**Aplicación de Patrones Creacionales y Principio Solid**

***Problema 1: Gestión de configuración global en una aplicación***

* **Contexto**: Una aplicación de escritorio necesita acceder a una configuración global (como idioma, tema, rutas de archivos, etc.) desde múltiples módulos. Es importante que esta configuración sea consistente y que solo exista una instancia en toda la aplicación.
* **Patrón aplicado**: Singleton
* Justificación: El patrón Singleton asegura que una clase tenga una única instancia y proporciona un punto de acceso global a ella. Esto es ideal para la configuración global, ya que:
  + Evita inconsistencias al tener múltiples instancias.
  + Facilita el acceso desde cualquier parte del sistema.
  + Controla el ciclo de vida del objeto de configuración.
* **Diagrama UML:**

Imagen que contiene Forma

El contenido generado por IA puede ser incorrecto.

***Problema 2: Creación de diferentes tipos de notificaciones***

* **Contexto**: Una plataforma de mensajería necesita enviar notificaciones por distintos canales: correo electrónico, SMS y notificaciones push. Cada tipo de notificación tiene una lógica de creación distinta, pero deben seguir una interfaz común.
* **Patrón aplicado**: Factory Method
* **Justificación**: El patrón Factory Method permite definir una interfaz para crear un objeto, pero deja que las subclases decidan qué clase instanciar. Es útil aquí porque:
  + Permite extender fácilmente nuevos tipos de notificaciones sin modificar el código existente.
  + Encapsula la lógica de creación de cada tipo de notificación.
  + Favorece el principio de abierto/cerrado (Open/Closed Principle).
* **Diagrama UML:**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

***Problema 3: Construcción de reportes complejos***

* **Contexto:** Una aplicación de análisis de datos genera reportes en PDF que pueden incluir gráficos, tablas, resúmenes y secciones opcionales. La construcción de estos reportes varía según el tipo de usuario (analista, gerente, cliente).
* **Patrón aplicado:** **Builder**
* **Justificación:** El patrón Builder separa la construcción de un objeto complejo de su representación, permitiendo crear diferentes representaciones con el mismo proceso de construcción. Es ideal para este caso porque:
  + Permite construir reportes paso a paso.
  + Facilita la reutilización del proceso de construcción para distintos tipos de reportes.
  + Mejora la **legibilidad y mantenibilidad** del código al separar la lógica de construcción.
* **Diagrama UML:**

Forma

El contenido generado por IA puede ser incorrecto.

Link github:

<https://github.com/KevinAPerez/Ptrones_Diseno_Software/tree/e0104cf971983a1be89248ac0700fb7821bd286b/Patrones%20Creacionales%20y%20Principio%20Solid>