Informe de diseño App2

Gestión de cultivos, parcelas y actividades

04/05/2025

Fecha: 04/05/2025

Felipe Álvarez Mercado

Jan Michaelsen Thiel

Lenguajes y paradigmas de la programación | Sección 2

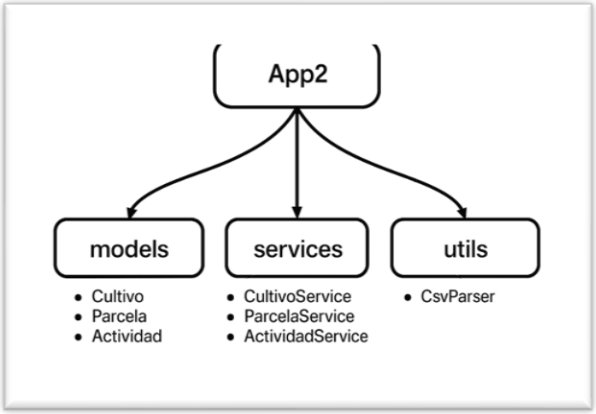
Profesor: Justo Vargas

Introducción

El presente documento tiene como finalidad registrar el diseño y desarrollo de una aplicación de consola en Java enfocada en la gestión agrícola, particularmente en el registro de cultivos, parcelas y las actividades de mantenimiento relacionadas. Este sistema fue creado con el objetivo de aplicar los principios fundamentales de la Programación Orientada a Objetos (POO), así como de fortalecer habilidades vinculadas al manejo de archivos CSV, la organización modular del código y el uso de herramientas de desarrollo como Git y GitHub. Durante el proceso de desarrollo, se enfrentaron diversos retos de diseño, como la estructuración del sistema en paquetes coherentes, la implementación de relaciones de herencia y composición entre clases, el adecuado encapsulamiento de los atributos y la persistencia de los datos. Asimismo, se abordaron buenas prácticas de codificación para mejorar la mantenibilidad, extensibilidad y claridad del código fuente. Este informe presenta la arquitectura general de la aplicación, detalla las decisiones de modelado orientado a objetos, justifica el uso de colecciones y patrones de diseño aplicados, y ofrece una reflexión final sobre los aprendizajes obtenidos a lo largo del proyecto.

Arquitectura general

La aplicación está organizada según un diseño modular dividido en paquetes, lo que posibilita una clara delimitación de responsabilidades y mejora la mantenibilidad del código. El proyecto adopta una arquitectura simple de tipo modelo-servicio-controlador (MVC), ajustada al contexto de una aplicación de consola. A continuación, se presenta una breve descripción de la función de cada paquete:



- **App:** Contiene la clase principal *App2,* que actúa como controlador de la aplicación. Aquí se encuentra el método *main,* desde el cual se despliega el menú de opciones y se orquesta la interacción entre el usuario y los distintos servicios.
- **Models:** Incluye las clases que representan las entidades principales del proyecto: *Cultivo, Parcela, Actividad* y la interfaz *ElementoAgricola.* Estas clases encapsulan los datos y comportamientos básicos relacionados con el sistema.
- **Services:** Define la lógica de negocio y las operaciones que afectan directamente a los modelos. Incluye servicios como *CultivoService, ParcelaService, Actividad Service y CsvService,* donde este último está encargado de la persistencia de datos en el archivo CSV.
- **Utils:** Contiene utilidades auxiliares que no pertenecen directamente al modelo de negocio, como la clase CsvParser, encargada de transformar líneas de texto en objetos del sistema y viceversa.

Flujo de ejecución

El flujo de ejecución del sistema está diseñado para ser sencillo y directo, facilitando la interacción del usuario desde la consola y garantizando al mismo tiempo la integridad de los datos. Es importante considerar que, durante el desarrollo del proyecto se trabajó bajo el supuesto indicado en el enunciado, de que el usuario ingresaría siempre información válida (inputs válidos).

