**الگوی Adapter درC#**

**تعریف الگوی Adapter**

الگوی **Adapter** یک الگوی **ساختاری (Structural)** است که به دو رابط (اینترفیس) ناسازگار اجازه می‌دهد تا با یکدیگر کار کنند. این الگو با تبدیل رابط یک کلاس به رابطی که کلاینت انتظار دارد، ناسازگاری بین کلاس‌ها را برطرف می‌کند.

**نام‌های دیگر:**

* Wrapper (پوشاننده(

**مشکلی که Adapter حل می‌کند**

فرض کنید:

* یک **سرویس یا کتابخانه خارجی** دارید که نمی‌توانید مستقیماً از آن در کد خود استفاده کنید، چون رابط آن با سیستم شما سازگار نیست.
* نمی‌خواهید کد اصلی را تغییر دهید، اما نیاز دارید از آن کلاس استفاده کنید.

**مثال واقعی:**

* اتصال یک درایور قدیمی USB به پورت جدید Type-C با استفاده از یک مبدل (Adapter).
* استفاده از یک کتابخانه پرداخت قدیمی در یک سیستم جدید.

**انواع الگوی Adapter**

* 1. **Adapter شیء-محور (Object Adapter)**

از **کمپوزیسیون (ترکیب)** برای تطبیق رابط‌ها استفاده می‌کند.

* 1. **2. Adapter کلاس-محور (Class Adapter)**

از **ارث‌بری چندگانه** (در زبان‌هایی مثل #C که از آن پشتیبانی نمی‌کند، با اینترفیس‌ها شبیه‌سازی می‌شود.)

**ساختار الگوی Adapter در C#**

### ****رابط هدف (Target Interface)****

اینترفیسی که کلاینت انتظار دارد:

public interface ITarget {

string GetRequest();

}

### ****سرویس ناسازگار (Adaptee)****

کلاسی که باید تطبیق داده شود:

public class Adaptee {

public string GetSpecificRequest() {

return "درخواست خاص از Adaptee";

}

}

### ****) Adapterمبدل(****

رابط ناسازگار را به رابط مورد انتظار تبدیل می‌کند:

public class Adapter : ITarget {

private readonly Adaptee \_adaptee;

public Adapter(Adaptee adaptee) {

\_adaptee = adaptee;

}

public string GetRequest() {

return $"مبدل شده: {\_adaptee.GetSpecificRequest()}";

}

}

### ****کلاینت (Client)****

از طریق ITarget با سرویس کار می‌کند:

class Program {

static void Main(string[] args) {

Adaptee adaptee = new Adaptee();

ITarget adapter = new Adapter(adaptee);

Console.WriteLine(adapter.GetRequest()); // خروجی: "مبدل شده: درخواست خاص از Adaptee"

}

}

## ****مثال کاربردی: سیستم پرداخت قدیمی به جدید****

فرض کنید یک **درگاه پرداخت قدیمی** دارید که باید در یک **سیستم جدید** استفاده شود:

### ****اینترفیس جدید (Target)****

public interface INewPaymentSystem {

void ProcessPayment(decimal amount);

}

### ****سیستم پرداخت قدیمی (Adaptee)****

public class LegacyPaymentSystem {

public void MakePayment(string amountInDollars) {

Console.WriteLine($"پرداخت قدیمی: ${amountInDollars}");

}

}

### ****Adapter برای تبدیل سیستم قدیمی به جدید****

public class LegacyPaymentAdapter : INewPaymentSystem {

private readonly LegacyPaymentSystem \_legacySystem;

public LegacyPaymentAdapter(LegacyPaymentSystem legacySystem) {

\_legacySystem = legacySystem;

}

public void ProcessPayment(decimal amount) {

string amountInDollars = amount.ToString("0.00");

\_legacySystem.MakePayment(amountInDollars);

}

}

### ****استفاده در کلاینت****

class Program {

static void Main(string[] args) {

LegacyPaymentSystem legacySystem = new LegacyPaymentSystem();

INewPaymentSystem adapter = new LegacyPaymentAdapter(legacySystem);

adapter.ProcessPayment(50.50m); // خروجی: "پرداخت قدیمی: $50.50"

}

}

**مزایای الگوی Adapter**

✅ **قابلیت استفاده مجدد**: اجازه می‌دهد کلاس‌های ناسازگار در سیستم‌های جدید استفاده شوند.  
✅ **انعطاف‌پذیری**: بدون تغییر کد اصلی، قابلیت ادغام فراهم می‌شود.  
✅ **تست‌پذیری**: می‌توان از Mock Adapter برای تست استفاده کرد.

**معایب الگوی Adapter**

❌ **پیچیدگی افزایش می‌یابد**: اگر بیش از حد استفاده شود، کد را شلوغ می‌کند.  
❌ **ممکن است عملکرد را کمی کاهش دهد** (به دلیل اضافه شدن یک لایه واسط).

**تفاوت Adapter با الگوهای مشابه**

| **الگو** | **تفاوت اصلی** |
| --- | --- |
| **Adapter** | دو رابط ناسازگار را به هم وصل می‌کند. |
| **Facade** | یک رابط ساده‌تر برای یک سیستم پیچیده ارائه می‌دهد. |
| **Decorator** | قابلیت‌های جدیدی به شیء اضافه می‌کند، بدون تغییر رابط اصلی. |
| **Bridge** | جداسازی انتزاع از پیاده‌سازی برای انعطاف‌پذیری بیشتر. |

**کاربردهای واقعی در #C و .NET**

1. **ADO.NET Data Adapters** (برای اتصال دیتابیس‌های مختلف).
2. **WCF Channel Adapters** (برای ارتباط بین سرویس‌ها).
3. **مبدل‌های API** (مثلاً تبدیل JSON قدیمی به مدل جدید).

**جمع‌بندی**

الگوی **Adapter** یک راه‌حل انعطاف‌پذیر برای **ادغام سیستم‌های قدیمی با جدید** بدون تغییر کد منبع است. این الگو در مواردی که **نمی‌خواهید یا نمی‌توانید 0 Adapter استفاده کنیم؟**

* وقتی نیاز دارید یک کتابخانه یا کامپوننت خارجی را در سیستم خود استفاده کنید، اما رابط آن با سیستم شما سازگار نیست.
* وقتی می‌خواهید یک لایه میانی برای تطبیق رابط‌ها ایجاد کنید.

**❌ چه زمانی استفاده نکنیم؟**

* اگر می‌توانید مستقیماً از رابط اصلی استفاده کنید (نیازی به Adapter نیست).
* اگر سیستم خیلی ساده است و Adapter فقط پیچیدگی اضافه می‌کند.

این الگو یکی از **پرکاربردترین** الگوهای طراحی در توسعه نرم‌افزار است، مخصوصاً هنگام کار با **کدهای Legacy** یا **کتابخانه‌های شخص‌ثالث**.