**Diseño de Software – PAO I 2025**

**Taller 06 - Patterns**

**Grupo 7**

**Integrantes:**

* CAJAS TOAPANTA JAHIR MANUEL – [jmcajas@espol.edu.ec](mailto:jmcajas@espol.edu.ec)
* MIÑACA QUIROZ JUAN SALVADOR – [jminaca@espol.edu.ec](mailto:jminaca@espol.edu.ec)
* NARANJO MUÑOZ PABLO DAVID - [pdnaranj@espol.edu.ec](mailto:pdnaranj@espol.edu.ec)
* VALVERDE MATIAS VICTOR MANUEL - [vmvalver@espol.edu.ec](mailto:vmvalver@espol.edu.ec)

[1. SECCIÓN A: Identificación y justificación de Patrones: 3](#_Toc1254165523)

[1.1. Factory Method: 3](#_Toc695125730)

[1.2. Singleton 3](#_Toc2114167324)

[1.3. Decorator 3](#_Toc458155616)

[1.4. Adapter 4](#_Toc1828730340)

[1.5. Bridge 4](#_Toc621809562)

[2. SECCIÓN B: Diseño de Diagramas UMLEnlace del proyecto a LucidChart: https://lucid.app/lucidchart/64017043-5d87-46bd-af53-c1afbd3087d7/edit?viewport\_loc=451%2C776%2C1957%2C878%2CHWEp-vi-RSFO&invitationId=inv\_4c642eb6-77ab-4607-89f4-a3b8f9bd4879 5](#_Toc1605445372)

[2.1. Patrón Factory Method en el sistema 6](#_Toc1490956122)

[2.2. Patrón Singleton en el sistema 6](#_Toc1348721931)

[2.3. Patrón Decorator en el sistema 7](#_Toc1294218519)

[2.4. Patron Adapter en el sistema 8](#_Toc870882245)

[2.5. Patrón Bridge en el sistema 9](#_Toc2071061223)

[3. SECCIÓN C: Implementación en JAVA 10](#_Toc869976526)

# **SECCIÓN A: Identificación y justificación de Patrones:**

## **1.1. Factory Method:**

**Motivación:** Permitir la **creación** de informes en diferentes formatos PDF, Excel, Word, encapsulándolos a la hora de usar la lógica de instanciación.

**Consecuencias a favor:** Ayuda a la extensión a nuevos formatos de informe y sigue con el principio Open/Closed.

**Consecuencias en contra:** Añade complejidad con múltiples subclases de fábrica.

Relación con SOLID: Apoya OCP (Open/Closed Principle) y SRP (Single Responsibility Principle).

**Asunción**: Se asume que cada tipo de reporte (PDF, Excel, Word) debe crearse a través de una interfaz común para permitir la extensión sin modificar el código existente.

## **1.2. Singleton**

**Motivación:** Garantizar que exista una única instancia para la generación de informes, tal como se especifica en el enunciado.

**Consecuencias a favor:** Control de acceso a una única instancia; consumo controlado de recursos.

**Consecuencias en contra:** Dificulta pruebas unitarias y la paralelización si no se maneja bien.

Relación con SOLID: Puede violar SRP si se abusa, pero en este caso es válido.

**Asunción**: Debe existir una única instancia global del generador de reportes para garantizar consistencia en todo el sistema.

## **1.3. Decorator**

**Motivación:** Agregar de forma dinámica opciones de personalización visual a los informes (fuentes, colores, estilos).

**Consecuencias a favor:** Flexibilidad al combinar decoraciones; respeta OCP.

**Consecuencias en contra:** Complejidad en la gestión de combinaciones múltiples.

Relación con SOLID: Apoya OCP y SRP.

**Asunción:** Se asume que los estilos visuales de los reportes deben aplicarse de forma flexible y combinable sin alterar las clases base.

## **1.4. Adapter**

**Motivación:** Integrar servicios de notificación externos con interfaces distintas (Email, WhatsApp, Telegram).

**Consecuencias a favor:** Permite reutilizar código de APIs externas; desacopla el sistema de las interfaces concretas.

**Consecuencias en contra:** Incrementa el número de clases adicionales (adapters).

Relación con SOLID: Refuerza DIP (Dependency Inversion Principle) y OCP.

**Asunción:** Se asume que las APIs externas de notificación tienen interfaces incompatibles que deben adaptarse al sistema actual sin modificarlas.

