ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

DISEÑO DE SOFTWARE

TALLER DE PATRONES DE DISEÑO CREACIONALES Y ESTRUCTURALES

Integrantes:

- Joey Bustamante

- Christian Olmedo

Herramienta de modelado: PlantUML

Fecha de entrega: 20/06/2025

Sección A:

1. Builder

Motivación:

Se utiliza para construir objetos complejos, como informes, de manera progresiva y controlada. Este patrón permite separar el proceso de construcción del objeto de su representación final, lo que lo hace ideal para generar informes en diferentes formatos.

Consecuencias:

✔ Aísla el proceso de creación de la representación del informe.

✔ Facilita la incorporación de nuevos formatos sin necesidad de modificar el código existente, respetando el Principio Abierto/Cerrado.

✘ Puede incrementar la cantidad de clases en el sistema.

1. Singleton

Motivación:

Garantiza que exista una única instancia del generador de reportes en toda la aplicación, tal como lo exige el enunciado. Esta instancia se comparte entre los distintos módulos que la requieren.

Consecuencias:

✔ Proporciona un punto de acceso global.

✔ Reduce el consumo de recursos al evitar instancias innecesarias.

✘ Si no se implementa con cuidado, puede dificultar la inyección de dependencias.

✘ Complica la realización de pruebas unitarias si no se desacopla adecuadamente.

1. Decorator

Motivación:

Permite añadir dinámicamente características visuales a los informes (como fuentes, colores y estilos) sin modificar la estructura básica de los mismos.

Consecuencias:

✔ Proporciona gran flexibilidad para personalizar los informes.

✔ Cumple con el Principio Abierto/Cerrado al permitir la extensión sin modificar clases existentes.

✘ Puede conducir a una proliferación de clases pequeñas si se combinan muchos estilos distintos.

1. Factory Method / Abstract Factory

Motivación:

Facilita la creación de objetos de tipo reporte sin depender de clases concretas, promoviendo el desacoplamiento y permitiendo variar el tipo de reporte a generar.

Consecuencias:

✔ Permite una fácil incorporación de nuevos tipos de informes.

✔ Elimina dependencias directas con clases específicas.

✘ Su combinación con Builder puede hacer más compleja la estructura general del sistema.

1. Adapter

Motivación:

Hace posible la integración con servicios externos como WhatsApp o Telegram, cuyas interfaces difieren del sistema actual de notificaciones por correo electrónico, sin necesidad de modificar el código base.

Consecuencias:

✔ Permite reutilizar el sistema existente sin cambios drásticos.

✔ Aísla la lógica de integración con servicios externos.

✘ Si su diseño no es cuidadoso, puede enmascarar incompatibilidades funcionales.

1. Bridge (opcional)

Motivación:

Permite desacoplar la abstracción del informe (su contenido) del medio a través del cual se entrega (correo, WhatsApp, Telegram), permitiendo variar ambos de forma independiente.

Consecuencias:

✔ Favorece la extensibilidad tanto en la variedad de informes como en los canales de distribución.

✘ Puede añadir complejidad innecesaria si no existen suficientes elementos variables por separado.

Supuestos:

