Adapter Pattern

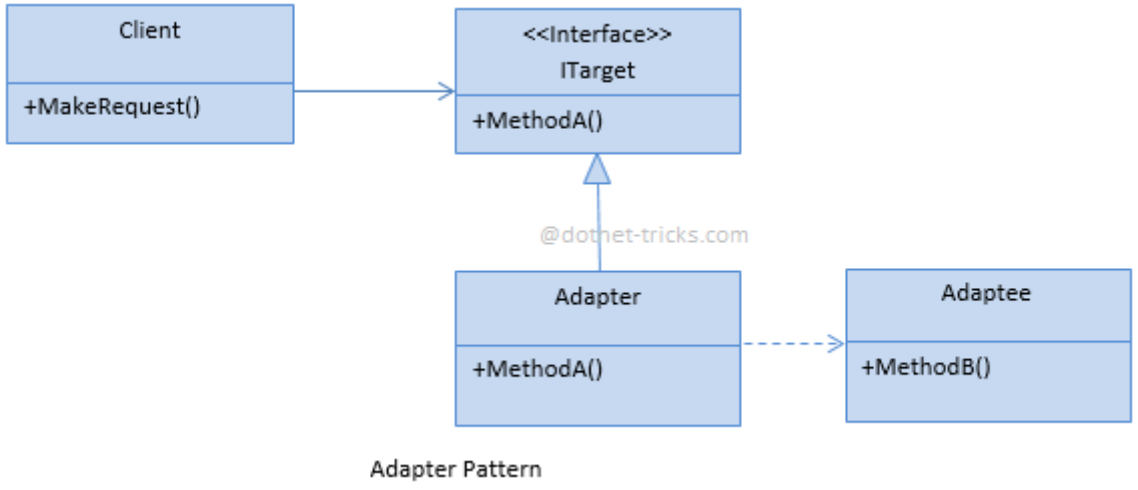
Adapter pattern falls under Structural Pattern of Gang of Four (GOF) Design Patterns in .Net. The Adapter Design pattern allows a system to use classes of another system that is incompatible with it. It is especially used for toolkits and libraries. In this article, I would like to share what is adapter pattern and how is it work?

What is Adapter Pattern

Adapter pattern acts as a bridge between two incompatible interfaces. This pattern involves a single class called adapter which is responsible for communication between two independent or incompatible interfaces.
For Example: A card reader acts as an adapter between a memory card and a laptop. You plugins the memory card into card reader and card reader into the laptop so that memory card can be read via laptop.

Adapter Pattern - UML Diagram & Implementation

The UML class diagram for the implementation of the Adapter design pattern is given below:



Adapter Pattern

The classes, interfaces, and objects in the above UML class diagram are as follows:

1. ITarget

This is an interface which is used by the client to achieve its functionality/request.

2. Adapter

This is a class which implements the ITarget interface and inherits the Adaptee class. It is responsible for communication between Client and Adaptee.

3. Adaptee

This is a class which has the functionality, required by the client. However, its interface is not compatible with the client.

4. Client

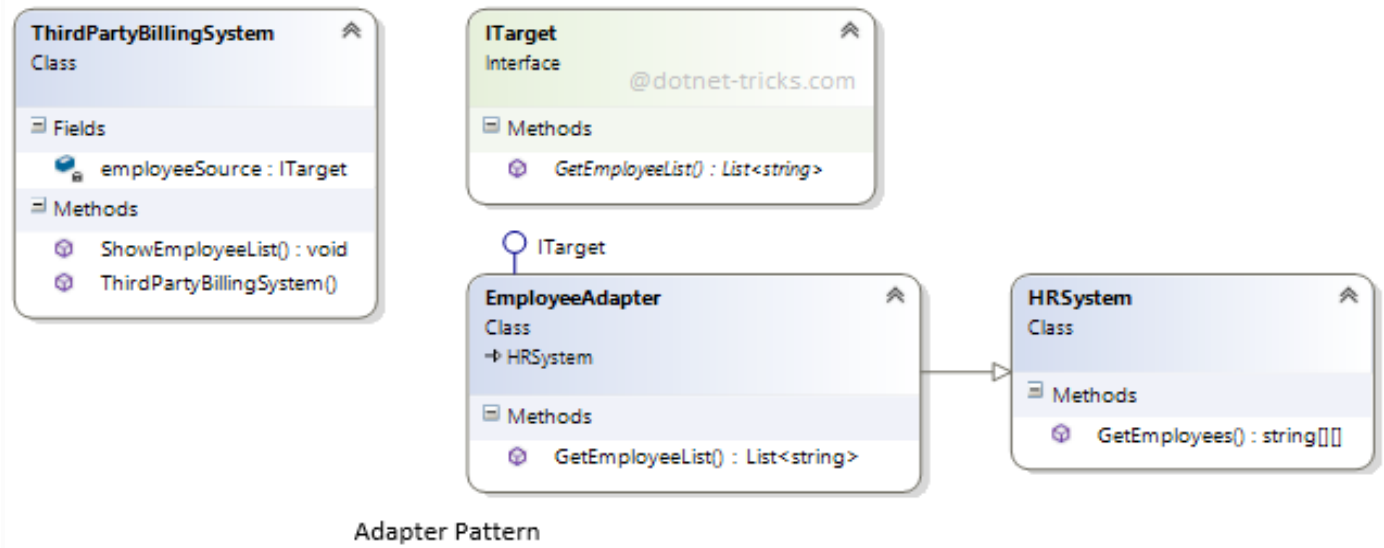
This is a class which interacts with a type that implements the ITarget interface. However, the communication class called adaptee, is not compatible with the client