## **1.5. Bridge**

**Motivación:** Separar la lógica de generación de reportes del canal de envío (Email, WhatsApp, Telegram).

**Consecuencias a favor:** Se puede modificar el modo de envío o el contenido sin afectar al otro.

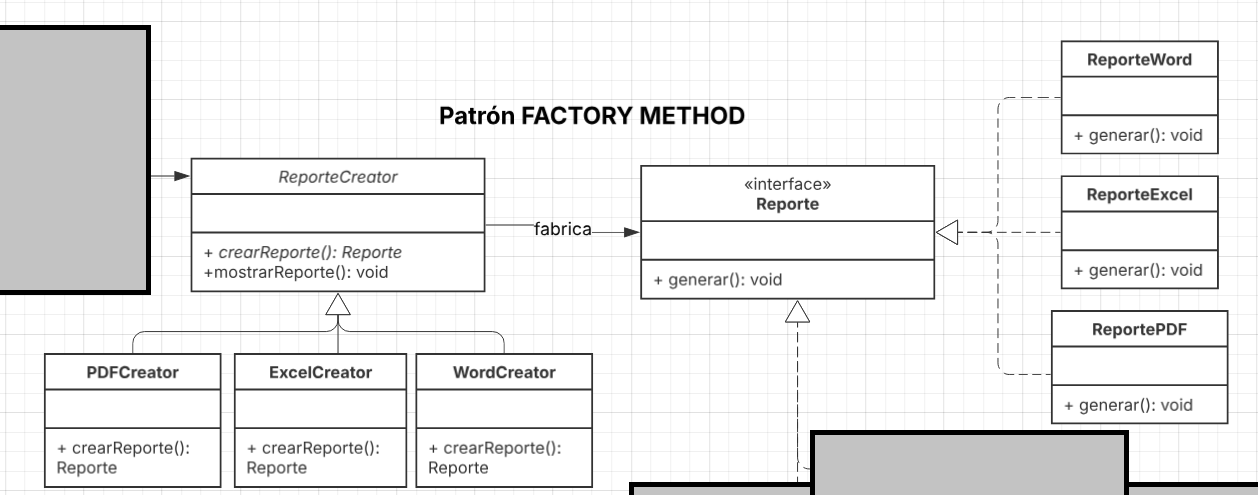
**Consecuencias en contra:** Añade complejidad inicial al diseño.

Relación con SOLID: Favorece el principio de inversión de dependencias.

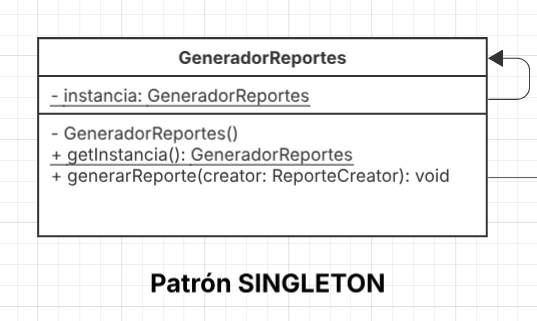
**Asunción:** Se asume que la lógica de generación de reportes y los canales de envío deben variar de forma independiente.

# **SECCIÓN B: Diseño de Diagramas UML** **Enlace del proyecto a LucidChart:** <https://lucid.app/lucidchart/64017043-5d87-46bd-af53-c1afbd3087d7/edit?viewport_loc=451%2C776%2C1957%2C878%2CHWEp-vi-RSFO&invitationId=inv_4c642eb6-77ab-4607-89f4-a3b8f9bd4879>

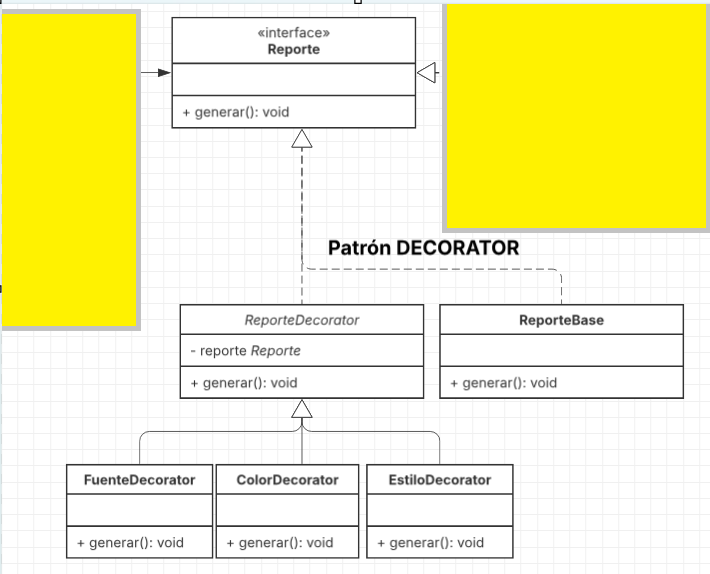
## **2.1. Patrón Factory Method en el sistema**



