Trabajo Práctico: Patrones Creacionales

Cátedra: Programación Orientada a Objetos II
Tema: Aplicación de Patrones de Diseño Creacionales
Lenguaje: Java

---

1. Objetivo

El objetivo de este trabajo práctico es que el alumno sea capaz de:

1. Analizar un conjunto de requerimientos de software.
2. Identificar los problemas de diseño creacional (instanciación) presentes en el sistema.
3. Seleccionar y aplicar de forma adecuada los patrones de diseño **Singleton**, **Factory** (Factory Method o Abstract Factory) y **Builder**.
4. Justificar por escrito las decisiones de diseño tomadas.
5. Modelar la solución mediante un diagrama de clases UML.

---

2. Escenario del Problema: Sistema de Configuración y Renderizado de Reportes

Se debe diseñar el núcleo de un sistema de generación de reportes para una aplicación de análisis de datos. Este sistema será utilizado por múltiples módulos de la aplicación (ej. el módulo de finanzas, el módulo de marketing, el módulo de RRHH).

El sistema tiene tres grandes desafíos de diseño que deben ser resueltos.

Requerimiento 1: El Motor de Renderizado

El sistema debe ser capaz de exportar los reportes en diferentes formatos. Por ahora, se requiere soporte para **ReportePDF**, **ReporteExcel** y **ReporteCSV**.

- Cada tipo de reporte (PDF, Excel, CSV) tiene una lógica de "renderizado" completamente diferente y se representa mediante una clase distinta (RenderizadorPDF, RenderizadorExcel, etc.).
- El código cliente (por ejemplo, el módulo de Finanzas) no debe acoplarse a las clases concretas. El cliente solo debe poder decir: "Necesito un motor de renderizado para 'PDF'" y recibir el objeto adecuado.
- El sistema debe ser fácil de extender en el futuro para agregar nuevos formatos (ej. ReporteXML) sin modificar el código cliente existente.

Requerimiento 2: La Construcción de los Reportes

Un objeto Reporte es una entidad compleja. Al momento de crear un reporte, se deben poder especificar múltiples parámetros, pero no todos son siempre necesarios.

- **Datos ObligatorIOS:**
    - titulo (String)
    - cuerpoPrincipal (String)
- **Datos Opcionales:**
    - encabezado (String)
    - pieDePagina (String)
    - fecha (LocalDate)
    - autor (String)
    - orientacion (Enum: VERTICAL, HORIZONTAL)

El método para crear una instancia de Reporte debe ser **limpio y legible**. Se debe evitar a toda costa el uso de un "constructor telescópico" (múltiples constructores) o un constructor que reciba 7 parámetros, obligando al cliente a pasar null para los valores opcionales.

Requerimiento 3: El Gestor de Configuración Global

Toda la aplicación (incluyendo los módulos de Finanzas, Marketing y el propio sistema de reportes) necesita acceder a configuraciones globales, como:

- La URL de la base de datos (String urlBd)
- El nombre de usuario de la BD (String userBd)
- El directorio de salida para los reportes (String pathReportes)

Debe existir un **único punto de acceso** a esta configuración en toda la aplicación. Es fundamental garantizar que solo exista **una y solo una instancia** del objeto GestorConfiguracion cargada en memoria. Crear múltiples objetos de configuración sería ineficiente y podría llevar a inconsistencias en los datos.

---

3. Entregables

El trabajo práctico debe ser entregado en un repositorio Git e incluir los siguientes elementos:

1. Documento de Justificación (README.md)

Un archivo README.md en la raíz del proyecto que explique claramente:

- **Para el Requerimiento 1 (Motor de Renderizado):**
    - ¿Qué patrón de diseño creacional eligieron?
    - ¿Por qué este patrón es la solución adecuada para este problema?
    - ¿Qué problema(s) evita (ej. acoplamiento, violación del principio Abierto/Cerrado)?
- **Para el Requerimiento 2 (Construcción de Reportes):**
    - ¿Qué patrón de diseño creacional eligieron?
    - ¿Por qué este patrón es la solución adecuada?
    - ¿Qué problemas específicos del "constructor" resuelve?
- **Para el Requerimiento 3 (Gestor de Configuración):**
    - ¿Qué patrón de diseño creacional eligieron?
    - ¿Por qué este patrón es la solución adecuada para este requerimiento?
    - ¿Cómo garantizaron la unicidad de la instancia?

2. Código Fuente en Java

El proyecto Java completo y funcional que implemente la solución a los tres requerimientos. El código debe ser claro, estar bien comentado y seguir las convenciones de Java. Debe incluir una clase Main (o tests) que demuestre el funcionamiento de la solución.

3. Diagrama de Clases UML

Un diagrama de clases de la solución final. Puede ser una imagen (.png, .jpg) o un archivo de código (ej. Mermaid, PlantUML). El diagrama debe mostrar todas las clases, interfaces, sus atributos, métodos y las relaciones entre ellas (herencia, implementación, asociación, etc.).