En un inicio, al ejecutar el programa mediante la instrucción “java app.App2 cultivos.csv”, se inicia la clase principal *App2* que espera recibir como argumento el nombre del archivo de datos. Este archivo, nombrado “cultivos.csv” en el enunciado, contiene la información persistida de los cultivos, parcelas y actividades. Si el archivo no se proporciona, el sistema muestra un mensaje de cómo debiera ser el uso correcto y finaliza su ejecución de forma segura. Luego de recibir el archivo correctamente, se invoca el método de lectura del servicio *CsvService,* el cual utiliza **BufferedReader** (clase nativa de Java) y expresiones regulares para leer línea por línea el contenido del archivo CSV. Cada línea se parsea y se convierte en un objeto de tipo *Cultivo,* utilizando la clase *CsvParsear*. Estos objetos se almacenan en una colección *List<Cultivo>* en memoria, la cual representa el estado actual del sistema. Teniendo los datos ya cargados, el sistema despliega un menú principal interactivo desde la consola, que permite acceder a distintas áreas funcionales: gestión de cultivos, parcelas, actividades y reportes. Cada opción del menú está implementada como un bloque de código que delega las operaciones a los servicios correspondientes:

- **CultivoService:** Gestiona el CRUD de cultivos y búsquedas por nombre o estado
- **ParcelaService:** Permite listar parcelas, crearlas, editarlas y asignar cultivos
- **ActividadService:** Administra actividades tales como riego, fertilización o cosecha, registrándolas y marcándolas como completadas.

Estas interacciones modifican directamente las listas en memoria, pero no se escriben de inmediato en el archivo CSV para evitar sobreescrituras constantes.

Cuando el usuario selecciona la opción “Salir” desde el menú principal, el sistema invoca nuevamente el servicio *CsvService,* esta vez para sobrescribir el archivo “cultivos.csv” con los datos actualizados. Para ello, se recorre la lista de objetos *Cultivo* y se invoca su método *toCsvLine(),* que convierte cada instancia en una línea compatible con el formato original del archivo, basándose en la estructura de ejemplo proporcionada en el enunciado. De esta forma, se garantiza que todos los cambios realizados durante la sesión sean persistidos correctamente.

De forma resumida:

1. Al ejecutarse el programa, se recibe como argumento el archivo “cultivos.csv”
2. El servicio *CsvService* lo procesa y carga los datos como una lista de objetos *Cultivo*
3. Los servicios especializados *CultivoService, ParcelaService y ActividadService* permiten gestionar estos datos mediante un menú interactivo en la consola
4. Al salir, la información modificada se persiste nuevamente en el mismo archivo CSV

Este flujo de ejecución asegura una carga eficiente, una experiencia de usuario fluida y una escritura segura de los datos. Además, al utilizar una estructura modular basada en servicios, el código permanece flexible y fácil de mantener.

Comunicación entre componentes

Una de las decisiones clave en el diseño de esta aplicación fue establecer una comunicación clara y desacoplada entre los distintos niveles del sistema. Este enfoque permite aislar responsabilidades, facilitar futuras modificaciones y mejorar la legibilidad general del código. Las principales relaciones entre componentes son las siguientes:

Controlador (*App2*)

La clase *App2,* ubicada en el paquete **app**, actúa como el punto de entrada del programa y es responsable de presentar el menú interactivo en la consola. Esta clase no contiene lógica de negocio, sino que se limita a:

- Mostrar las opciones del menú al usuario
- Capturar la entrada por teclado utilizando un objeto *Scanner*
- Delegar las operaciones específicas a los servicios correspondientes

Este enfoque sigue el principio de separación de responsabilidades, ya que *App2* se encarga únicamente de la interacción con el usuario, manteniéndose ajena a los detalles del modelo o de la persistencia de datos.

