1. Singleton *(Creational)*

Amaç: Bir sınıfın yalnızca tek bir örneğinin olmasını sağlamak.
Kullanım Alanı: Logger, Config Manager, Database Connection.

```
public class Singleton
{
    private static Singleton _instance;
    private static readonly object _lock = new object();

    private Singleton() { }

    public static Singleton Instance
    {
        get
        {
            lock (_lock)
            {
                if (_instance == null)
                    _instance = new Singleton();
                return _instance;
            }
        }
    }

    public void Log(string message)
    {
        Console.WriteLine($"[LOG] {message}");
    }
}

// Kullanım
Singleton.Instance.Log("Uygulama başlatıldı.");
```

2. Factory Method *(Creational)*

Amaç: Nesne oluşturma işini alt sınıflara bırakmak.
Kullanım Alanı: Farklı tipte nesneler üretmek.

```
public abstract class Shape
{
    public abstract void Draw();
}

public class Circle : Shape
{
    public override void Draw() => Console.WriteLine("Daire
çizildi.");
}

public class Square : Shape
{
    public override void Draw() => Console.WriteLine("Kare
çizildi.");
}

public abstract class ShapeFactory
{
    public abstract Shape CreateShape();
}

public class CircleFactory : ShapeFactory
{
    public override Shape CreateShape() => new Circle();
}

public class SquareFactory : ShapeFactory
{
    public override Shape CreateShape() => new Square();
}

// Kullanım
ShapeFactory factory = new CircleFactory();
Shape shape = factory.CreateShape();
shape.Draw();
```

3. Adapter *(Structural)*

Amaç: Uyumlu olmayan arayüzleri bağlamak.
Kullanım Alanı: Eski sistem ile yeni sistem entegrasyonu.

```
public interface ITarget
{
    void Request();
}

public class OldSystem
{
    public void SpecificRequest() => Console.WriteLine("Eski
sistem çalıştı.");
}

public class Adapter : ITarget
{
    private OldSystem _oldSystem = new OldSystem();
    public void Request() => _oldSystem.SpecificRequest();
}

// Kullanım
ITarget target = new Adapter();
target.Request();
```

4. Decorator *(Structural)*

Amaç: Nesneye dinamik olarak yeni özellik eklemek.
Kullanım Alanı: UI bileşenleri, sipariş sistemleri.

```
public interface ICoffee
{
    string GetDescription();
    double GetCost();
}

public class SimpleCoffee : ICoffee
{
    public string GetDescription() => "Sade kahve";
    public double GetCost() => 5.0;
```

```
}

public class MilkDecorator : ICoffee
{
    private ICoffee _coffee;
    public MilkDecorator(ICoffee coffee) => _coffee = coffee;

    public string GetDescription() => _coffee.GetDescription()
+ ", süt";
    public double GetCost() => _coffee.GetCost() + 2.0;
}

// Kullanım
ICoffee coffee = new SimpleCoffee();
coffee = new MilkDecorator(coffee);
Console.WriteLine($"{coffee.GetDescription()} -
{coffee.GetCost()}₺");
```

5. Observer *(Behavioral)*

Amaç: Bir nesnedeki değişiklikleri diğerlerine bildirmek.
Kullanım Alanı: Event sistemleri, bildirim mekanizmaları.

```
public interface IObserver
{
    void Update(string message);
}

public class User : IObserver
{
    public string Name { get; }
    public User(string name) => Name = name;
    public void Update(string message) =>
Console.WriteLine($"{Name} bildirimi aldı: {message}");
}

public class NotificationService
{
    private List<IObserver> observers = new List<IObserver>();
```

```
    public void Subscribe(IObserver observer) =>
observers.Add(observer);
    public void Unsubscribe(IObserver observer) =>
observers.Remove(observer);

    public void Notify(string message)
    {
        foreach (var observer in observers)
            observer.Update(message);
    }
}

// Kullanım
var service = new NotificationService();
var user1 = new User("Ahmet");
var user2 = new User("Mehmet");

service.Subscribe(user1);
service.Subscribe(user2);

service.Notify("Yeni mesajınız var!");
```

6. Strategy *(Behavioral)*

Amaç: Algoritmaları değiştirilebilir yapmak.
Kullanım Alanı: Sıralama, ödeme yöntemleri.

```
public interface ISortStrategy
{
    void Sort(List<int> list);
}

public class BubbleSort : ISortStrategy
{
    public void Sort(List<int> list) =>
Console.WriteLine("Bubble Sort ile sıralandı.");
}

public class QuickSort : ISortStrategy
{
```

```
    public void Sort(List<int> list) =>
Console.WriteLine("Quick Sort ile sıralandı.");
}

public class Sorter
{
    private ISortStrategy _strategy;
    public Sorter(ISortStrategy strategy) => _strategy =
strategy;
    public void SetStrategy(ISortStrategy strategy) =>
_strategy = strategy;
    public void Sort(List<int> list) => _strategy.Sort(list);
}

// Kullanım
var sorter = new Sorter(new BubbleSort());
sorter.Sort(new List<int> { 5, 3, 1 });

sorter.SetStrategy(new QuickSort());
sorter.Sort(new List<int> { 5, 3, 1 });
```

