tarea con el ID
            Task tareaAEliminar = null;
            for (Task t : tasks) {
                if (t.getIdentificador() == id) {
                    tareaAEliminar = t;
                    break;
                }
            }

            if (tareaAEliminar != null) {
                // Llamar directamente a la capa de repositorio para archivar
                model.removeTask(tareaAEliminar);
                view.showMessage("Tarea eliminada con ID: " + id);
            } else {
                view.showMessage(message:"No se encontró ninguna tarea con el ID especificado.");
            }
        } catch (Exception e) {
            view.showErrorMessage("Error al eliminar tarea: " + e.getMessage());
        }


    } catch (Exception e) {
        view.showErrorMessage("Error al eliminar tarea: " + e.getMessage());
    }
}
```

También hay métodos para mostrar el tiempo en milisegundos y para hacer “clear”.

Por brevedad, se puede resumir estos métodos en prácticar filtros, hacer load y save con las distintas listas y colecciones y enlazar la vista y el modelo.

```java
public void start() {
    DiaUtil.startTimerMS();

    try {
        // Cargar datos iniciales y actualizar identificadores
        List<Task> tasks = model.loadData();
        view.showMessage("Datos cargados correctamente. Total de tareas: " + tasks.size());
    } catch (Exception e) {
        view.showErrorMessage("Error al cargar los datos: " + e.getMessage());
    }

    view.init();
}
```

4. APP

En esta sección no hay mucho que reseñar, básicamente es el main que orquesta toda la aplicación.

```
try {
    // Verificar los argumentos de la línea de comandos
    if (args.length < 2 || !args[0].equalsIgnoreCase(anotherString:"--repository")) {
        throw new IllegalArgumentException(s:"Uso esperado: --repository [bin|notion|memory] [API_KEY DATABASE_ID]");
    }

    String repositoryType = args[1];

    // Selección del repositorio según el argumento
    switch (repositoryType.toLowerCase()) {
        case "bin":
            repositorio = new BinaryRepository();
            break;

        case "notion":
            if (args.length < 4) {
                throw new IllegalArgumentException(s:"Uso esperado para Notion: --repository notion API_KEY DATABASE_ID");
            }
            String apiKey = args[2];
            String databaseId = args[3];
            repositorio = new NotionRepository(apiKey, databaseId);
            break;

        case "memory":
            repositorio = new InMemoryRepository(); // Repositorio por defecto en memoria
            break;

        default:
            throw new IllegalArgumentException("Tipo de repositorio no soportado: " + repositoryType);
```

Para dejar algo en constancia, con este bloque instigamos la entrada de argumentos y su corrección de tipo.

```
} catch (IllegalArgumentException e) {
    System.err.println("Error: " + e.getMessage());
    System.err.println(x:"Uso esperado:");
    System.err.println(x:"  java -jar app.jar --repository bin");
    System.err.println(x:"  java -jar app.jar --repository notion API_KEY DATABASE_ID");
    System.err.println(x:"  java -jar app.jar --repository memory");
    return;
```

Agrego este bloque de información para que sea muy intuitivo a la hora de ejecutar el programa.

5. EJEMPLO DE EJECUCIÓN

Como final del informe, asocio una ejecución del código como ejemplo para la implementación.

Entre las tres opciones a elegir (Notion, binario y claves guardadas) elijo a Notion y comunica correctamente con la bbdd.

Para ejemplo visual introduzco por teclado la opción de gestionar tareas y me abre automáticamente al menú interactivo.

Desde este puedo hacer funcional cualquier opción de CRUD que pide el enunciado.

En el caso de exportar / importar se debe elegir entre JSON y CSV.

En el caso de ver el tiempo transcurrido, sale por pantalla en MS el tiempo de ejecución.

```
PS C:\Users\Vetas\Desktop\VicarioGarciaRaul\jTasks\src> java -jar app.jar --repository notion "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" "152b2129115080d39a01e8c742c3f7e9"
[SYSTEM]: Datos cargados correctamente. Total de tareas: 16
Bienvenido a la aplicacion de gestion de tareas.

