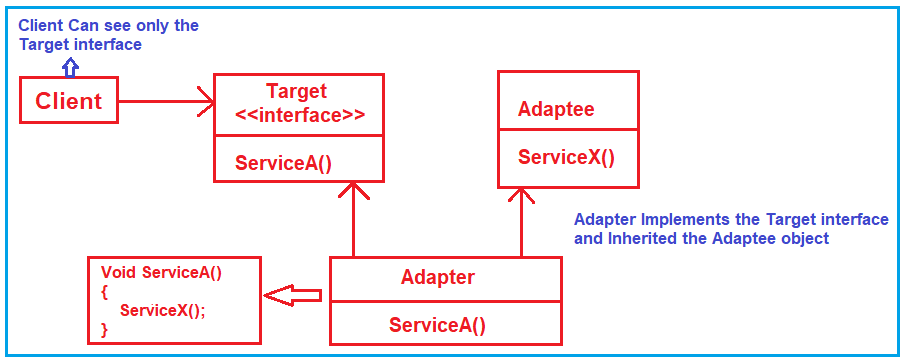
**Class Adapter Design Pattern in C#:**

This is another approach to implement the Adapter Design Pattern in C#. In this approach, the Adapter calls the methods inherited from the Adaptee class. Before implementing the same example using the Class Adapter Design Pattern, let us first understand the class diagram of the Class Adapter Design Pattern. Please have a look at the following image.



The class diagram is the same as the object adapter class diagram. The only difference is that the Adapter class now implements the Target interface and inherited the Adaptee class. In the case of the Object Adapter pattern, the adapter has a reference to the Adaptee object, and using that reference it will call the adaptee methods. But in the case of the Class Adapter Pattern, the adapter will call the inherited method of the Adaptee class.

**Implementation of Class Adapter Design Pattern in C#:**

Let us implement the previous example using the Class Adapter Design Pattern in C# step by step. The implementation is exactly the same as the object adapter implementation. The only difference is in the EmployeeAdapter class.

**Modifying the EmployeeAdapter:**

Please modify the EmployeeAdapter class as shown below. Now, the EmployeeAdapter class inherited the Adaptee i.e. ThirdPartyBillingSystem class, and implements the ITarget interface.

**using** *System;*

**using** *System.Collections.Generic;*

**namespace** *AdapterDesignPattern*

**{**

**public** **class** EmployeeAdapter : ThirdPartyBillingSystem, ITarget

**{**

**public** **void** ProcessCompanySalary**(**string**[**,**]** employeesArray**)**

**{**

string Id = **null**;

string Name = **null**;

string Designation = **null**;

string Salary = **null**;

List**<**Employee**>** listEmployee = new List**<**Employee**>()**;

**for** **(int** i = 0; i **<** employeesArray.GetLength**(**0**)**; i++**)**

**{**

**for** **(int** j = 0; j **<** employeesArray.GetLength**(**1**)**; j++**)**

**{**

**if** **(**j == 0**)**

**{**

Id = employeesArray**[**i, j**]**;

**}**

**else** **if** **(**j == 1**)**

**{**

Name = employeesArray**[**i, j**]**;

**}**

**else** **if** **(**j == 1**)**

**{**

Designation = employeesArray**[**i, j**]**;

**}**

**else**

**{**

Salary = employeesArray**[**i, j**]**;

**}**

**}**

listEmployee.Add**(**new Employee**(**Convert.ToInt32**(**Id**)**, Name, Designation, Convert.ToDecimal**(**Salary**)))**;

**}**

Console.WriteLine**(**"Adapter converted Array of Employee to List of Employee"**)**;

Console.WriteLine**(**"Then delegate to the ThirdPartyBillingSystem for processing the employee salary\n"**)**;

ProcessSalary**(**listEmployee**)**;

**}**

**}**

**}**

**Output:**

Implementation of Class Adapter Design Pattern in C#

**When to use the Object Adapter pattern and when to use the Class Adapter Pattern in C#?**

It is completely based on the situation. For example, if you have a java class and you want to make it compatible with dot net class, then you need to use the object adapter pattern and the reason is it is not possible to make inheritance. On the other hand, if both the classes are within the same project and using the same programming language and if the inheritance is possible then you need to go for Class Adapter Pattern.

**When to use the Adapter Design Pattern in the real-time application?**

We need to choose the Adapter Design Pattern in real-time applications when

1. A class needs to be reused that does not have an interface that a client requires.
2. Allow a system to use classes of another system that is incompatible with it.
3. Allow communication between a new and already existing system that is independent of each other.
4. Sometimes a toolkit or class library cannot be used because its interface is incompatible with the interface required by an application.