Factory Design Pattern

The Factory Pattern defines an interface for creating an object, but allows subclasses to alter the type of objects that will be created.

**Why Use the Factory Pattern?**

* To encapsulate object creation
* To simplify code and promote loose coupling
* When object creation is complex or depends on logic (like a string or a configuration)
* When a class can't anticipate the class of objects it must create

**🧱 Components of the Factory Pattern**

1. **Product Interface** – common interface for all products
2. **Concrete Products** – actual classes implementing the interface
3. **Factory Class** – contains logic to create and return instances

## C# Example of Factory Pattern

### Step 1: Create the Product Interface

public interface IVehicle

{

void Drive();

}

Step 2: Create Concrete Classes

public class Car : IVehicle

{

public void Drive()

{

Console.WriteLine("Driving a Car");

}

}

public class Bike : IVehicle

{

public void Drive()

{

Console.WriteLine("Riding a Bike");

}

}

Step 3: Create the Factory

public class VehicleFactory

{

public IVehicle GetVehicle(string type)

{

switch (type.ToLower())

{

case "car":

return new Car();

case "bike":

return new Bike();

default:

throw new ArgumentException("Invalid vehicle type");

}

}

}

Step 4: Use the Factory in Client Code

class Program

{

static void Main(string[] args)

{

VehicleFactory factory = new VehicleFactory();

IVehicle myCar = factory.GetVehicle("car");

myCar.Drive(); // Output: Driving a Car

IVehicle myBike = factory.GetVehicle("bike");

myBike.Drive(); // Output: Riding a Bike

}

}

## When to Use the Factory Pattern

* When a class doesn’t know what sub-classes it may need to create
* To decouple object creation from the business logic
* To follow **Open/Closed Principle** (open for extension, closed for modification)

## 🔄 Variants of Factory Pattern

* **Simple Factory** – as shown above (technically not a GoF pattern, but commonly used)
* **Factory Method** – lets subclasses decide which class to instantiate
* **Abstract Factory** – used for creating families of related or dependent objects

## 🔍 Interview Tips

### Q: What are the advantages of using the Factory Pattern?

* Loose coupling
* Better code maintainability
* Simplified object creation logic
* Supports Open/Closed Principle

### Q: How is Factory Pattern different from Abstract Factory?

* **Factory**: Creates a single product.
* **Abstract Factory**: Creates **families of related products** without specifying their concrete classes.