Adapter

UNIFCV - PARADIGMAS DE LINGUAGEM DE PROGRAMAÇÃO

Gildo Junior

Vinicius Araujo

Introdução

Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores. Os de classes utilizam a herança para compor interfaces ou implementações, e os de objeto ao invés de compor interfaces ou implementações, eles descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade obtida pela composição de objetos provém da capacidade de mudar a composição em tempo de execução o que não é possível com a composição estática (herança de classes).

“

É um dos padrões de projeto estruturais do GoF mais simples do grupo, sendo sua responsabilidade converter uma interface de uma classe para outra interface que o cliente espera encontrar.

Case

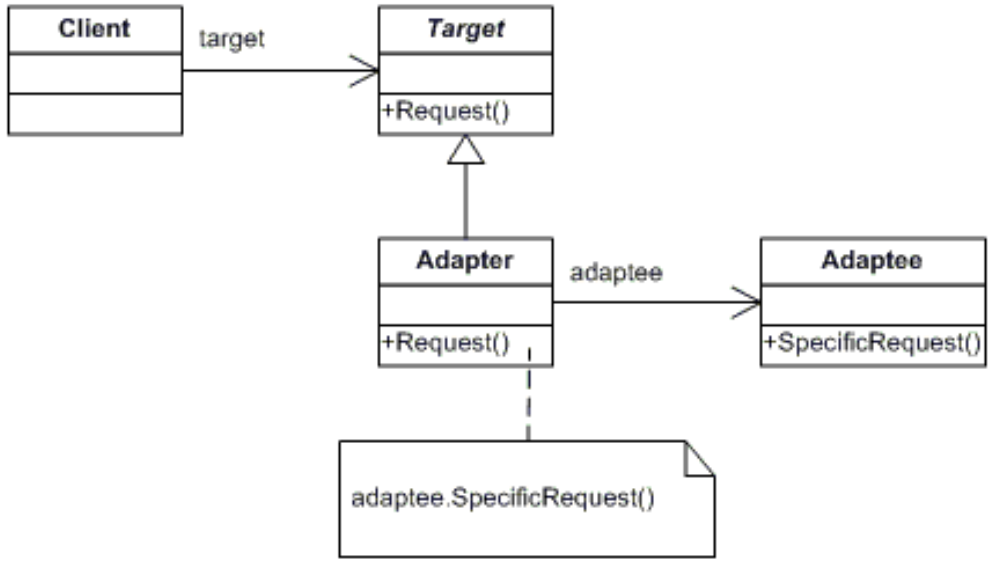
Vamos pensar de uma forma simples e com um exemplo real:

Por anos, utilizamos na construção civil um determinado padrão de plugues e tomadas, e em um determinado momento, o governo definiu um novo padrão. Com isso, novas construções eram entregues com esse novo padrão de tomadas, mas muitos de nossos aparelhos, fabricados antes da definição do novo padrão, não eram mais compatíveis com essas tomadas. Com isso, temos que utilizar adaptadores, que permitem plugar nossos aparelhos que possuem um padrão de plugue X nas tomadas que agora são fabricadas em um padrão Y.

Em um software, o Adapter faz exatamente isto, ele converte uma determinada interface ou classe para outra interface que o código espera, permitindo que módulos com interfaces, antes incompatíveis, agora possam se comunicar.

UML do pattern:

- Target (Alvo): Define a interface do domínio específico que o cliente utiliza
- Adapter (Adaptador): Adapta a interface Adaptee para a interface da classe Target
- Adaptee (Adaptada): Define uma interface existente que necessita ser adaptada
- Client (Cliente): Colabora com os objetos em conformidade com a interface Target



Connector

O Adapter é bastante utilizado em conjunto de outro pattern, o Connector. Adiantando um pouco, o conector é responsável por acessar um determinado recurso como API, socket etc. e obter uma resposta (vamos detalhar quando este padrão for abordado), enquanto o Adapter converte esta resposta, muitas vezes não compatível com a estrutura de nosso software, para uma interface ou classe que nosso software possa utilizar.