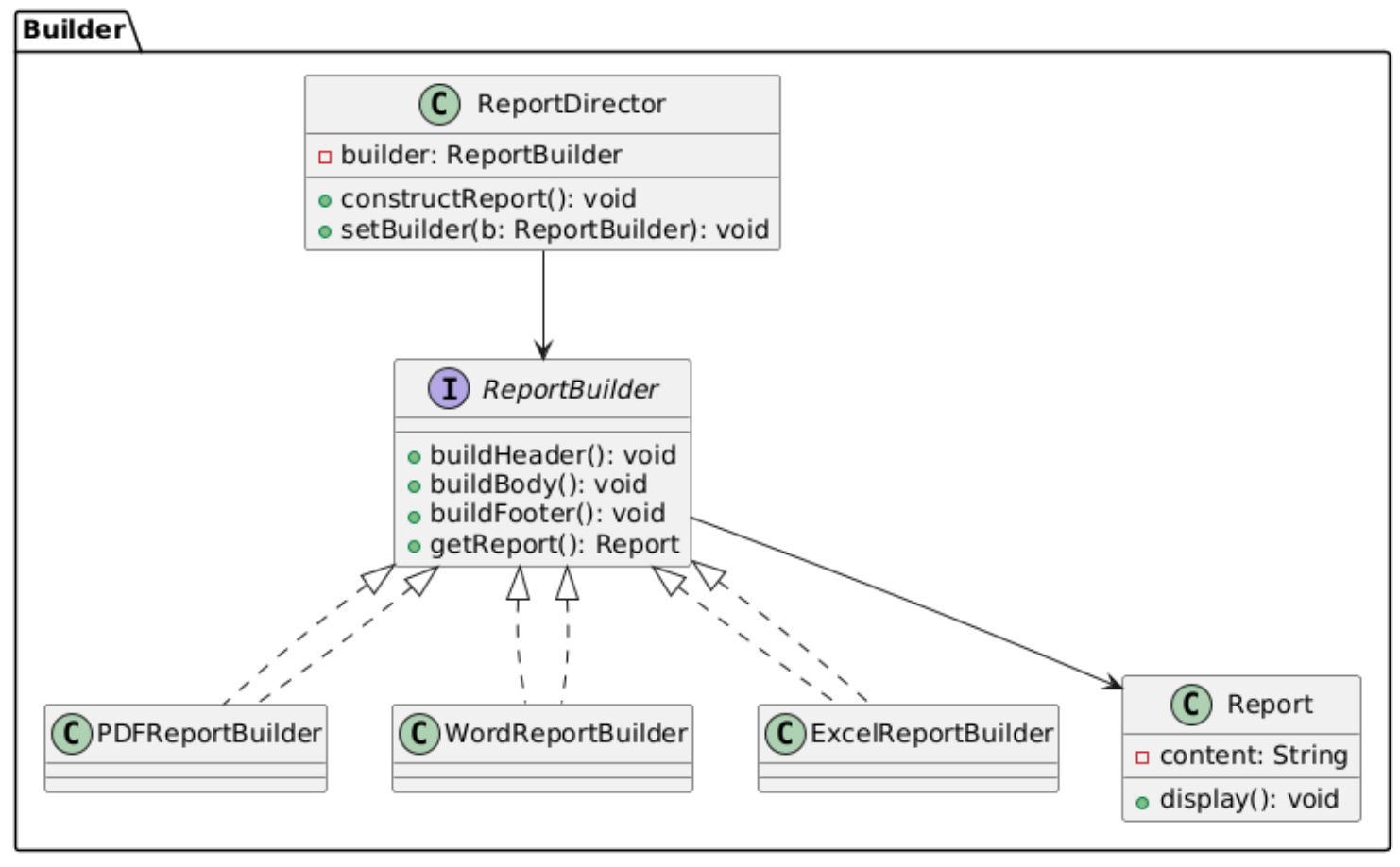
La responsabilidad de mostrar el informe recae en el propio objeto de informe o su constructor (builder).

