Práctica 1

1. Miembros del Equipo

Miguel Orzáez Pintor (**i12orpim**) GM3
Álvaro Prieto Cano (**i12prcaa**) GM2
José Andrés Trujillo Henares (**f92trhej**) GM3
Alberto Quesada Valle (**i12quvaa**) GM3

2. Decisiones de metodología.

2.1. Implementación del código en inglés.

Hemos decidido programar todas las clases y métodos en inglés por una serie de factores:

- Estándar de la industria**.** Al utilizar el inglés en nuestro código y documentación, nos alineamos con las convenciones y prácticas comunes en la industria del software. Además de que la mayoría de la documentación y los recursos están en inglés.

- Consistencia y legibilidad**.** Al utilizar un solo lenguaje en nuestro código reducimos la posibilidad de confusión y errores causados por mezclar diferentes idiomas, además de mejorar la legibilidad.

2.2. Desarrollo guiado por pruebas (TDD).

- Garantía de calidad**.** Si en algún momento el código no se comporta como se espera, los tests nos proporcionarán un indicador claro de dicho problema. Esto nos permite corregir los errores de manera temprana y asegurarnos de que el código cumple con los requisitos definidos en los tests a partir del enunciado del ejercicio.

- Ayuda en el desarrollo de nuevas funcionalidades y la factorización**.** Muchas veces ocurre que, durante el desarrollo de una nueva funcionalidad o refactorización, rompemos otras que ya funcionaban antes sin darnos cuenta. En cambio, si tenemos una suite de tests y rompemos alguna funcionalidad existente, estos nos alertarán de inmediato, pudiendo identificar el problema y corregirlo.

2.3. Asistencia al desarrollo de Javadoc con IA.

Optamos por emplear inteligencia artificial para generar el javadoc de nuestra práctica por diversas razones fundamentales que han transformado nuestra experiencia:

- Ampliación de la comprensión del funcionamiento: Colaborar con la inteligencia artificial nos ha permitido comprender el funcionamiento de la herramienta y ser capaces de utilizarla de forma óptima.
- Agilización del proceso de trabajo: La IA automatizó la generación de comentarios de funciones, ahorrando tiempo y nos permitió centrarnos en tareas más con mas dificultad y que conllevarán más lógica

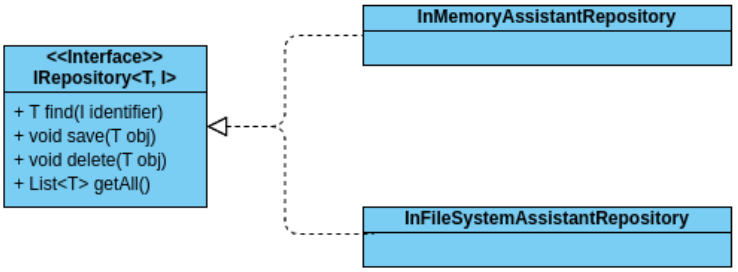
3. Decisiones de implementación.

3.1. Abstracción de métodos de manejo de fechas en la clase Utils.

Se centra en la conceptualización y extracción de los métodos relacionados con el manejo de fechas que se encuentran dentro de la clase denominada "Utils". En este apartado, se describen las funciones relacionadas con el manejo de fechas dentro de la clase Utils, incluyendo cómo estas funciones fueron diseñadas, implementadas y utilizadas en el contexto más amplio del proyecto o del software en desarrollo.

3.2. Implementación de repositorios.

El patrón Repository encapsula el acceso a los datos, desacoplando la lógica referente al dominio del problema del mundo real que estamos solucionando de la implementación concreta de persistencia.



Esta decisión nos ha ayudado en el desarrollo ya que en al principio solo tuvimos que desarrollar una implementación del repositorio en memoria con un diccionario, lo cual nos ayudó para poder probar los tests, ya que pudimos isolarlos los unos de los otros, sin la necesidad de escribir en un fichero cada vez que se ejecutaban los tests. Además, el cambio de persistencia en memoria a persistencia en ficheros o a base de datos solo nos supone implementar el repositorio concreto que necesitamos en cada momento.

3.3. Implementación de interfaces.

Optamos por interfaces en lugar de clases abstractas en el caso de las factorías y los repositorios con el objetivo de establecer un contrato universal para todas las clases que implementan esas interfaces, sin compartir ninguna lógica común. Si hubiera habido una lógica compartida, habríamos utilizado clases abstractas para heredar, pero nuestro enfoque se centraba solo en compartir las funciones públicas. Además, las interfaces ofrecen flexibilidad para dividirlas en un futuro ya que en java se permite implementar varias interfaces pero no la herencia múltiple.

3.3. Implementación de excepciones.

Con el objetivo de facilitar la tarea de los diferentes gestores de nuestro programa y el propio desarrollo del mismo, hemos decidido crear excepciones lo más concretas posibles las cuáles nos proporcionan una detección de errores e imprevistos más precisa.

4. Análisis de dificultades

- Comprensión del enunciado. Inicialmente, nos enfrentamos al desafío de comprender completamente el enunciado del proyecto y obtener los requisitos para elaborar el proyecto. A medida que avanzamos, surgieron nuevas restricciones que no habíamos anticipado en un principio. Esto nos llevó a reevaluar constantemente nuestra estrategia y adaptarnos a las nuevas circunstancias.

- Implementación del enunciado. La transición del enunciado del proyecto a código real resultó desafiante. La necesidad de comprender y cumplir con todos los requisitos antes de comenzar la implementación añadió una capa adicional de complejidad.

- Reestructuración del proyecto. Experimentamos con varias estructuras para nuestro proyecto antes de decidirnos por la versión actual, que está organizada y bien definida. Esta fase de reestructuración fue esencial para optimizar nuestro flujo de trabajo y asegurar que cada componente del proyecto estuviera en su lugar correcto.