C# - Implementation Code

```
public class Client{
 private ITarget target;
 public Client(ITarget target){
 this.target = target;
 }
public void MakeRequest(){
 target.MethodA();
 }
}
```

```
public interface ITarget
{
 void MethodA();
}
public class Adapter : Adaptee, ITarget{
 public void MethodA(){
 MethodB();
 }
}
public class Adaptee{
 public void MethodB(){
 Console.WriteLine("MethodB() is called");
 }
}
```

Adapter Pattern - Example



Adapter Pattern

Who is what?

The classes, interfaces, and objects in the above class diagram can be identified as follows:

1. ITraget - Target interface
2. Employee Adapter- Adapter Class
3. HR System- Adaptee Class
4. ThirdPartyBillingSystem - Client

C# - Sample Code

```
/// <summary>
/// The 'Client' class
/// </summary>
public class ThirdPartyBillingSystem
{
 private ITarget employeeSource;

 public ThirdPartyBillingSystem(ITarget employeeSource)
 {
 this.employeeSource = employeeSource;
 }

 public void ShowEmployeeList()
 {
 List<string> employee = employeeSource.GetEmployeeList();
 //To DO: Implement you business logic

 Console.WriteLine("######### Employee List ##########");
```

```
 foreach (var item in employee)
 {
 Console.Write(item);
 }


 }
}


/// <summary>
/// The 'ITarget' interface
/// </summary>
public interface ITarget
{
 List<string> GetEmployeeList();
}


/// <summary>
/// The 'Adaptee' class
/// </summary>
public class HRSystem
{
 public string[][] GetEmployees()
 {
 string[][] employees = new string[4][];


 employees[0] = new string[] { "100", "Deepak", "Team Leader" };
```

```
 employees[1] = new string[] { "101", "Rohit", "Developer" };
 employees[2] = new string[] { "102", "Gautam", "Developer" };
 employees[3] = new string[] { "103", "Dev", "Tester" };

 return employees;
 }
}

/// <summary>
/// The 'Adapter' class
/// </summary>
public class EmployeeAdapter : HRSystem, ITarget
{
 public List<string> GetEmployeeList()
 {
 List<string> employeeList = new List<string>();
 string[][] employees = GetEmployees();
 foreach (string[] employee in employees)
 {
 employeeList.Add(employee[0]);
 employeeList.Add(",");
 employeeList.Add(employee[1]);
 employeeList.Add(",");
 employeeList.Add(employee[2]);
 employeeList.Add("\n");
 }
```

```
 return employeeList;
 }
}


///
/// Adapter Design Pattern Demo
///
class Program
{
 static void Main(string[] args)
 {
 ITarget Itarget = new EmployeeAdapter();
 ThirdPartyBillingSystem client = new ThirdPartyBillingSystem(Itarget);
 client.ShowEmployeeList();


 Console.ReadKey();


 }
}
```

Adapter Pattern Demo - Output

```
########## Employee List ###########
100,Deepak,Team Leader
101,Rohit,Developer
102,Gautam,Developer
103,Dev,Tester
```

When to use it?

1. Allow a system to use classes of another system that is incompatible with it.

2. Allow communication between a new and already existing system which are independent of each other
3. Ado.Net SqlAdapter, OracleAdapter, MySqlAdapter are the best example of Adapter Pattern.

Note

1. Internally, Adapter uses Factory design pattern for creating objects. But it can also use Builder design pattern and prototype design pattern for creating a product. It completely depends upon your implementation for creating products.
2. Adapter can be used as an alternative to Facade to hide platform-specific classes.
3. When Adapter, Builder, and Prototype define a factory for creating the products, we should consider the following points :
    1. Adapter uses the factory for creating objects of several classes.
    2. Builder uses the factory for creating a complex product by using simple objects and a step by step approach.
    3. Prototype use the factory for building a product by copying an existing product.