Design Pattern: Factory

When we want to create objects, we usually use the 'new' keyword. But imagine if the project grows and we need to create many different types of objects. Wouldn’t it be better to delegate the creation of these objects to a 'factory'? This is where the Factory Pattern comes in.

What is the Factory Pattern?

Factory is one of the Creational Design Patterns. Its goal is to centralize object creation logic in one place, instead of spreading 'new' statements across the code.

Advantages

- Separation of object creation logic from usage logic
- Follows the Single Responsibility Principle
- Flexibility to add new types without changing existing code

Example in C#

```
// Interface
public interface IShape
{     void Draw(); }

// Concrete Classes public class Circle : IShape {     public
void Draw() => Console.WriteLine("Drawing a Circle"); }

public class Square : IShape
{
    public void Draw() => Console.WriteLine("Drawing a Square");
}
public class ShapeFactory
{
    public IShape CreateShape(string type)
{
        return type switch
{
            "Circle" => new Circle(),
            "Square" => new Square(),             _ => throw new
ArgumentException("Unknown shape type")
        };
    }
}
var factory = new ShapeFactory();

IShape circle = factory.CreateShape("Circle");
circle.Draw();

IShape square = factory.CreateShape("Square");
square.Draw();
```

Why is Factory important?

- We get cleaner and more flexible code.
- Adding a new shape (e.g., Triangle) requires only a new class and factory update.

- Code that uses shapes does not need to change at all.

Conclusion

The Factory Pattern helps us reduce the direct use of 'new' and centralizes object creation in one place. It is especially useful in large projects where many objects need to be created.

Author: **Morteza Akbari**