Centro de operaciones:
1. Gestionar tareas
2. Exportar/Importar tareas
3. Ver tiempo transcurrido
4. Salir
Seleccione una opcion: (1 <= numero <= 4)1

Gestion de tareas:
1. Anadir tarea
2. Listar todas las tareas
3. Listar tareas pendientes (ordenadas por prioridad)
4. Modificar tarea
5. Cambiar estado de completado de una tarea
6. Eliminar tarea por ID
7. Volver al centro de operaciones
Seleccione una opcion: (1 <= numero <= 7)2

Lista de tareas:
Tarea [ID=43248, Título=pepa, Fecha=Mon Dec 09 11:56:00 CET 2024, Contenido=pepa, Prioridad=4, Duración=11 mins, Completada=false]
Tarea [ID=43247, Título=raul, Fecha=Sun Nov 11 00:00:00 CET 2029, Contenido=raul, Prioridad=3, Duración=11 mins, Completada=true]
Tarea [ID=43246, Título=raul, Fecha=Sun Nov 11 00:00:00 CET 2029, Contenido=raul, Prioridad=3, Duración=11 mins, Completada=true]
Tarea [ID=43245, Título=pepa, Fecha=Mon Dec 09 11:56:00 CET 2024, Contenido=pepa, Prioridad=4, Duración=11 mins, Completada=false]
Tarea [ID=43244, Título=pepa, Fecha=Mon Dec 09 11:56:00 CET 2024, Contenido=pepa, Prioridad=4, Duración=11 mins, Completada=true]
Tarea [ID=43243, Título=raul, Fecha=Sun Nov 11 00:00:00 CET 2029, Contenido=raul, Prioridad=3, Duración=11 mins, Completada=true]
Tarea [ID=43242, Título=raul, Fecha=Sun Nov 11 00:00:00 CET 2029, Contenido=raul, Prioridad=3, Duración=11 mins, Completada=true]
Tarea [ID=43241, Título=pepa, Fecha=Mon Dec 09 11:56:00 CET 2024, Contenido=pepa, Prioridad=4, Duración=11 mins, Completada=false]
Tarea [ID=43240, Título=pepa, Fecha=Mon Dec 09 11:56:00 CET 2024, Contenido=pepa, Prioridad=4, Duración=11 mins, Completada=false]
Tarea [ID=43239, Título=raul, Fecha=Sun Nov 11 00:00:00 CET 2029, Contenido=raul, Prioridad=3, Duración=11 mins, Completada=true]
Tarea [ID=43238, Título=raul, Fecha=Sun Nov 11 00:00:00 CET 2029, Contenido=raul, Prioridad=3, Duración=11 mins, Completada=true]
Tarea [ID=43237, Título=pepa, Fecha=Mon Dec 09 11:56:00 CET 2024, Contenido=pepa, Prioridad=4, Duración=11 mins, Completada=true]
Tarea [ID=43236, Título=pepa, Fecha=Mon Dec 09 11:56:00 CET 2024, Contenido=pepa, Prioridad=4, Duración=11 mins, Completada=false]
Tarea [ID=43235, Título=raul, Fecha=Sun Nov 11 00:00:00 CET 2029, Contenido=raul, Prioridad=3, Duración=11 mins, Completada=true]
```

6. PROBLEMÁTICAS Y FUTURO

Bajo mi punto de vista, refactorizar el código exhaustivamente sería un buen punto de partida, seguramente con tiempo y pruebas automatizadas podría tener un margen de mejora.

En cuanto a futuro, se podría implementar el texto a voz y otras prácticas opcionales del enunciado.

La problemática mas grande que he encontrado ha sido la comunicación con la bbdd de notion y la complejidad del controlador y sus métodos.

Me ha parecido una práctica muy interesante y con una gran iniciativa por detrás, enhorabuena a los desarrolladores de esta.