# ****الگوی Decorator در C#****

**تعریف الگوی Decorator**

الگوی **Decorator** یک الگوی **ساختاری (Structural)** است که **به صورت پویا** مسئولیت‌های جدیدی به یک شیء **اضافه می‌کند**، بدون اینکه ساختار اصلی آن را تغییر دهد. این الگو جایگزین مناسبی برای ارث‌بری چندگانه است و از **ترکیب (Composition)** به جای ارث‌بری استفاده می‌کند.

**نام‌های دیگر:**

* Wrapper (پوشاننده)
* Dynamic Decorator

**مشکلی که Decorator حل می‌کند**

فرض کنید:

* یک کلاس اصلی دارید که نمی‌خواهید آن را تغییر دهید، اما نیاز دارید **قابلیت‌های جدیدی** به آن اضافه کنید.
* استفاده از **ارث‌بری** باعث ایجاد **انبوهی از کلاس‌های مشتق‌شده** می‌شود (مثلاً برای هر ترکیب ویژگی‌ها یک کلاس جدید).

**مثال واقعی:**

* اضافه کردن **امکانات جدید به یک ماشین** (مثلاً سیستم صوتی، سانروف، روکش چرم) بدون تغییر کلاس اصلی ماشین.
* افزودن **عملکردهای اضافی به یک فایل** (مثلاً فشرده‌سازی، رمزنگاری) بدون تغییر کلاس اصلی فایل.

**ساختار الگوی Decorator در #C**

**1. کامپوننت پایه (Component Interface)**

رابطی که تمام کلاس‌های اصلی و دکوراتورها باید پیاده‌سازی کنند:

public interface ICar {

string GetDescription();

decimal GetPrice();

}

### ****2. کلاس Concrete Component ( پیاده‌سازی اصلی )****

کلاسی که می‌خواهیم آن را "تزئین" کنیم:

public class BasicCar : ICar {

public string GetDescription() => "ماشین پایه";

public decimal GetPrice() => 100\_000;

}

### ****3. کلاس Decorator پایه****

کلاس پایه برای تمام دکوراتورها:

public abstract class CarDecorator : ICar {

protected ICar \_car;

public CarDecorator(ICar car) {

\_car = car;

}

public virtual string GetDescription() => \_car.GetDescription();

public virtual decimal GetPrice() => \_car.GetPrice();

}

### ****4. دکوراتورهای Concrete (پیاده‌سازی شده)****

هر دکوراتور یک ویژگی جدید اضافه می‌کند:

public class LeatherSeatsDecorator : CarDecorator {

public LeatherSeatsDecorator(ICar car) : base(car) { }

public override string GetDescription() => \_car.GetDescription() + "، روکش چرم";

public override decimal GetPrice() => \_car.GetPrice() + 20\_000;

}

public class SunroofDecorator : CarDecorator {

public SunroofDecorator(ICar car) : base(car) { }

public override string GetDescription() => \_car.GetDescription() + "، سانروف";

public override decimal GetPrice() => \_car.GetPrice() + 15\_000;

}

### ****5. کلاینت (استفاده از دکوراتورها)****

class Program {

static void Main(string[] args) {

ICar basicCar = new BasicCar();

Console.WriteLine($"{basicCar.GetDescription()} - قیمت: {basicCar.GetPrice()} تومان");

ICar luxuryCar = new SunroofDecorator(new LeatherSeatsDecorator(basicCar));

Console.WriteLine($"{luxuryCar.GetDescription()} - قیمت: {luxuryCar.GetPrice()} تومان");

}

}

**خروجی:**

ماشین پایه - قیمت: 100000 تومان

ماشین پایه، روکش چرم، سانروف - قیمت: 135000 تومان

## ****مثال کاربردی: سیستم لاگینگ با دکوراتور****

فرض کنید یک سیستم لاگینگ دارید که می‌خواهید **امکانات جدیدی** مانند **رمزنگاری** یا **فشرده‌سازی** به آن اضافه کنید:

### ****1. اینترفیس اصلی Logger****

public interface ILogger {

void Log(string message);

}

### ****2. Logger پایه****

public class FileLogger : ILogger {

public void Log(string message) {

File.AppendAllText("log.txt", message + Environment.NewLine);

