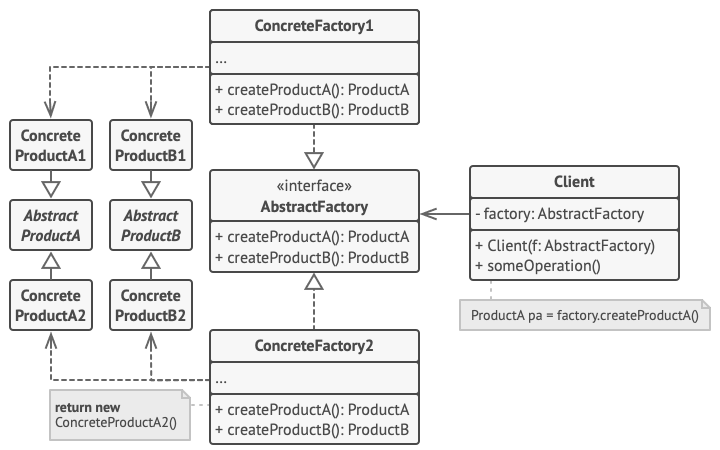
**Abstract Factory**



El Abstract Factory es un patrón de diseño creacional que proporciona una interfaz para crear familias de objetos relacionados sin especificar sus clases concretas. Se utiliza para asegurar que los objetos creados sean compatibles entre sí, promoviendo la cohesión y el desacoplamiento en el código.

**Método de Uso**

Definir una interfaz o clase abstracta (Fábrica abstracta): Esta interfaz declara los métodos para crear diferentes tipos de objetos relacionados.

Implementar fábricas concretas: Cada fábrica concreta implementa la fábrica abstracta y proporciona las instancias específicas de los objetos.

Definir interfaces o clases base para los productos: Estas interfaces son implementadas por las clases concretas que serán creadas por las fábricas.

Crear clases concretas para cada tipo de producto: Estas son las clases reales que serán instanciadas.

Sustituir la instanciación directa: En el código cliente, en lugar de usar new, se usa la fábrica abstracta para obtener los objetos correctos.

**Ventajas del Abstract Factory**

1. Desacoplamiento: Separa la lógica de creación de objetos de su uso, facilitando el mantenimiento y la escalabilidad.
2. Consistencia: Asegura que los objetos creados sean compatibles entre sí dentro de una misma familia.
3. Cumple el Principio de Responsabilidad Única (SRP): La creación de objetos está encapsulada en una fábrica, reduciendo la complejidad del código cliente.
4. Cumple el Principio Abierto/Cerrado (OCP): Se pueden agregar nuevas familias de productos sin modificar el código existente.

**Desventajas del Abstract Factory**

1. Aumento en la complejidad: Puede llevar a la creación de muchas clases y fábricas, lo que puede hacer el código más difícil de mantener en proyectos pequeños.
2. Rigidez: Agregar nuevos productos dentro de una familia existente puede requerir modificaciones en todas las fábricas concretas.
3. Mayor esfuerzo en la configuración inicial: Comparado con Factory Method, este patrón requiere definir más interfaces y clases para garantizar la abstracción.

**¿Cuándo usar Abstract Factory?**

Cuando se necesita crear grupos de objetos relacionados sin acoplar el código a clases específicas.  
Cuando el sistema debe ser escalable, permitiendo agregar nuevas familias de objetos sin afectar el código cliente.  
Cuando es importante garantizar la compatibilidad entre objetos creados por la misma fábrica.

**¿Cuándo evitar Abstract Factory?**

Cuando el número de familias de productos es pequeño y no justifica la complejidad adicional.  
Cuando no se necesita garantizar la compatibilidad entre los productos creados.  
Cuando el código puede manejarse bien con una simple fábrica en lugar de una abstracta.

**Explicación del Código**

Este patrón se usa en la clase FabricaAbstracta.java, que define la estructura de la fábrica abstracta para crear diferentes tipos de conexiones y servicios.

Interfaz FabricaAbstracta → Define los métodos getConexionBD() y getREST(), que deben ser implementados por las fábricas concretas.

Clase FabricaBD → Implementa FabricaAbstracta y se encarga de crear conexiones de base de datos (ConexionMySQL).

Clase FabricaREST → Implementa FabricaAbstracta y se encarga de crear servicios REST (ServicioVentas).

Código cliente (Main.java) → Utiliza la fábrica abstracta para obtener instancias de conexiones a bases de datos o servicios REST sin conocer su implementación concreta.

**Beneficios del Abstract Factory**

Desacoplamiento total entre la lógica de creación y el código cliente.  
Permite cambiar fácilmente familias de objetos sin modificar el código existente.  
Facilita la adición de nuevos tipos de productos dentro de una familia sin afectar el código base.

Ejemplo práctico: Si en el futuro se necesita agregar una nueva base de datos (PostgreSQL) o un nuevo servicio REST, solo se deben crear nuevas fábricas y clases sin modificar el código que ya está funcionando.