**Factory Method**

El patrón Factory Method es un patrón de diseño creacional que se utiliza para crear objetos sin especificar la clase exacta de objeto que se creará. En lugar de llamar directamente al constructor de una clase para crear un objeto, el patrón Factory Method define un método que se utiliza para crear el objeto, permitiendo a las subclases cambiar el tipo de objetos que se crearán.

**Funcionamiento:**

El funcionamiento básico del patrón Factory Method implica la definición de una interfaz o clase abstracta que declara el método de fábrica. Las subclases concretas implementan este método para crear instancias de clases concretas.

**Situación que resuelve:**

El patrón Factory Method es útil cuando una clase no puede anticipar la clase de objetos que debe crear, o cuando una clase quiere delegar la responsabilidad de crear objetos a sus subclases. El patrón Factory Method es útil cuando una clase necesita crear objetos, pero no conoce exactamente el tipo de objeto que necesitará hasta tiempo de ejecución. Esto permite una mayor flexibilidad y extensibilidad en el diseño del código, ya que las subclases pueden proporcionar implementaciones diferentes del método de fábrica para crear objetos específicos.

**Casos típicos para su uso:**

1. Cuando una clase no puede anticipar la clase de objetos que debe crear.
2. Cuando una clase quiere delegar la responsabilidad de la creación de objetos a sus subclases.
3. Cuando se desea proporcionar una forma de extender y personalizar la creación de objetos sin modificar el código existente.
4. Cuando se necesita ocultar la lógica de creación de objetos de los clientes.
5. Cuando se quiere proporcionar una interfaz común para crear diferentes tipos de objetos.

**Ejemplo Simple:**

Supongamos que tenemos una interfaz IVehiculo con métodos Mover() y Detenerse(). Podemos tener una clase FabricaVehiculos que tenga un método CrearVehiculo(). Luego, las subclases de FabricaVehiculos, como FabricaCoche y FabricaBicicleta, implementarán este método para crear instancias de Coche y Bicicleta, respectivamente.

interface IVehiculo

{

void Mover();

void Detenerse();

}

class Coche : IVehiculo

{

public void Mover()

{

Console.WriteLine("El coche se está moviendo.");

}

public void Detenerse()

{

Console.WriteLine("El coche se ha detenido.");

}

}

class FabricaVehiculos

{

public virtual IVehiculo CrearVehiculo()

{

return null;

}

}

class FabricaCoche : FabricaVehiculos

{

public override IVehiculo CrearVehiculo()

{

return new Coche();

}

}

// Uso:

FabricaVehiculos fabrica = new FabricaCoche();

IVehiculo coche = fabrica.CrearVehiculo();

coche.Mover();

-----------------------------------------------------------------//-----------------------------------------------------------------------

**Ejemplo Realista:**

Imagina un sistema de gestión de documentos donde diferentes tipos de documentos (como documentos de texto, PDFs, hojas de cálculo, etc.) necesitan ser creados y manipulados. Utilizando el patrón Factory Method, podemos tener una clase FabricaDocumentos con métodos específicos para crear diferentes tipos de documentos.