Servicios (*services*)

Los servicios son el intermediario de la aplicación y son los encargados de ejecutar la lógica de negocio. Cada servicio está asociado a un área funcional distinta:

- *CultivoService* administra la creación, edición, eliminación y búsqueda de cultivos.
- *ParcelaService* gestiona parcelas y sus relaciones con los cultivos.
- *ActividadService* registra y manipula actividades asociadas a los cultivos.
- *CsvService* se encarga de leer y escribir en el archivo CSV, que actúa como fuente de datos persistente.

Los servicios operan sobre colecciones de objetos del paquete *models.* Por ejemplo, *CultivoService* recibe una *List<Cultivo>* al ser instanciado y realiza operaciones sobre dicha lista, como agregar, filtrar o eliminar cultivos. De esta forma, los servicios actúan como una especie de “puente” entre los datos almacenados y las acciones solicitadas desde el controlador.

Modelo (*models*) e interfaz común *ElementoAgricola*

Las clases *Cultivo, Parcela* y *Actividad* representan las entidades del dominio y encapsulan los atributos y comportamientos relevantes a cada concepto agrícola. Todas ellas implementan la interfaz *ElementoAgricola,* que define tres métodos fundamentales:

```
public interface ElementoAgricola {
    String getNombre();
    LocalDate getFecha();
    String toCsvLine();
}
```

Este enfoque permite que distintas clases del modelo puedan ser manipuladas de forma genérica si así se desea. Además, garantiza que cada objeto pueda ser convertido a una representación en formato CSV mediante el método *toCsvLine().*

Persistencia (*CsvService)*

El componente *CsvService* se encarga de la lectura y escritura de datos en archivos CSV. Su funcionamiento depende de la interfaz *ElementoAgricola,* ya que al momento de guardar datos:

- Recorre una lista de objetos *ElementoAgricola*
- Invoca el método *toCsvLine()* en cada uno para obtener una línea de texto formateada
- Escribe esas líneas en el archivo CSV, sobrescribiendo el contenido anterior

Esto permite mantener el almacenamiento externo completamente desacoplado del resto de la lógica interna. Además, la simetría entre los métodos de lectura y escritura garantiza que los datos pueden ser serializados y de-serializados sin pérdida de información ni alteraciones en su estructura.

Reflexiones

Durante el desarrollo de la aplicación, nos enfrentamos a diversos desafíos relacionados con el diseño orientado a objetos, la persistencia de los datos y la organización general del código. Por otro lado, fue el primer proyecto implementado en Java por parte de los integrantes, por lo que también existieron dificultades y dudas que se fueron resolviendo al ir adentrándose más en el lenguaje.

¿Qué fue lo más desafiante de implementar POO?

Sin duda, uno de los mayores desafíos para el grupo fue estructurar correctamente las relaciones entre las clases y distribuir adecuadamente las responsabilidades en cada una. En particular, aplicar composición sin acoplar excesivamente las clases. Por ejemplo, fue necesario decidir si los objetos *Parcela* debían tener referencias directas a objetos *Cultivo,* o si era preferible usar solo los códigos de *Parcela* para mantener las entidades más independientes. Finalmente se optó por una estructura mas bien híbrida, en la que las parcelas mantienen una lista de cultivos asociada, pero los cultivos también conservan el código de la parcela a la que pertenecen, lo que permite búsquedas cruzadas sin crear dependencias complejas.

El diseñar una interfaz genérica permitió unificar ciertas operaciones como la exportación a CSV, pero requirió pensar cuidadosamente en qué métodos debían ser obligatorios para cada entidad del dominio y cómo garantizar que todos los modelos se implementaran de forma coherente.

Encapsular los datos correctamente también supuso dificultad, al inicio fue tentador trabajar con atributos públicos o accesibles directamente, pero una vez aplicado el principio de encapsulamiento, se prefirió definir todos los atributos como *private* y controlar el acceso mediante *getters* y *setters*, lo que mejoró la seguridad y claridad del sistema.