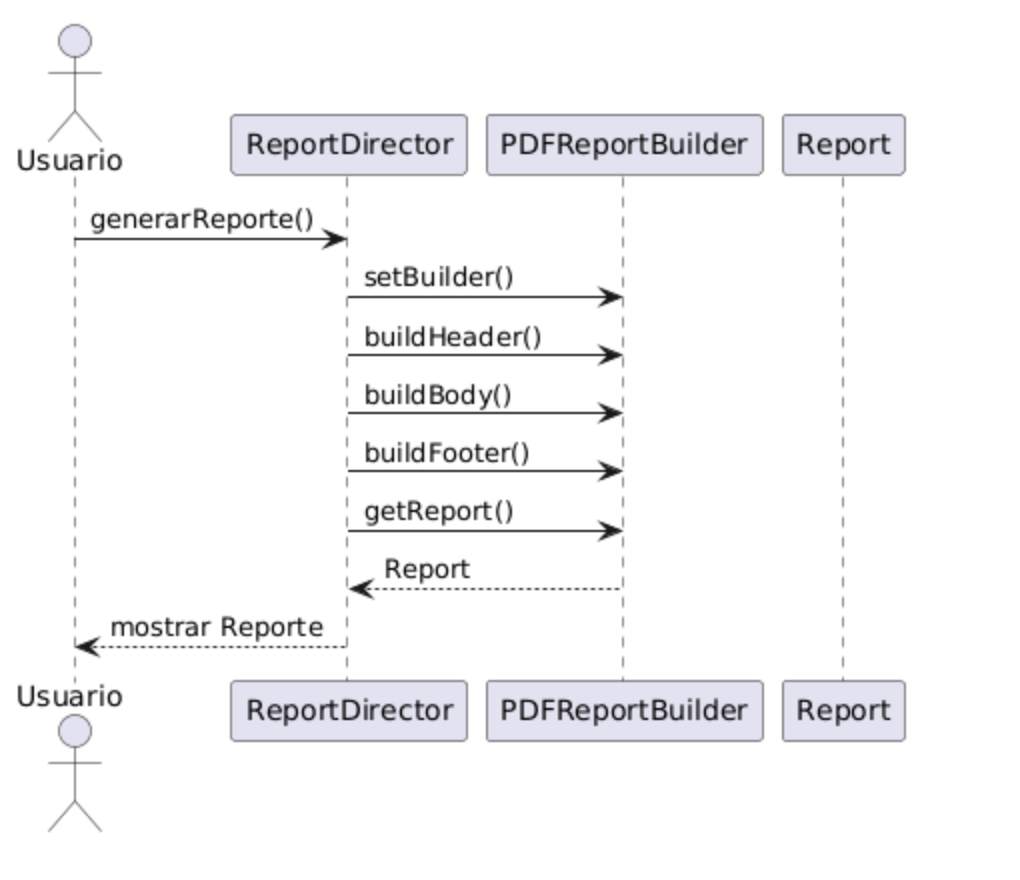
Se asume la existencia de interfaces comunes IReporte e INotificador.

Los elementos visuales se añaden mediante decoradores, y no forman parte del proceso de construcción.

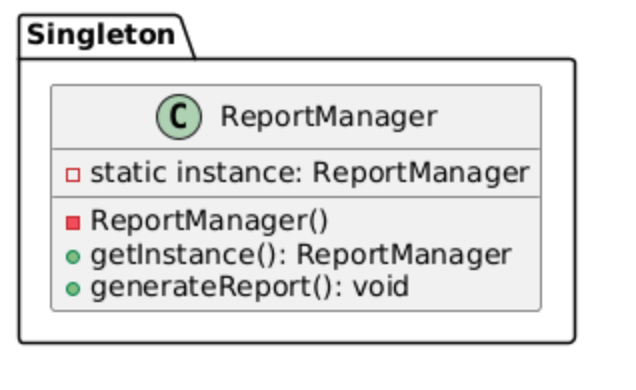
Diagramas de clase y Secuencias. (Se Uso PlantUML)