7-23. Diğer Pattern'ler (Kısa Format)

Abstract Factory – İlişkili nesneler ailesi oluşturur.

```
interface IFactory { IButton CreateButton(); }
```

Builder – Karmaşık nesneleri adım adım oluşturur.

```
class Director { public void Construct(Builder b) {
b.BuildPartA(); b.BuildPartB(); } }
```

Prototype – Mevcut nesneden kopya oluşturur.

```
class Prototype { public Prototype Clone() =>
(Prototype)this.MemberwiseClone(); }
```

Bridge – Soyutlama ile implementasyonu ayırır.

```
class Abstraction { protected IImplementor impl; public void Operation() => impl.OperationImpl(); }
```

Composite – Nesneleri ağaç yapısında temsil eder.

```
class Composite : Component { List<Component> children = new(); }
```

Facade – Karmaşık sistemi basit arayüzle sunar.

```
class Facade { Sub1 s1 = new(); Sub2 s2 = new(); }
```

Flyweight – Hafızadan tasarruf için veri paylaşır.

```
class FlyweightFactory { Dictionary<string, Flyweight> pool = new(); }
```

Proxy – Nesneye erişimi kontrol eder.

```
class Proxy : ISubject { RealSubject real; }
```

Command – İstekleri nesne olarak kapsüller.

```
interface ICommand { void Execute(); }
```

Iterator – Koleksiyon elemanlarını sırayla dolaşır.

```
interface IIterator { bool HasNext(); object Next(); }
```

Mediator – Nesneler arası iletişimi merkezileştirir.

```
class Mediator { public void Send(string msg, Colleague c) { }
}
```

Memento – Nesnenin durumunu kaydeder/geri yükler.

```
class Memento { public string State { get; } }
```

State – Nesnenin durumuna göre davranışını değiştirir.

```
interface IState { void Handle(); }
```

Template Method – Algoritmanın iskeletini tanımlar.

```
abstract class Template { public void Run() { Step1();
Step2(); } }
```

Chain of Responsibility – İstekleri zincir halinde işler.

```
abstract class Handler { protected Handler next; }
```

Interpreter – Dil veya ifadeleri yorumlar.

```
interface IExpression { int Interpret(); }
```

Visitor – Nesneler üzerinde yeni işlemler tanımlar.

```
interface IVisitor { void Visit(Element e); }
```

Pattern Kullanım Karar Tablosu

Problem / İhtiyaç	Önerilen Pattern	Kategori
Tek bir nesneye global erişim lazım	Singleton	Creational
Nesne oluşturma sürecini alt sınıflara bırakmak	Factory Method	Creational
Birbiriyle ilişkili nesneler ailesi oluşturmak	Abstract Factory	Creational
Karmaşık nesneleri adım adım oluşturmak	Builder	Creational
Mevcut nesneden kopya oluşturmak	Prototype	Creational
Uyumlu olmayan arayüzleri bağlamak	Adapter	Structural
Soyutlama ile implementasyonu ayırmak	Bridge	Structural
Nesneleri ağaç yapısında temsil etmek	Composite	Structural
Nesneye dinamik özellik eklemek	Decorator	Structural
Karmaşık sistemi basit arayüzle sunmak	Facade	Structural
Hafızadan tasarruf için veri paylaşmak	Flyweight	Structural
Nesneye erişimi kontrol etmek	Proxy	Structural
Bir nesnedeki değişiklikleri diğerlerine bildirmek	Observer	Behavioral
Algoritmaları değiştirilebilir yapmak	Strategy	Behavioral
İstekleri nesne olarak kapsüllemek	Command	Behavioral
Koleksiyon elemanlarını sırayla dolaşmak	Iterator	Behavioral
Nesneler arası iletişimi merkezileştirmek	Mediator	Behavioral
Nesnenin durumunu kaydetmek/geri yüklemek	Memento	Behavioral
Nesnenin durumuna göre davranış değiştirmek	State	Behavioral
Algoritmanın iskeletini tanımlamak	Template Method	Behavioral
İstekleri zincir halinde işlemek	Chain of Responsibility	Behavioral
Dil veya ifadeleri yorumlamak	Interpreter	Behavioral
Nesneler üzerinde yeni işlemler tanımlamak	Visitor	Behavioral