



Adapter Design Pattern (C#)



Rahul Dhammy, 16 May 2014



4.95 (28 votes)

Rate this:



The article explains Adapter pattern, implemented in C#.

Introduction

In this article I would try to explore a very useful and commonly used design pattern- "**The Adapter Pattern**". I will cite my examples using C# language.

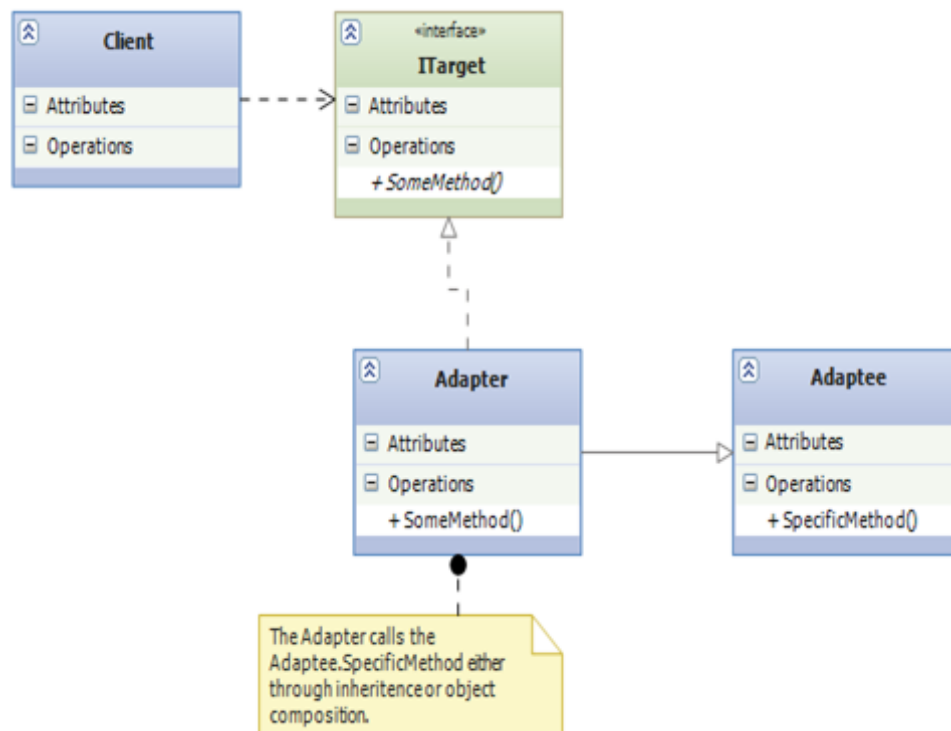
Background

Adapter pattern is placed under the category of structural design pattern. "**Adapter**" as the name suggests is the object which lets two mutually incompatible interfaces communicate with each other. But why we want "someone" to adapt to something? You will get an answer if you answer this simple thing- Your laptop charger which you bought in US has flattish pins which easily gets hooked into electrical sockets in US. But when you travel to European countries you may have round holes in the electrical sockets. What do you do then?-Simple buy socket adapters/converters for that.



A electrical socket adapter which would make electrical devices made in US, fit in European sockets.

We use Adapters when incompatible interfaces are involved. Our client object wants to call a method but it is not able to because the interface which our client object can use, is not available with the code which our client object wants to call. Based on what an adapter does the **adapter design pattern** is also called **wrapper pattern**, **translator pattern**. Let's look at the various participants (objects, interfaces) involved in an adapter pattern.



ITarget: This is the interface which is used by the client to achieve functionality.<o:p>

Adaptee: This is the functionality which the client desires but its interface is not compatible with the client.<o:p>

Client: This is the class which wants to achieve some functionality by using the adaptee's code.

Adapter: This is the class which would implement ITarget and would call the Adaptee code which the client wants to call.

Using the code

Now let's look at an example to see how adapter pattern works. Imagine an **online shopping portal** which displays the products for selling on its home page. These products are coming from a **third party vendor** with which the portal has tied hands to sell products. The third party vendor already has an inventory system in place which can give the list of products it is selling. There is no interface available to online shopping portal with which the portal can call the third party vendor's inventor code.<o:p>

Since we want to call the third party vendor's code which is incompatible with client (online shopping portal) code we can apply "Adapter design pattern" here. Let's first fit in various participants of an adapter pattern in our scenario.<o:p>

ITarget: Method which the online shopping portal calls to get the list of products. Here getting the list of products is the functionality which this portal wants to achieve and this request has been encapsulated in this interface. In short- Functionality to achieve is exposed through this interface.<o:p>

Adaptee: The third party vendor's code which gives us the list of products.<o:p>

Adapter: The wrapper which would implement ITarget and would call third party vendor's code.<o:p>

Client: The online shopping portal code which gets the list of products and then displays them.<o:p>

Let's have a look at the code demonstrating adapter pattern depicting the above scenario then we would revisit the various classes to map them to the adapter pattern participant's.

