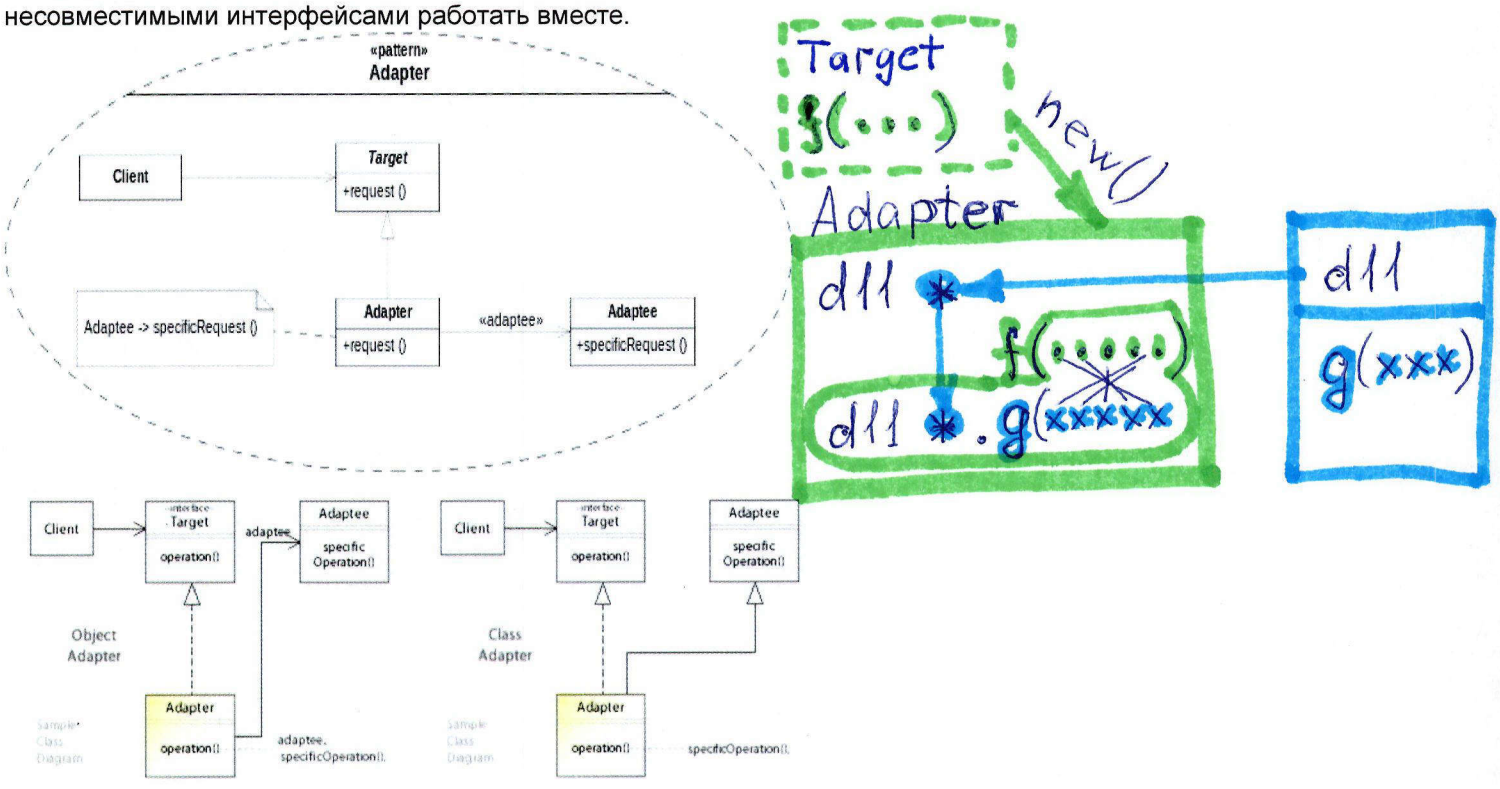
Адаптер (шаблон проектирования)

Адаптер (англ. *Adapter*) — структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс. Другими словами — это структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.



Основные характеристики

Задача

Система поддерживает требуемые данные и поведение, но имеет неподходящий интерфейс.

Способ решения

Адаптер предусматривает создание класса-оболочки[1] с требуемым интерфейсом.

Участники

Класс **Adapter** приводит интерфейс класса ***Adaptee*** в соответствие с интерфейсом класса **Target** который реализуется классом **Adapter:Target**. Это позволяет объекту ***Client*** использовать объект ***Adaptee*** посредством адаптера **Adapter:Target** так, словно он является экземпляром класса **Target**.
Таким образом ***Client*** обращается к интерфейсу **Target**, реализованному классом **Adapter:Target** - > , который перенаправляет обращение к ***Adaptee*** :
Adapter - > ***Adaptee***.