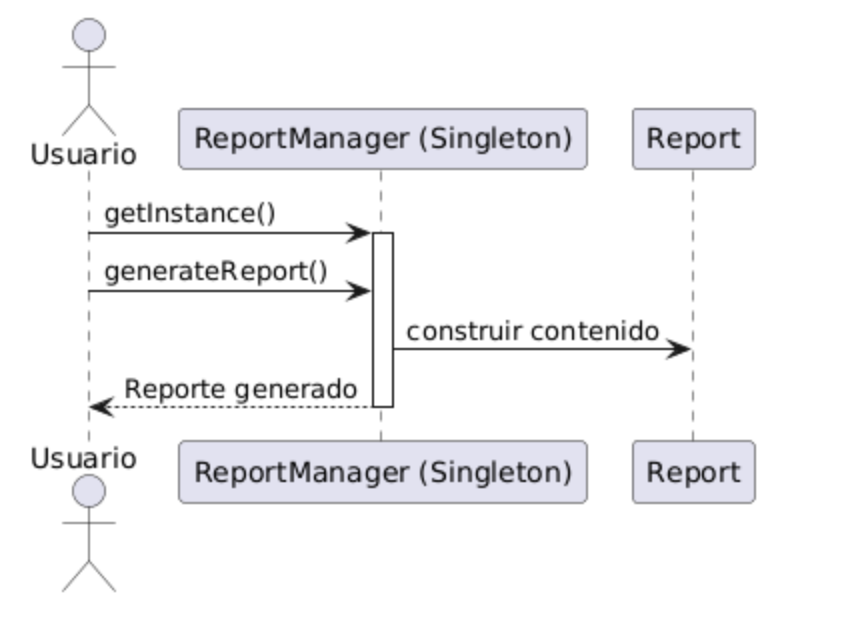
1. Builder



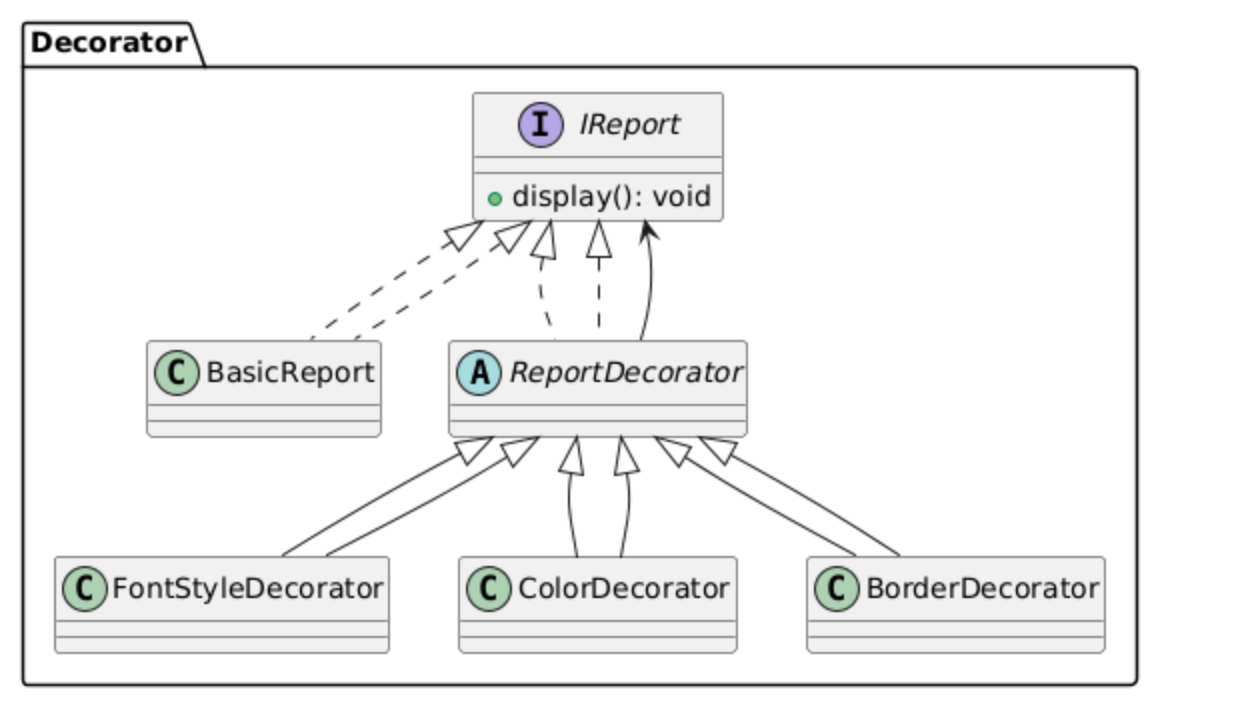


