**PATRON FABRICA**



El **Factory** es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, pero permite que las subclases alteren el tipo de objetos que se crean. Esto permite la creación de objetos sin especificar su clase concreta, promoviendo la flexibilidad y desacoplando el código.

**Método de Uso**

**Crear una interfaz o clase base:** Define una interfaz o clase abstracta que declare el método de creación de objetos.

**Implementar clases concretas:** Desarrolla múltiples implementaciones concretas de la interfaz o clase base.

**Definir una clase creadora con un Factory:** La clase creadora declara un método abstracto que devolverá objetos de la interfaz o clase base.

**Extender la clase creadora:** Las subclases implementarán el método Factory para instanciar objetos específicos.

**Sustituir la instanciación directa:** En el código cliente, en lugar de usar new, se usará el método Factory.

**Ventajas del Factory**

1. **Desacoplamiento**: El código cliente no necesita conocer las clases concretas, solo la interfaz o clase base.
2. **Flexibilidad**: Permite extender el código sin modificar las clases existentes.
3. **Principio de Responsabilidad Única (SRP)**: Separa la lógica de creación de objetos de su uso.
4. **Cumple con el Principio Abierto/Cerrado (OCP)**: Puedes agregar nuevas clases sin modificar el código existente.

**Desventajas del Factory**

1. **Complejidad adicional**: Puede aumentar la cantidad de código y clases, lo que hace más difícil su mantenimiento en aplicaciones pequeñas.
2. **Dependencia de subclases**: Para modificar el comportamiento de creación, se debe crear una nueva subclase en lugar de modificar un método existente.
3. **Posible sobrecarga en el rendimiento**: Si se crean muchas clases, puede afectar el rendimiento y la legibilidad del código.

**¿Cuándo usar Factory?**

Cuando el código debe depender de interfaces en lugar de implementaciones concretas.  
Cuando se necesita delegar la responsabilidad de creación a subclases para mayor flexibilidad.  
Cuando la creación de objetos implica lógica compleja y no simplemente instanciaciones directas.

**¿Cuándo evitar Factory?**

Cuando la cantidad de clases adicionales no justifica su uso.  
Cuando una simple creación de objetos con new es suficiente.  
Cuando el diseño no requiere personalización del proceso de creación.

***Explicación de Código***

Este patrón se usa en la clase ConexionFabrica.java, la cual es responsable de crear instancias de distintos tipos de conexión (ConexionMySQL y ConexionOracle).

1. **Interfaz IConexion2** → Define los métodos conectar() y desconectar(), que serán implementados por diferentes tipos de conexiones.
2. **Clases ConexionMySQL y ConexionOracle** → Implementan IConexion2, proporcionando su propia versión de la conexión y desconexión.
3. **Clase ConexionFabrica** → Contiene el método getConexion(String motor), que determina qué tipo de conexión se debe crear dependiendo del parámetro recibido (MYSQL u ORACLE).
4. **Clase APP2.java** → Es el punto de entrada de la aplicación y utiliza la fábrica para obtener y usar una conexión sin preocuparse por su tipo específico.

**Beneficios del Factory Method**

Desacopla la lógica de creación de objetos, lo que facilita la extensión y mantenimiento del código.  
Permite cambiar los tipos de objetos creados sin modificar el código que los usa.  
Fomenta el principio de inversión de dependencias, ya que el código principal no depende directamente de clases concretas.

Por ejemplo, si en el futuro necesitas agregar otro tipo de conexión (como PostgreSQL), solo debes crear una nueva clase ConexionPostgreSQL e integrarla en ConexionFabrica sin tocar APP2.java.