Informe App 2

### **Arquitectura del Proyecto**

El proyecto está estructurado en paquetes según su responsabilidad, siguiendo los principios de separación de preocupaciones:

* app: Clase principal (App2.java) que contiene el punto de entrada.
* models: Clases que representan las entidades del dominio (por ejemplo, Cultivo, Parcela, Actividad, etc.).
* services: Contiene la lógica de negocio, como CultivoService y CSVService, responsables de gestionar la información.
* ui: Implementa el menú en consola y la interacción con el usuario (Menu.java).
* utils: Funciones auxiliares como CSVHandler para la lectura/escritura de archivos CSV.

#### **Herencia y Polimorfismo**

Se utilizó una clase abstracta ElementoAgricola como superclase de Cultivo, Parcela, y Actividad para reutilizar atributos comunes.

#### **Composición**

* Un Cultivo contiene una colección de Parcela.
* Cada Parcela puede tener múltiples Actividad. **Justificación del Uso de Colecciones y Patrones**

#### **Colecciones:** Se usaron principalmente ArrayList por su flexibilidad, facilidad de uso y eficiencia para listas dinámicas.

* List<Cultivo> para representar cultivos cargados desde el archivo.
* List<Actividad> para registrar actividades de cultivo.
* List<Parcela> como parte de la composición dentro del objeto Cultivo.

#### **Patrones de Diseño**

* **MVC Ligero**: Separación clara entre modelos (models), lógica de negocio (services) e interfaz (ui).
* **DAO simplificado**: CSVHandler simula el comportamiento de un acceso a datos simple desde un CSV, separando persistencia del resto del sistema.

### **Manejo de Modificadores de Acceso**

* **private** para atributos internos de clases (encapsulamiento).
* **public** solo para métodos que deben ser accedidos desde otros paquetes o clases.
* **protected** en métodos heredables por subclases, si corresponde.

El encapsulamiento se aplicó rigurosamente a través de getters y setters cuando fue necesario para exponer o modificar atributos.

### **Reflexiones Finales**

#### **¿Qué fue lo más desafiante de implementar en POO?**

La parte más compleja fue diseñar correctamente la jerarquía de clases y decidir qué lógica debía ir en los modelos, qué en los servicios, y qué en la interfaz. Aplicar principios como SRP (Single Responsibility Principle) ayudó a mantener el código modular.

#### **¿Cómo controlaron la lectura y escritura de CSV?**

Se creó una clase CSVHandler en utils para encapsular la lógica de lectura y escritura, utilizando BufferedReader y BufferedWriter. El archivo cultivos.csv se carga al inicio de la aplicación y se guarda automáticamente al salir.

#### **¿Qué aprendizajes surgieron del proyecto?**

* La importancia de organizar el código por capas.
* Cómo estructurar un proyecto Java real con paquetes, servicios y separación de lógica.
* Uso de Maven, JUnit 5 y control de versiones con Git.
* La utilidad del principio de responsabilidad única en el diseño de clases.

### **Uso de Inteligencia Artificial (IA)**

### **¿Qué tipo de ayuda proporcionó la herramienta?**

Se utilizó IA (como **ChatGPT** y **GitHub Copilot**) para asistir en distintas etapas del desarrollo:

* **ChatGPT**:
  + Generación de plantillas base para clases Java, estructura de paquetes y separación por capas.
  + Propuestas de refactorización para mejorar cohesión y cumplir con SRP.
  + Diagnóstico y solución de errores relacionados con compilación, Maven y ejecución en terminal (Git Bash, MinGW64, etc.).
* **GitHub Copilot**:
  + Autocompletado de métodos simples (constructores, getters/setters).
  + Sugerencias automáticas de bucles, condiciones y estructuras comunes durante la escritura de código.
  + Apoyo en la creación de pruebas unitarias y métodos repetitivos.

#### **¿Cómo validaron o contrastaron las sugerencias?**

* Se revisaron manualmente todas las sugerencias y se modificaron según el contexto del dominio de la aplicación.
* Se probaron las clases individualmente tras cada cambio.
* Se utilizó **JUnit 5** para asegurar que las funciones críticas se comportaran como se esperaba.
* Se priorizó la comprensión del código por sobre la generación automática, descartando sugerencias que no encajaran con el diseño establecido.