**ADAPTER (Структурный паттерн)**

**Паттерн Адаптер (Adapter)** предназначен для преобразования интерфейса одного класса в интерфейс другого. Благодаря реализации данного паттерна мы можем использовать вместе классы с несовместимыми интерфейсами.

**Когда надо использовать Адаптер?**

Когда необходимо использовать имеющийся класс, но его интерфейс не соответствует потребностям

Когда надо использовать уже существующий класс совместно с другими классами, интерфейсы которых не совместимы

Главная задача Адаптера – реализация требуемого интерфейса и трансляция его вызовов адаптируемому объекту. Подобную ситуацию можно встретить при использовании сторонних библиотек. Они далеко не всегда предоставляют интерфейсы, которые необходимы для связи с другими объектами. При этом изменить код или добавить поддержку интерфейса не предоставляется возможным.

**Различают четыре роли, отводимые участвующим в работе шаблона объектам:**

- Адаптируемый объект (Adaptee);

- Цель (Target), определяющая требуемый интерфейс;

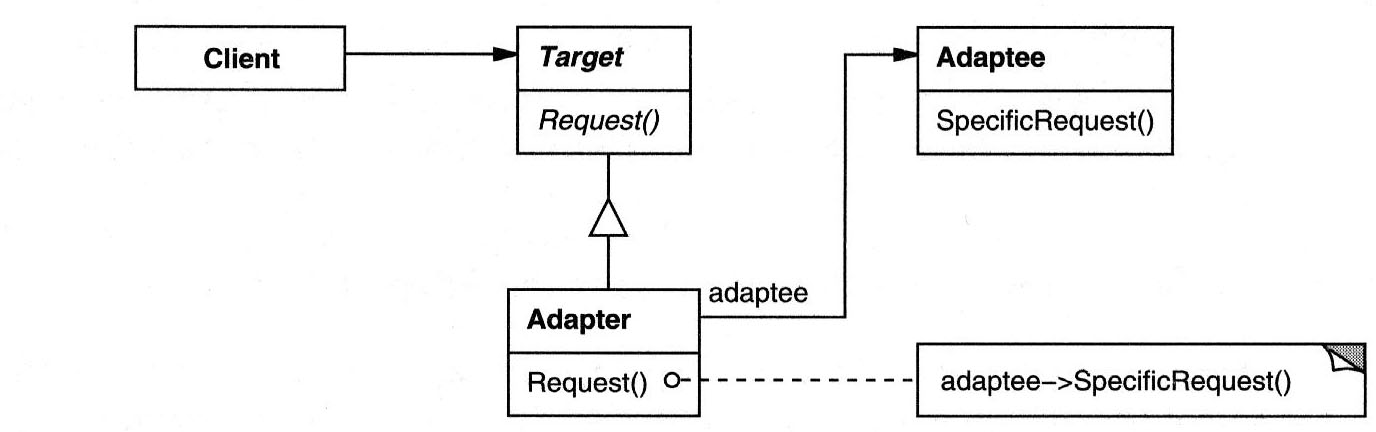
- Адаптер (Adapter);

- Клиент (Client), который умеет работать с только объектами, реализующими интерфейс цели.

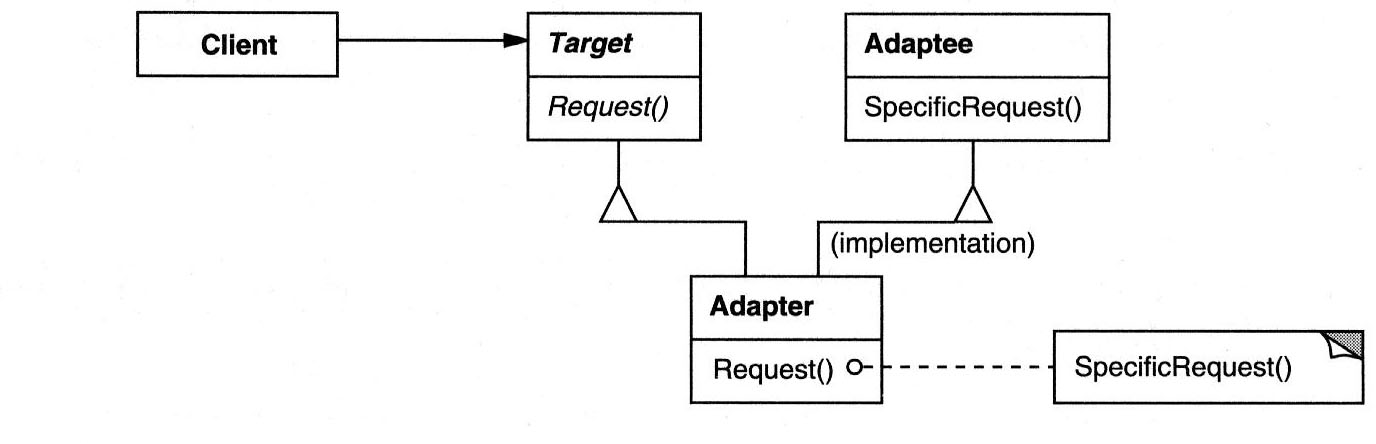
**Реализация шаблона в общем виде**

**По схеме, используемой для работы с адаптируемым объектом, выделяют два варианта:**

- Адаптер объекта – использует композицию, т.е. содержит экземпляр адаптируемого объекта.



- Адаптер класса – использует наследование от адаптируемого объекта для получения его функциональности.



Приоритетным является первый вариант, т.к. он обеспечивает меньшую связанность с адаптируемым объектом. В этом случае может даже осуществляться преобразование одного интерфейса в другой без привязки к конкретной реализации.