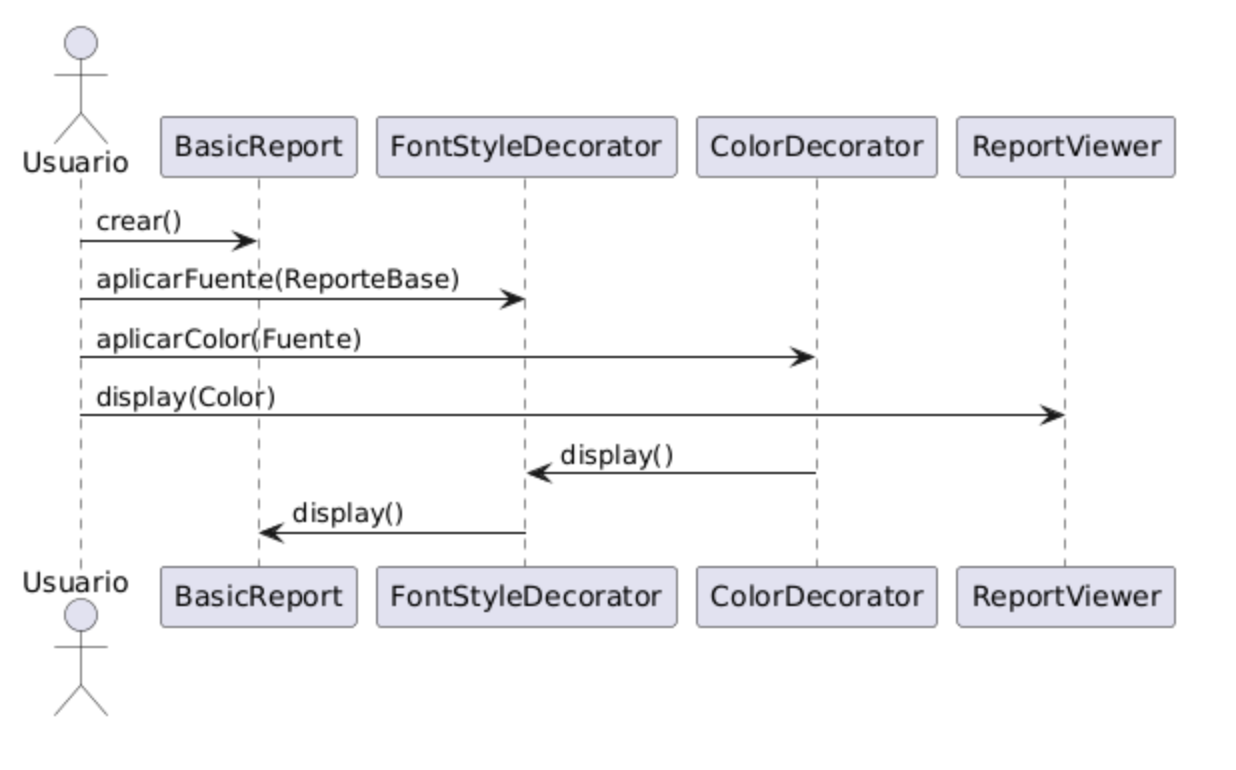
1. Singelton



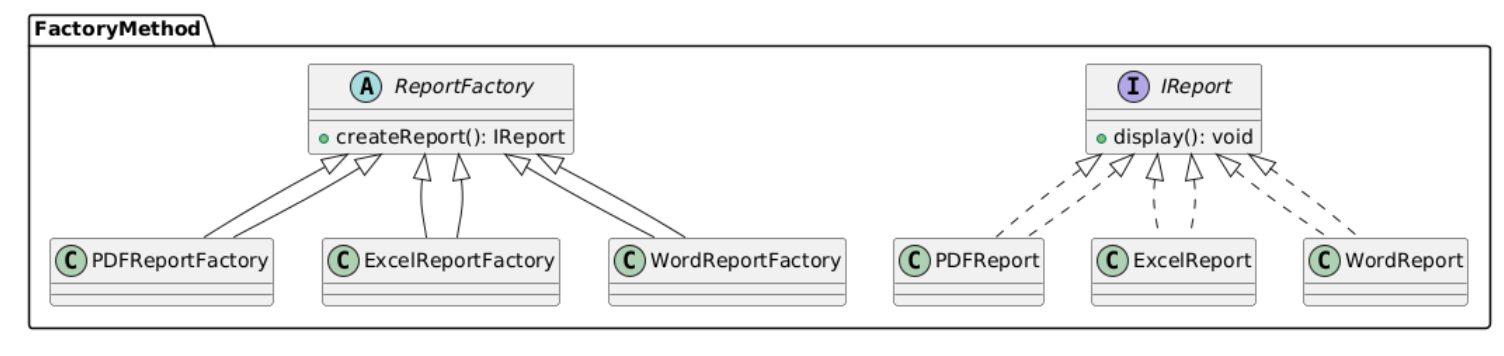


