# Informe final

Este informe habla de los elementos y decisiones más representativas de la aplicación desarrollada, el mismo es simplemente un guía las cual permitirá demostrar las competencias y conceptos vistos en la asignatura de Orientación a Objetos 2 durante el examen final. Estos conceptos son principalmente cuatro, de los cuales se hablará sobre:

- **Framework:** propósito del framework y tipo, partes principales.
- **TDD:** herramientas utilizadas y su efecto en el desarrollo del proyecto. Decisiones importantes al crear los test de unidad.
- **Patrones de diseño:** patrones de diseño utilizados y justificando de su uso en el framework.
- **Refactoring y code smells:** detección de code smells, aplicación de refactorings y cambios en el diseño.

## Framework

El framework tiene como fin permitir a las personas que lo instancian crear aplicaciones que permitan determinar qué cultivos se pueden o no plantar en los distintos tipos de espacio de su quinta teniendo en cuenta distintos factores como el tipo de suelo y la época del año y llevar un control de los cultivos que se encuentran en sus distintos espacios.

En primer lugar hablemos de los hotspots, las partes que lxs programdorxs van a poder personalizar, estos son dos principales:

- La creación de los tipos de suelo.
- La configuración de la quinta con estos tipos de suelo.

En cuanto a los frozen spots importantes de la aplicación podemos hablar donde se da el loop de control el cual arranca una vez se instala y corre la aplicación. Este loop imprime en pantallas las opciones posibles a tomar y espera a recibir el input de lxs usuarixs. La más importante es la opción de agregar cultivo que es donde da la inversión de control.

## TDD

Para la automatización de los test de unidad se utilizó el framework Junit el cual junto con la ayuda de las IDE nos permitió realizar TDD con mayor facilidad.

El proyecto fue desarrollado en partes usando esta dinámica y otra parte realizando primero el código. Este fue el caso de las clases relacionadas al tipo de suelo y parcialmente las clases de estaciones del año.

## Patrones de Diseño

En la aplicación podemos encontrar 4 patrones de diseño, ellos son:

- **Strategy:** para el modelado de las distintas estrategias de tipo de suelo.
- **State:** para el modelado de estaciones del año el cual es un estado de la quinta.
- **Singleton:** para controlar la cantidad de estaciones que existen.
- **Template method:** para declarar un esqueleto que controla en la estación como se hace el cálculo de saber si el cultivo puede plantarse o no.

Esta decisión de forzar patrones antes de tener código fue tomada debido a que debía presentarse un diseño inicial que presentará que mostrará en el dominio donde pueden aplicarse estos conceptos, si bien tenían sentido trajo con sigo algunos problemas que se discutirán en más profundidad en la parte de bad smells y refactoring.

## Refactoring y Code Smells

Partiendo de que construir el sistema perfecto es imposible, nosotrxs no fuimos la excepción. En el camino al desarrollo del framework nos encontramos con varios errores o cambios que nos vimos forzadxs a realizar para adaptar el problema a lo que teníamos planeado.

Algunos de los “malos olores” con los que nos encontramos fueron:

- Clase Dios: Quinta antes una superclase que ahora Quinta y Menú.
- Romper encapsulamiento: Todas las clases en principio fueron creadas con todos sus getter y setters por no saber cuales iban a resultar necesarias al final.
- Envidia Atributos: La clase Quinta en el método agregarCultivoAEspacio realizaba tareas que no le correspondía a su clase.
- Nombres de variables/métodos no representativos: Muchos nombres, tanto de métodos como de variables en un principio tenían nombres que no representaban su funcionamiento.
- Métodos largos: En varios lugares usamos ***extract methods*** y también ***replace temp with query*** (terminoLaEstacion()).
- Muchos parámetros: En este caso solo los constructores de estaciones, se podría realizar **Introduce Parameter Object** para que sea más descriptivo el constructor y no tener tantos parámetros. Además en el template method se mandaba un parámetro como resultado de una query que en realidad podría llamarse en ese mismo método así que se decidió reemplazar el parámetro con la llamada al método.
- Data class: la única clase que puede ser considerada dataclass es cultivo pero en este caso no existen funcionalidades que puedan serle agregadas.
- Generalidad especulativa: como ya se habló muchas veces, el diseño elegido hizo que hubiera clases que no hacían casi nada. Además se agregaron varios parámetros y métodos porque se quería agregar un montón de funcionalidades más a la aplicación pero por problema de tiempo se tuvieron que sacar.

Algunos malos olores que nos pareció importante resaltar por que no los encontramos:

- Código duplicado: En este caso antes de que sucediera este mal olor se pusieron las variables y los métodos en el lugar correspondiente. Este es el caso de la variable de instancia de EstacionDelAnio que tiene la lista de meses correspondiente a cada estación y el método terminoLaEstacion() que chequea si el mes actual está dentro de esa lista.
- Comentarios: no hay comentarios en el código, trato de usar los nombre más descriptivos tanto para variables como para métodos y clases.
- Sentencias switch: no llegamos a tener debido al diseño que aplicamos pero en caso de no haber aplicado Strategy o State seguramente habríamos tenido este problema.

Una vez hechos los cambios nos encontramos con el beneficio de la técnica TDD. Solo bastaba con volver a correr los test y que todo siga funcionando como lo venía haciendo.