Grupo #4

Herramienta Usada

VIRTUAL PARADISE

Taller #4

Barrios Ureta Roberto Carlos

Macias Mendoza Christian Javier

Maldonado Bonilla Diego Alejandro

Tapia Loor Paulo Marcelo

Contenido

[Sección A 2](#_Toc181803766)

[1. Factory Method 2](#_Toc181803767)

[2. Builder 2](#_Toc181803768)

[3. Singleton 2](#_Toc181803769)

[4. Decorator 3](#_Toc181803770)

[5. Adapter 3](#_Toc181803771)

[Sección B 3](#_Toc181803772)

[Sección C 4](#_Toc181803773)

# Sección A

## 1. Factory Method

Motivación: Debido a que en el sistema manejan distintos formatos para los informes, sería ideal que dichos formatos se manejen en distintas instancias controlados por las subclases, además debido al manejar y variabilidad de los distintos formatos, se presenta una delegación de las responsabilidades o métodos que corresponde a cada objeto especifico y una a una clase general.

##### Pros:

* Mejor manejo y organización de objetos específicos bajo la mecánica de los distintos formatos.
* Facilita la delegación de otras responsabilidades a las subclases.

##### Contras:

* Complejidad del sistema y de la lógica de programación debido a las distintas subclases a manejar.

##### Principios SOLID:

* OCP que facilita la extensión del programa en otras subclases
* SRP el cual delega responsabilidades específicas para una clase especificas

## 2. Builder

Motivación: Debido a que el sistema posee distintos tipos de formato, es esencial tratar a los informes de manera modular de modo que la construcción de esta clase pueda soportar distintas representaciones. Es una alternativa al Factory method.

##### Pros:

* Maneja de forma sistemática el manejo de distintos tipos de una clase especifica, en este caso, el manejo de distintos formatos para los reportes.
* Facilita distintas representaciones de una misma clase.

##### Contras:

* Prolonga y agrega complejidad a la construcción de un objeto especifico.

##### Principios SOLID:

* OCP que facilita la extensión del programa en otras subclases sin el riesgo de modificar demasiado el programa.
* SRP el cual delega responsabilidades específicas para una clase especifica.

## 3. Singleton

Motivación: Para la manipulación de los reportes, el principio Singleton sostiene manejar una sola instancia para la visualización y generación de reporte en donde seria innecesario crear múltiples instancias para trabajar con todo lo que es la clase reporte.

##### Pros:

* Asegura usar instancias individuales para la generación de reportes.
* Menor consumo de recursos al trabajar con menos instancias.

##### Contras:

* Dificultades para pruebas individuales en el sistema

##### Principios SOLID:

* SRP el cual delega responsabilidades específicas para una clase específicas, en este caso la instancia individual como la generación de reportes tienen una sola responsabilidad.

## 4. Decorator

Motivación: Ya que cada reporte posee sus propias funciones de personalización como estilos, fuentes o colores, sería buena idea aplicar el patrón Decorator para agregar dichas funcionalidades a los objetos reporte específicos.

##### Pros:

* Agrega de forma dinámica nuevas funciones de personalización a los reportes.
* Permite la personalización modularizada de cada reporte.

##### Contras:

* Debido a las distintas capas para la agregación de funciones, esto puedo afectar el rendimiento y agregar complejidad.

##### Principios SOLID:

* SRP, ya que delega responsabilidades especificas a los objetos en este caso las opciones de personalización de cada reporte.
* OCP, ya que se agregan más funciones o responsabilidades al objeto reporte sin necesidad de modificar en gran manera el programa o código.

## 5. Adapter

Motivación: El sistema debe ser capaz de interactuar con interfaces de notificación muy diferentes entre sí Entonces se utilizaría para permitir que clases con interfaces incompatibles trabajen juntas. En este caso, el sistema de informes interactúe con cada servicio de notificación sin necesidad de cambiar la lógica interna de generación de reportes.

##### Pros:

* Simplifica la integración con sistemas externos que no siguen una interfaz común.
* Permite facilidad tanto el mantenimiento como una posible expansión del sistema.

##### Contras:

* Complica un poco al código debido a que crea una capa adicional de abstracción.
* Los adaptadores pueden volverse difíciles de mantener si las interfaces de los servicios de notificación cambian frecuentemente.

##### Principios SOLID:

* Principio de Inversión de Dependencias (DIP): El sistema de generación de reportes depende de interfaces de notificación, no de implementaciones concretas.
* Principio de Segregación de Interfaces (ISP): Cada adaptador implementa solo las interfaces que necesita, sin forzar alguna dependencia innecesaria.

# Sección B

# Sección C