Реализация

Включение уже существующего класса в другой класс. Интерфейс включающего класса приводится в соответствие с новыми ребованиями, а вызовы его методов преобразуются в вызовы методов включённого класса.

Шаги реализации

1. Убедитесь, что у вас есть два класса с несовместимыми интерфейсами:
    - полезный сервис — служебный класс, который вы не можете изменять (он либо сторонний, либо от него зависит другой код);
        - `class Adaptee: def specific_request(self)`
    - один или несколько клиентов — существующих классов приложения, несовместимых с сервисом из-за неудобного или несовпадающего интерфейса.
        - `def request(self):`
2. Опишите клиентский интерфейс, через который классы приложения смогли бы использовать класс сервиса.
    - `class Target():`
3. Создайте класс адаптера, реализовав этот интерфейс.
    - `class Adapter(Target):`
4. Поместите в адаптер поле, которое будет хранить ссылку на объект сервиса. Обычно это поле заполняют объектом, переданным в конструктор адаптера. В случае простой адаптации этот объект можно передавать через параметры методов адаптера.
    - `adapter = Adapter(adaptee)`
5. Реализуйте все методы клиентского интерфейса в адаптере. Адаптер должен делегировать основную работу сервису.
    - `def request(self): {return self.adaptee.specific_request()}`
6. Приложение должно использовать адаптер только через клиентский интерфейс. Это позволит легко изменять и добавлять адаптеры в будущем.

```
####################################
class Adaptee:
    """
    Адаптируемый класс содержит некоторое полезное поведение, но его интерфейс
    несовместим с существующим клиентским кодом. Адаптируемый класс нуждается в
    некоторой доработке, прежде чем клиентский код сможет его использовать.
    """
    def specific_request(self) -> str:
        return ".eetpadA eht fo roivaheb laicepS"

####################################
class Target():
    """Целевой класс объявляет интерфейс, с которым может работать клиентский код."""
    def request(self) -> str:
        return "Target():"
####################################
class Adapter(Target):
    """
    Адаптер делает интерфейс Адаптируемого класса совместимым с целевым интерфейсом.
    """
    def __init__(self, adaptee: Adaptee) -> None:
        self.adaptee = adaptee
    def request(self) -> str:
        return "Adapter(Target): (TRANSLATED)" + self.adaptee.specific_request()[::-1]
```

```
#####################################
if __name__ == "__main__":
    print("Client: I can work just fine with the Target objects:")
    print(Target().request())

    adaptee = Adaptee()
    print("Client: Adaptee(): The Adaptee class has a weird interface:")
    print("Adaptee():",adaptee.specific_request())

    print("Client: Adapter(adaptee):")
    adapter = Adapter(adaptee)
    print(adapter.request())
```

```
Client: I can work just fine with the Target objects:
Target():
Client: Adaptee(): The Adaptee class has a weird interface:
Adaptee(): .eetpadA eht fo roivaheb laicepS
Client: Adapter(adaptee):
Adapter(Target): (TRANSLATED)Special behavior of the Adaptee.
```

```
using System;

////////////////////////////////////////////////////////////////////////
// Адаптируемый класс содержит некоторое полезное поведение, но его интерфейс
// несовместим с существующим клиентским кодом. Адаптируемый класс
// нуждается в некоторой доработке, прежде чем клиентский код сможет его использовать.
class Adaptee
{
public string GetSpecificRequest()
{return "Adaptee.GetSpecificRequest() == Specific request.";
}
}

////////////////////////////////////////////////////////////////////////
// Целевой класс объявляет интерфейс, с которым может работать клиентский код.
public interface ITarget
{
string GetRequest();
}
////////////////////////////////////////////////////////////////////////
// Адаптер делает интерфейс Адаптируемого класса
// совместимым с целевым интерфейсом.
class Adapter : ITarget
{
private readonly Adaptee _adaptee;
public Adapter(Adaptee adaptee)
{this._adaptee = adaptee;
}

public string GetRequest()
{return "Adapter.GetRequest() == {}".this._adaptee.GetSpecificRequest() ;
}}

////////////////////////////////////////////////////////////////////////
class Program
{
static void Main(string[] args)
{Adaptee adaptee = new Adaptee();
ITarget target = new Adapter(adaptee);
Console.WriteLine(target.GetRequest());
Console.ReadLine();
}}


/////////////////////////////////////
Adapter.GetRequest() == Adaptee.GetSpecificRequest() == Specific request.
```