}

}

### ****3. دکوراتورها برای افزودن قابلیت‌های جدید****

public abstract class LoggerDecorator : ILogger

{

protected ILogger \_logger;

public LoggerDecorator(ILogger logger)

{

\_logger = logger;

}

public virtual void Log(string message) => \_logger.Log(message);

}

// دکوراتور رمزنگاری

public class EncryptLogger : LoggerDecorator

{

public EncryptLogger(ILogger logger) : base(logger) { }

public override void Log(string message)

{

string encryptedMessage = $"رمز شده: {Convert.ToBase64String(Encoding.UTF8.GetBytes(message))}";

\_logger.Log(encryptedMessage);

}

}

// دکوراتور فشرده‌سازی

public class CompressLogger : LoggerDecorator

{

public CompressLogger(ILogger logger) : base(logger) { }

public override void Log(string message)

{

string compressedMessage = $"فشرده شده: {message.Substring(0, Math.Min(10, message.Length))}...";

\_logger.Log(compressedMessage);

}

}

### ****4. استفاده در کلاینت****

class Program

{

static void Main(string[] args)

{

ILogger logger = new FileLogger();

logger.Log("اولین لاگ ساده");

ILogger encryptedLogger = new EncryptLogger(logger);

encryptedLogger.Log("لاگ محرمانه");

ILogger compressedEncryptedLogger = new CompressLogger(new EncryptLogger(logger));

compressedEncryptedLogger.Log("لاگ فشرده و رمز شده");

}

}

## ****مزایای الگوی Decorator****

✅ **انعطاف‌پذیری بالا**: امکان ترکیب ویژگی‌ها به صورت پویا.  
✅ **جلوگیری از انفجار زیرکلاس‌ها**: نیازی به ایجاد کلاس‌های ترکیبی مثل CarWithLeatherAndSunroof نیست.  
✅ **پیروی از Open/Closed Principle**: بدون تغییر کد موجود، قابلیت‌های جدید اضافه می‌شود.

## ****معایب الگوی Decorator****

❌ **پیچیدگی افزایش می‌یابد**: اگر بیش از حد استفاده شود، دنبال کردن کد سخت می‌شود.  
❌ **اشیا زیاد می‌شوند**: هر دکوراتور یک لایه جدید اضافه می‌کند.

## ****تفاوت Decorator با الگوهای مشابه****

| **الگو** | **تفاوت اصلی** |
| --- | --- |
| **Decorator** | **افزودن** مسئولیت‌ها به صورت پویا (ترکیب). |
| **Adapter** | **تبدیل** یک رابط به رابط دیگر. |
| **Composite** | ایجاد سلسله‌مراتب درختی از اشیا. |
| **Strategy** | تغییر **الگوریتم** یک عملیات، نه اضافه کردن قابلیت. |

## ****کاربردهای واقعی در #C و .NET****

1. **System.IO (Stream Decorators)**
   * BufferedStream, CryptoStream, GZipStream روی FileStream سوار می‌شوند.
2. **ASP.NET Core Middleware**
   * هر Middleware یک لایه Decorator روی Pipeline می‌سازد.
3. **WPF/Silverlight Controls**
   * افزودن قابلیت‌های جدید به کنترل‌های پایه.

## ****جمع‌بندی****

✅ **چه زمانی از Decorator استفاده کنیم؟**

* وقتی می‌خواهید **قابلیت‌ها را به صورت پویا** اضافه/حذف کنید.
* وقتی **ارث‌بری** گزینه مناسبی نیست (ترکیب بهتر است).

❌ **چه زمانی استفاده نکنیم؟**

* اگر تغییرات **ثابت** هستند (ارث‌بری ساده‌تر است).
* اگر **کارایی** بسیار مهم است (هر دکوراتور یک لایه اضافه می‌کند).

**مثال نهایی:**

// ایجاد یک ماشین با تمام آپشن‌ها

ICar superCar = new SunroofDecorator(

new LeatherSeatsDecorator(

new TurboEngineDecorator(

new BasicCar())));

این الگو یکی از **پرکاربردترین** الگوها در توسعه انعطاف‌پذیر نرم‌افزار است! 🚀

-------------------------------------------------------------------------------------------------------