Stefan Djokic

Factory

Design Pattern

Simplifed

swipe >>>

Clients: We want to buy a BMW

Dealership: My client wants to buy a **BMW**

Factory: Okay, let's produce **BMW**

When a dealership orders a specific type of car from the factory, they don't need to understand the intricacies of how the car is assembled, painted, tested, etc. They just place an order specifying the car type, and the factory takes care of the rest.

Car Factory = Factory in Factory Design Pattern
Dealership = Client. Interacts with the interface

The dealership doesn't need to know the internal workings of the car factory == the client code does not need to understand how the BMW, Audi, or Mercedes objects are created.
All the client needs to call the CreateCar() method on the CarFactory, and it gets the car object.
The Factory Pattern encapsulates the creation logic and hides the complexities of object creation from the client.

Let's see the code 👉

Abstract and concretes

```
public abstract class Car
{
    public abstract void Assemble();
}

public class BMW : Car
{
    public override void Assemble()
    {
        Console.WriteLine("Assembling BMW Car.");
    }
}

public class Audi : Car
{
    public override void Assemble()
    {
        Console.WriteLine("Assembling Audi Car.");
    }
}

public class Mercedes : Car
{
    public override void Assemble()
    {
        Console.WriteLine("Assembling Mercedes Car.");
    }
}
```

CarFactory = Factory in Factory Design Pattern

```
public class CarFactory
{
    public Car CreateCar(string carType)
    {
        switch (carType.ToLower())
        {
            case "bmw":
                return new BMW();
            case "audi":
                return new Audi();
            case "mercedes":
                return new Mercedes();
            default:
                throw new ArgumentException
                  ($"Invalid car type: {carType}");
        }
    }
}
```

Main = Dealership

```
static void Main()
{
    CarFactory carFactory =
                  new CarFactory();

    carFactory.AssembleCar("BMW");
    carFactory.AssembleCar("Audi");
    carFactory.AssembleCar("Mercedes");
}
```

When to use it?

1. **Object creation is complex:** This simplifies the client code, as it only needs to call the factory method instead of dealing with the details of object creation.
2. **Dependency management:** If a class depends on concrete implementations, using the Factory Pattern allows you to inject the factory as a dependency.
3. **Loose coupling**: By using the Factory Pattern, the client code does not need to know the exact class of the objects it creates.
4. **Extensibility**: You can simply create new concrete classes and update the factory, without modifying the client code.
5. **Unit testing**: Factories can be easily mocked or replaced with stubs during unit testing, enabling isolated testing of client code.

WANT MORE POSTS LIKE THIS?

CLICK ON THE NOTIFICATION BELL ON MY PROFILE 🔔