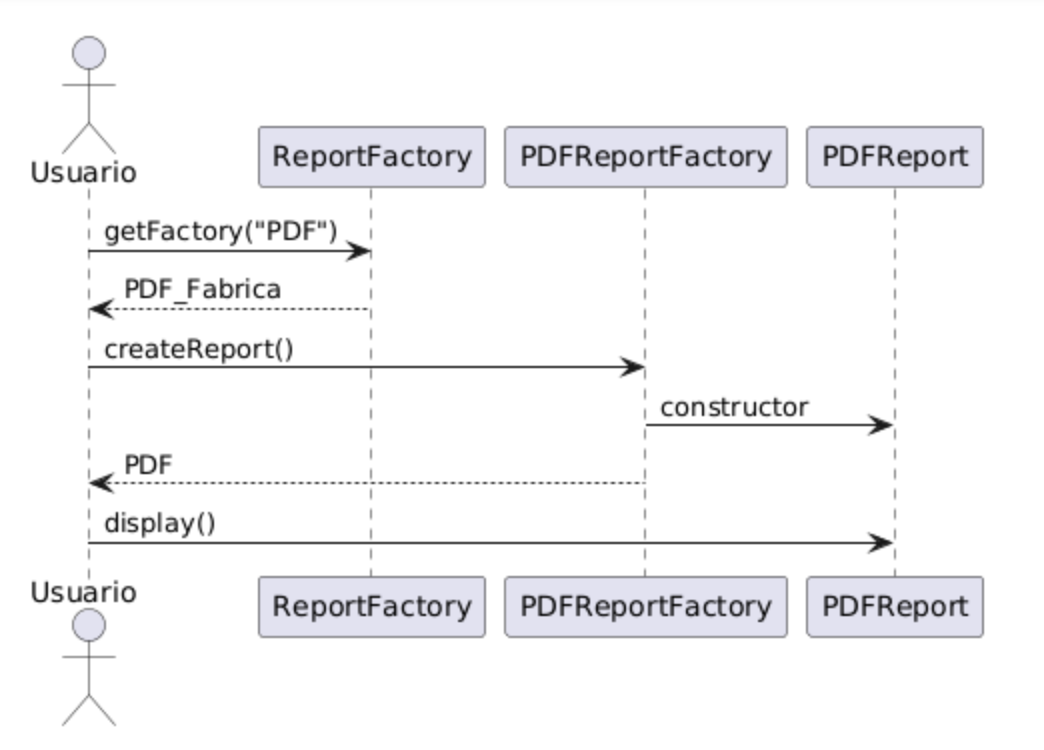
1. Decorator





1. Factory Method





1. Adapter

