# Reporte de patrones creacionales y estructurales

Diseño de software

**Especificaciones**

Imagina una empresa de gestión de proyectos se encuentra en la necesidad de desarrollar un nuevo sistema para generar y visualizar reportes de manera eficiente.

Este sistema debe ser capaz de generar informes en diversos formatos, como archivos PDF, hojas de cálculo Excel y documentos de texto en formato Word, todos los reportes independientes del formato se crean siguiendo los mismos pasos. Además, es crucial que el sistema sea flexible y escalable, de manera que pueda adaptarse fácilmente a la generación de informes en nuevos formatos que puedan surgir en el futuro. Tenga en cuenta que no tiene sentido tener varias instancias para generar y visualizar los reportes.

La empresa desea ofrecer informes atractivos visualmente, por lo que es necesario que cada informe tenga la capacidad de poder agregar opciones de personalización como opciones de estilo, fuentes, colores u otros elementos visuales a los informes dependiendo de lo que se requiera el usuario en ese momento, por ejemplo, algunos informes podrían requerir un diseño solo con cambios de fuente y estilos, mientras que otros podrían necesitar un enfoque más colorido.

Por último, el sistema tiene un módulo de notificaciones por correo electrónico, sin embargo, se requiere la integración con otros servicios externos de notificaciones como mensajería instantánea de WhastApp o de Telegram, pero ambas tienen distintas interfaces de uso y también difieren del módulo de notificaciones por email. Por lo tanto, se debe encontrar una forma para que el sistema pueda enviar los informes en cualquier formato, independientemente del modo de notificación que se escoja.

**Sección A**

Elabore un **reporte** en el que **identifique los patrones de diseño** aplicables para el sistema de software

descrito. Para cada patrón que considere pertinente, debe **indicar** la **motivación** de su uso y las

**consecuencias** de su **decisión (ej.: pros, contras, relación con principios SOLID)**. Indique cualquier asunción

que realice.

## Patrones creacionales

### Patrón Factory Method

**Motivación:** Se usa para crear diferentes tipos de informes (PDF, Excel, Word) sin tener que especificar las clases concretas. Esto permite que el código cliente se desacople de las clases concretas que se instancian. Además, se puede añadir la capacidad de crear nuevos tipos de informes en el futuro sin cambiar el código existente, haciendo que el sistema sea escalable.

**Consecuencias:** El patrón Factory Method facilita la adición de nuevos tipos de informes, pero puede aumentar la complejidad si hay muchas variantes de informes. Este patrón está alineado con el principio de inversión de dependencias de SOLID, ya que permite que el código cliente dependa de abstracciones, no de clases concretas.

### Patrón Singleton

**Motivación:** A pesar de tener varios tipos de fábricas es necesario tener un ReportManager que nos permita crear los reportes, pero a través de una única instancia de cada tipo de fábrica.

**Consecuencias:** impide la concurrencia al haber solo una instancia de cada fábrica, pero aumenta el acoplamiento de las clases al tener que modificar el código ReportManager si necesito aumentar un nuevo tipo de reporte. El principio SOLID que se violaría sería el OCP (Open/close principle).

## Patrones estructurales

### Patrón Decorator

**Motivación:** Se usa para añadir opciones de personalización a los informes de manera dinámica. Esto permite modificar el aspecto de los informes en tiempo de ejecución según las necesidades del usuario.

**Consecuencias:** Proporciona una mayor flexibilidad para personalizar los informes, pero puede aumentar la complejidad si hay muchas opciones de personalización. Alineado con el principio de responsabilidad única de SOLID.

### Patrón Adapter

**Motivación:** Se utiliza para permitir que la aplicación funcione con servicios de notificación externos que tienen interfaces diferentes. Esto permite enviar notificaciones a través de diferentes canales sin cambiar el código existente.

**Consecuencias:** Facilita la integración con servicios externos, pero puede aumentar la complejidad si hay muchos servicios de notificación. Alineado con el principio de inversión de dependencias de SOLID.

### Asunciones

**1.** **Múltiples formatos de informes:** Se asume que la aplicación necesita generar informes en varios formatos (PDF, Excel, Word), y que cada formato de informe requiere una implementación diferente.

**2. Personalización de informes:** Se asume que los informes necesitan ser personalizables. Esta personalización puede incluir la configuración de diferentes estilos de fuente y colores. Además, se asume que la personalización puede variar de un informe a otro y puede cambiar en tiempo de ejecución.

**3. Envío de notificaciones:** Se asume que la aplicación necesita enviar notificaciones a través de varios canales, y que cada canal de notificación tiene una interfaz diferente.

**4. Un único gestor de informes:** Se asume que solo se necesita una instancia del gestor de informes en la aplicación. Esto puede ser debido a que la generación y visualización de informes comparten algún estado común o son operaciones costosas en términos de recursos.

**5. Extensibilidad y mantenibilidad:** Se asume que es importante que el diseño de la aplicación permita la extensión (por ejemplo, añadir nuevos formatos de informes o canales de notificación) sin requerir modificaciones importantes en el código existente. Esto está en línea con el principio de inversión de dependencias y el principio abierto/cerrado de SOLID.

Por último, se asume que el uso de patrones de diseño como Factory method, Decorator, Adapter y Singleton puede ayudar a satisfacer los requisitos y restricciones de este problema. Sin embargo, es importante tener en cuenta que la elección de los patrones de diseño debe basarse en las necesidades específicas del problema y que no todos los patrones de diseño son adecuados para todas las situaciones.