¿Cómo se controló la lectura y escritura del archivo CSV?

La persistencia de los datos fue manejada mediante la clase *CsvService,* diseñada específicamente para abstraer la lectura y escritura del archivo CSV. Algunas decisiones clave en este aspecto fueron:

- Lectura controlada línea por línea con *BufferedReader*, una clase nativa de Java que permite leer texto de forma eficiente desde archivos. Esto permitió trabajar con archivos de cualquier tamaño sin cargar todo el contenido de la memoria.
- Cada línea del archivo fue dividida mediante expresiones regulares, respetando la estructura del CSV original, que contenía campos entre comillas y listas anidadas representadas como arreglos JSON. Para transformar cada línea en un objeto *Cultivo,* se creó una clase *CsvParser,* que encapsula toda la lógica de de-serialización y facilita su mantenimiento.
- La escritura fue implementada recorriendo las listas de objetos y delegando en cada uno el formato de su línea correspondiente. El uso del método *toCsvLine()* en la interfaz *ElementoAgricola* garantizó una serialización consistente y reutilizable, incluso si en el futuro se tuvieran que integrar nuevas entidades al sistema.
- Se eligió que los datos se escribieran en el archivo solo al finalizar el programa, lo que evita errores por escrituras parciales en caso de interrupciones inesperadas. Esta estrategia también simplificó la lógica de control de errores, que de igual forma no se desarrolló de forma exhaustiva.

¿Qué aprendizajes surgieron del proyecto?

Este proyecto fue una excelente oportunidad para poner en práctica y consolidar diversos conceptos clave del desarrollo en Java y la programación orientada a objetos. Como grupo, cada vez más nos asombramos de la importancia del diseño modular. Al dividir el sistema en paquetes y servicios, no solo se mejora la organización del código, sino que también se facilita su comprensión y extensión futura. Claramente como estudiantes de informática, recién en el primer semestre de la especialidad, somos bastante inexpertos en la programación. Los trabajos relacionados a la programación que realizamos previamente para otros ramos en general requerían desarrollar un código en un único archivo. El descubrimiento del diseño modular como también la herramienta de manejo de versiones GitHub nos abrió los ojos respecto a el abanico de posibilidades del mundo informático.

También, en un foco más técnico, aprendimos la gran utilidad de algunas herramientas nativas de Java. A lo largo del desarrollo del trabajo, se utilizaron clases estándar como *BufferedReader, ArrayList, HashMap y LocalDate,* lo que permitió aprovechar al máximo las capacidades del lenguaje sin depender de bibliotecas externas.

¿Cuál fue el aporte de la inteligencia artificial?

La inteligencia artificial ha significado una enorme ayuda para el avance del proyecto. En un principio, al leer el enunciado, como grupo nos vimos un poco perdidos sobre dónde y cómo empezar. La IA significó un aporte fundamental en la discusión previa a la programación de la aplicación como tal. Ayudó a definir la estructura de software, como también sugiriéndonos una metodología a seguir, proporcionando un estilo de “timeline” sobre en qué orden atacar los desafíos del problema.

Asimismo, la IA resultó ser crucial para el código como tal del proyecto, considerando que como grupo nos vimos un tanto agobiados por la combinación del breve tiempo de desarrollo y la cantidad de evaluaciones que se acumularon la presente semana. Por esto, la IA complementó de gran manera aportando principalmente algunas sugerencias sobre cómo afrontar ciertos problemas con el código, resolviendo errores donde no sabíamos donde se originaban y, por último, fue de inmensa ayuda con todo el código relacionado a la lectura del archivo CSV. Experimentamos una gran cantidad de errores porque no lográbamos parsear el archivo de manera correcta, generando que básicamente el resto del programa no

funcione ya que depende fuertemente de los datos del archivo CSV. La inteligencia artificial nos ayudó enormemente a implementar el método de parseo como también de escritura en el CSV.