Но встречаются ситуации, когда требуется применение адаптера класса. Например, необходимость доступа к protected методам. В другом случае может потребоваться использовать Адаптер и вместо адаптируемого объекта.

**Пример:**

Допустим, у нас есть путешественник, который путешествует на машине. Но в какой-то момент ему приходится передвигаться по пескам пустыни, где он не может ехать на машине. Зато он может использовать для передвижения верблюда. Однако в классе путешественника использование класса верблюда не предусмотрено, поэтому нам надо использовать адаптер:

class Program **(Адаптер обьекта)**

{

    static void Main(string[] args)

    {

        // путешественник

        Driver driver = new Driver();

        // машина

        Auto auto = new Auto();

        // отправляемся в путешествие

        driver.Travel(auto);

        // встретились пески, надо использовать верблюда

        Camel camel = new Camel();

        // используем адаптер

        ITransport camelTransport = new CamelToTransportAdapter(camel);

        // продолжаем путь по пескам пустыни

        driver.Travel(camelTransport);

        Console.Read();

    }

}

interface ITransport

{

    void Drive();

}

// класс машины

class Auto : ITransport

{

    public void Drive()

    {

        Console.WriteLine("Машина едет по дороге");

    }

}

class Driver

{

    public void Travel(ITransport transport)

    {

        transport.Drive();

    }

}

// интерфейс животного

interface IAnimal

{

    void Move();

}

// класс верблюда

class Camel : IAnimal

{

    public void Move()

    {

        Console.WriteLine("Верблюд идет по пескам пустыни");

    }

}

// Адаптер от Camel к ITransport

class CamelToTransportAdapter : ITransport

{

    Camel camel;

    public CamelToTransportAdapter(Camel c)

    {

        camel = c;

    }

    public void Drive()

    {

        camel.Move();

    }

}

public class Program **(Адаптер класса)**

{

static void Main(string[] args)

{

// путешественник

Driver driver = new Driver();

// машина

Auto auto = new Auto();

// отправляемся в путешествие

driver.Travel(auto);

// используем адаптер

ITransport camelTransport = new CamelToTransportAdapter();

// продолжаем путь по пескам пустыни

driver.Travel(camelTransport);

Console.Read();

}

}

public interface ITransport

{

void Drive();

}

// класс машины

public class Auto : ITransport

{

public void Drive()

{

Console.WriteLine("Машина едет по дороге");

}

}

public class Driver

{

public void Travel(ITransport transport)

{

transport.Drive();

}

}

// интерфейс животного

public interface IAnimal

{

void Move();

}

// класс верблюда

public class Camel : IAnimal

{

public void Move()

{

Console.WriteLine("Верблюд идет по пескам пустыни");

}

}

// Адаптер от Camel к ITransport

public class CamelToTransportAdapter : Camel, ITransport

{

public void Drive()

{

Move();

}

}

## **Схожие шаблоны и их отличия**

|  |  |  |  |
| --- | --- | --- | --- |
| **Адаптер** | Изменяет интерфейс объекта не изменяя его функциональности. Может адаптировать несколько объектов к одному интерфейсу. | Позволяет повторно использовать уже существующий код. | Содержит или наследует адаптируемый объект. |
| [**Фасад**](http://andrey.moveax.ru/post/patterns-oop-structural-facade.aspx) | Объединяет группу объектов под одним специализированным интерфейсом. | Упрощает работу с группой объектов, вносит новый уровень абстракции. | Содержит или ссылается на объекты, необходимые для реализации специализированного интерфейса. |
| [**Мост**](http://andrey.moveax.ru/post/patterns-oop-structural-bridge.aspx) | Разделяет объект на абстракцию и реализацию. Используется для иерархии объектов. | Позволяет отдельно изменять (наследовать) абстракцию и реализацию, повышая гибкость системы. | Содержит объект(реализацию), который предоставляет методы для заданной абстракций и ее уточнений (наследников). |
| [**Декоратор**](http://andrey.moveax.ru/post/patterns-oop-structural-decorator.aspx) | Расширяет возможности объекта, изменяет его поведение. Поддерживает интерфейс декорируемого объекта, но может добавлять новые методы и свойства. | Дает возможность динамически менять функциональность объекта. Является альтернативой наследованию (в том числе множественному). | Содержит декорируемый объект. Возможна цепочка объектов, вызываемых последовательно. |
| [**Прокси**](http://andrey.moveax.ru/post/patterns-oop-structural-proxy.aspx) | Прозрачно замещает объект и управляет доступом к нему. Не изменяет интерфейс или поведение. | Упрощает и оптимизирует работу с объектом. Может добавлять свою функциональность, скрывая ее от клиента. | Содержит объект или ссылку на него, может управлять существованием замещенного объекта. |
| [**Компоновщик**](http://andrey.moveax.ru/post/patterns-oop-structural-composite.aspx) | Предоставляет единый интерфейс для взаимодействия с составными объектами и их частями. | Упрощает работу клиента, позволяет легко добавлять новые варианты составных объектов и их частей. | Включается в виде интерфейса в составные объекты и их части. |
| [**Приспособленец**](http://andrey.moveax.ru/post/patterns-oop-structural-flyweight.aspx) | Не ставит целью изменение интерфейса объекта. Но это может потребоваться для получения обратно данных из вынесенной части состояния. | Позволяет уменьшить число экземпляров объекта в приложении и тем самым сэкономить его ресурсы. | Выносит контекстно-зависимую часть состояния объекта вовне, заменяя несколько его экземпляров одним. |