الگوی طراحی Factory :

وقتی می‌خوایم Objects بسازیم، معمولاً سراغ newمی‌ریم. ولی فرض کن پروژه بزرگ شده و باید انواع مختلفی از Objects بسازیم...

اینجا هر جا بخوای newبزنی، کد می‌شه شلوغ و به هم ریخته!

راه حلش؟ دادن مسئولیت ساخت Objects به یه .Factory

اینجاست که Factory Patternوارد می‌شه و همه چیز مرتب می‌شه.

Factory Pattern چیه؟

Factory یکی از الگوهای **Creational Design Patterns**هست که کارش متمرکز کردن منطق ساخت Objects تو یه جاست.
به جای اینکه هرجا newبزنی، فقط می‌گی:

```
var circle = factory.CreateShape("Circle");
```

و Factory خودش می‌دونه چه چیزی بسازه.

مزایایFactory

- جداسازی ساخت Object از استفاده‌ش✓
- رعایت اصل ✓ **Single Responsibility Principle**
- راحت اضافه کردن نوع جدید بدون دست زدن به کدهای قبلی✓

چرا Factory مهمه؟

- کد **تمیزتر و منعطف‌تر** می‌شه
- اضافه کردن Shape جدید مثل **Triangle** خیلی ساده‌ست: فقط یه کلاس جدید و یه خط توFactory
- کدی که از Shape استفاده می‌کنه، بدون تغییر باقی می‌مونه

جمع‌بندی

Factory Patternباعث می‌شه وابستگی مستقیم به `new`کم بشه،
کد مرتب و قابل نگهداری باشه و پروژه وقتی بزرگ شد، راحت توسعه پیدا کنه.
همین الگو مخصوصاً تو پروژه‌های بزرگ که تعداد زیادی Object ساخته می‌شه، نجات‌بخش برنامه‌نویس‌هاست!

یک مثال ساده در#C

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

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

// Factory
public class ShapeFactory
{
  public IShape CreateShape(string type)
  {
    return type switch
    {
      "Circle" => new Circle(),
      "Square" => new Square(),
_ => throw new ArgumentException("Unknown shape type")
    };
  }
}

//استفاده

var factory = new ShapeFactory();

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

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

✍نویسنده :**مرتضی اکبری**