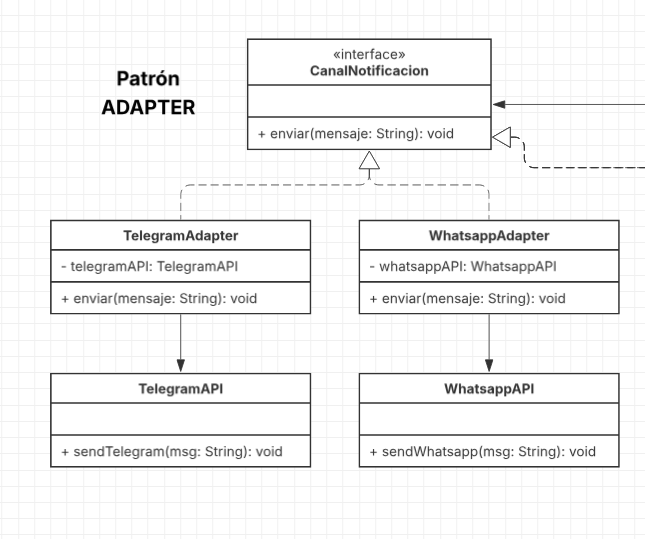
## **2.2. Patrón Singleton en el sistema**



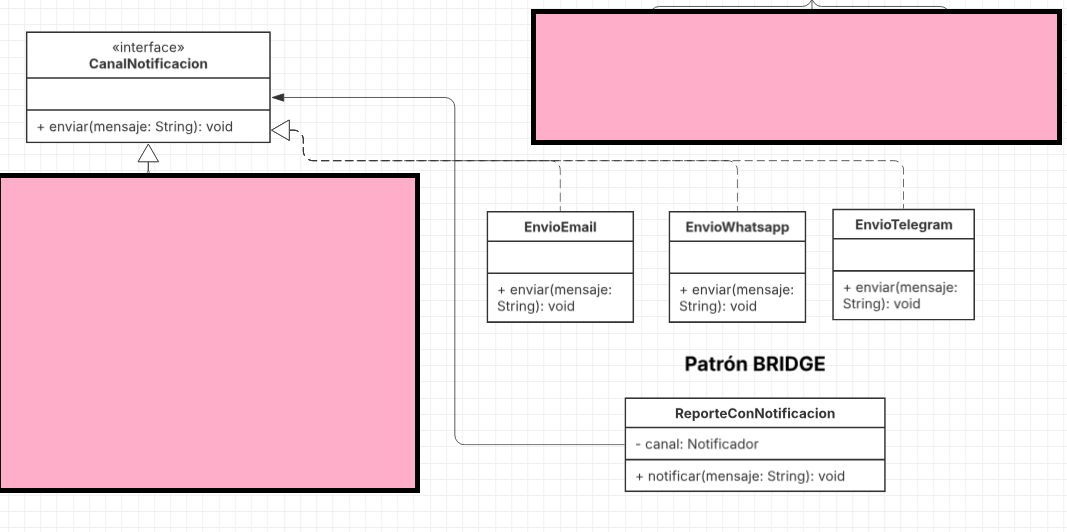
## **2.3. Patrón Decorator en el sistema**



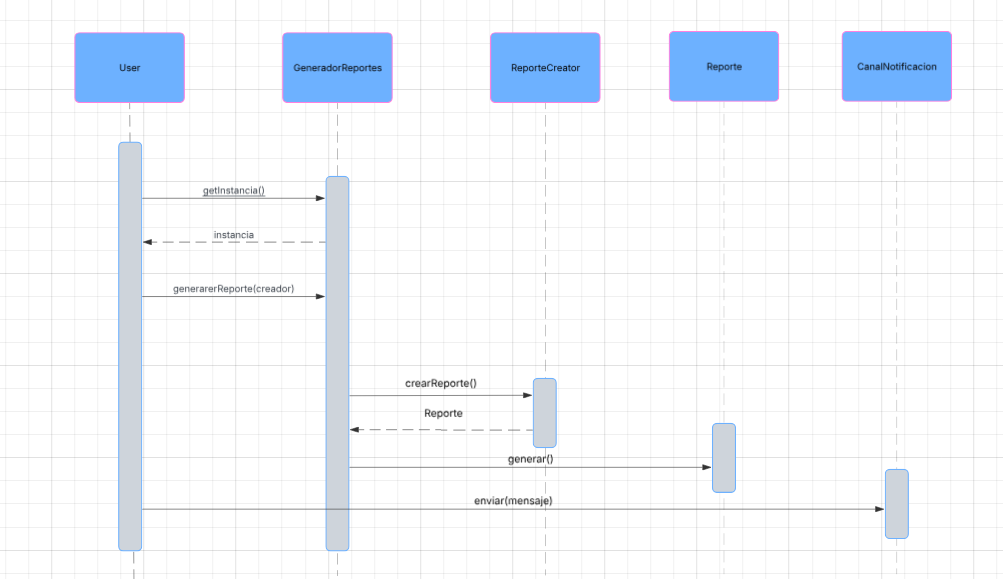
## **2.4. Patron Adapter en el sistema**



## **2.5. Patrón Bridge en el sistema**



**2.6 Diagrama de Secuencia**



# **SECCIÓN C: Implementación en JAVA**

<https://github.com/Jahir124/Taller06-Patterns/tree/main/src>