Resumo

01. Nome:

Adapter

02. Objetivo/Intenção:

Permitir que classes com interfaces incompatíveis trabalhem juntas, convertendo uma interface ou classe para outra compatível;

03. Motivação:

Muitas vezes, uma classe que poderia ser reaproveitada não é reutilizada justamente pelo fato de sua interface não corresponder à interface específica de um domínio requerida por uma aplicação;

04. Aplicabilidade:

O padrão Adapter pode ser utilizado quando se deseja utilizar uma classe existente, porém sua interface não corresponde à interface que se necessita, ou desenvolvedor quiser criar classes reutilizáveis que cooperem com classes não-relacionadas ou não-previstas, ou seja, classes que não possuem necessariamente interfaces compatíveis ou é necessário utilizar muitas subclasses existentes, porém, impossível de adaptar essas interfaces criando subclasses para cada uma. Um adaptador de objeto pode adaptar a interface de sua classe mãe;

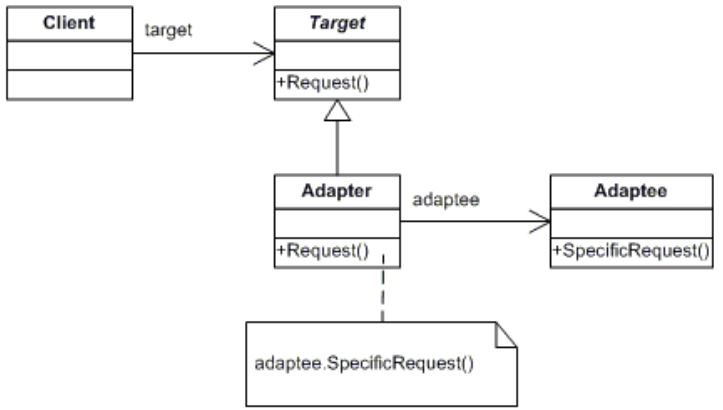
Resumo

05. Padrões relacionados:

Bridge, Decorator e Proxy

06. Estrutura:

Ao lado, temos a estrutura UML do pattern, descrita no livro do GoF. Target define a interface do domínio específico que o cliente utiliza, enquanto o Adapter adapta a interface Adaptee para a interface da classe Target. O Adaptee define uma interface existente que necessita ser adaptada e o Client colabora com os objetos em conformidade com a interface Target:



07. Uso Conhecido:

Muito utilizada quando temos um Connector, retorna dados de um ou mais serviços REST, por exemplo, e precisamos converter o JSON de retorno para uma classe específica que nosso software possa utilizar.

08. Consequências:

Cada adaptador de classes e de objetos tem diferentes soluções de compromisso. Um adaptador de classe adapta a classe Adaptee a Target através do uso efetivo de uma classe Adapter concreta. Em consequência disso, um adaptador de classe não funcionará quando quisermos adaptar uma classe e todas as suas subclasses. Permite que a classe Adapter substitua algum comportamento da classe Adaptee, uma vez que Adapter é uma subclasse de Adaptee. Já um adaptador de objeto permite a um único Adapter trabalhar com muitos Adaptees, ou seja, o Adaptee em si e todas as suas subclasses (caso existam), e torna mais difícil redefinir um comportamento de uma classe Adaptee. Ele exigirá a criação de subclasses de Adaptee e fará com que a classe Adapter referencie a subclasse, ao invés da classe Adaptee em si;

Implementação C#:

```
namespace PadraoAdapter
{
    public class EntradaP10
    {
        //Solicitação Especifica
        public void conectarEntradaP10()
        {
            Console.Write("Conectado na entrada P10");
        }
    }

    //Target
    public class EntradaXLR
    {
        //Solicitação
        public void conectarEntradaXLR()
        {
            Console.Write("Conectado na entrada XLR");
        }
    }
```

```
    //Adapter
    public class AdapterEntrada : EntradaXLR
    {
        private EntradaP10 entradaP10;

        public AdapterEntrada(EntradaP10 entradaP10)
        {
            this.entradaP10 = entradaP10;
        }

        //Solicitação
        public void conectarEntradaXLR()
        {
            entradaP10.conectarEntradaP10();
        }
    }

    public class MesaDeSom
    {
        static void Main(string[] args)
        {
            EntradaP10 p10 = new EntradaP10();
            AdapterEntrada adaptador = new AdapterEntrada(p10);
            adaptador.conectarEntradaXLR();
        }
    }
```

Referência:

https://refactoring.guru/pt-br/design-patterns/adapter

https://pt.wikipedia.org/wiki/Adapter

https://www.devmedia.com.br/padrao-de-pr
adapter-em-java/26467

https://imasters.com.br/desenvolvimento/arquitet
desenvolvimento-de-software-parte-7-adapter

https://pt.wikipedia.org/wiki/Padr%C3%A3o_de_projeto
_de_software

Código do projeto

vabarboza/Adapter: Trabalho Adapter - UniFCV (github.com)