Hide Shrink ▲ Copy Code

```
interface ITarget
{
    List<string> GetProducts();
}

public class VendorAdaptee
{
    public List<string> GetListOfProducts()
    {
        List<string> products = new List<string>();
        products.Add("Gaming Consoles");
        products.Add("Television");
        products.Add("Books");
    }
}
```

```

        products.Add("Musical Instruments");
        return products;
    }
}

class VendorAdapter:ITarget
{
    public List<string> GetProducts()
    {
        VendorAdaptee adaptee = new VendorAdaptee();
        return adaptee.GetListOfProducts();
    }
}

class ShoppingPortalClient
{
    static void Main(string[] args)
    {
        ITarget adapter = new VendorAdapter();
        foreach (string product in adapter.GetProducts())
        {
            Console.WriteLine(product);
        }
        Console.ReadLine();
    }
}

```

In the above code the participants are mapped as: ITarget: interface ITarget Adapter: class VendorAdapter, implementing the ITarget interface and acting as a wrapper/link between VendorAdaptee and ShoppingPortalClient. Adaptee: class VendorAdaptee, this is the code which ShoppingPortalClient is interested in calling. Client: class ShoppingPortalClient, the client wants to call the code of VendorAdaptee. The code above is self-explanatory, the client which has access to the interface ITarget wants to call the method VendorAdaptee.GetListOfProduct() but since VendorAdaptee does not have an ITarget interface there was a need to create an adapter VendorAdapter. The VendorAdapter implements ITarget interface and calls the method of adaptee.

Based on how the adapter calls the adaptee it can be named as Class Adapter i.e. when the adapter uses class inheritance to call the Adaptee code. It is called Object Adapter when it uses the object composition to call the adaptee code. For example the VendorAdapter shown above is an object adapter because it creates an instance of (object composition) VendorAdaptee.

```
class VendorAdapter: ITarget
{
    public List<string> GetProducts()
    {
        VendorAdaptee adaptee = new VendorAdaptee();
        return adaptee.GetListOfProducts();
    }
}
```

This is an Object Adapter as the adapter uses object composition.

```
class VendorAdapter: VendorAdaptee, ITarget
{
    public List<string> GetProducts()
    {
        return GetListOfProducts();
    }
}
```

This is a Class Adapter as the adapter uses inheritance.

Points of Interest

The Adapter design pattern is easy to implement and ensures calling the existing code which was otherwise difficult because their interfaces being incompatible. It is quite common when legacy code has to be called.

History

Version 1 (16/5/2014)

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)


Share



About the Author



Rahul Dhammy

Team Leader
United States 

I am a software developer by profession presently working as an Lead Software Engineer in New York City. My passion is technology. I like to learn new things every day and improve my skills in terms of technology. I feel software development can be made very interesting if we inculcate the habit of "beautiful thinking".

You may also be interested in...



[Adapter Design Pattern](#)



[Window Tabs \(WndTabs\) Add-In for DevStudio](#)

[Adapter Design Pattern in C#](#)

[Introduction to D3DImage](#)



SAPrefs - Netscape-like Preferences Dialog



OLE DB - First steps

Comments and Discussions

You must [Sign In](#) to use this message board.

Search Comments



Spacing Tight Layout Thread View Per page 50 Update

First Prev Next

not clear	Member 3693701	6-Mar-17 10:04	2
My vote of 5	Humayun Kabir Mamun	6-Nov-14 2:06	2
See MSDN for more detailed explanation - http://msdn.microsoft.com/en-us/library/orm-9780596527730-01-04.aspx	amit trivedi	6-Aug-14 5:13	1
My vote of 3	CatchExAs	18-May-14 8:32	1
My vote of 5	Volynsky Alex	18-May-14 5:58	3
What will happen for different product class for adaptee and client?	Shirajul Mamun	18-May-14 2:52	3
My vote of 5	Sunam39	17-May-14 14:13	1
My vote of 5	Kannan.Ramjalwar	17-May-14 1:24	1

Last Visit: 31-Dec-99 19:00 Last Update: 8-Nov-17 6:35

[Refresh](#)

1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

