Padrões de design: Adapter

Arthur
Djalma
Leonardo

Padrão Adapter

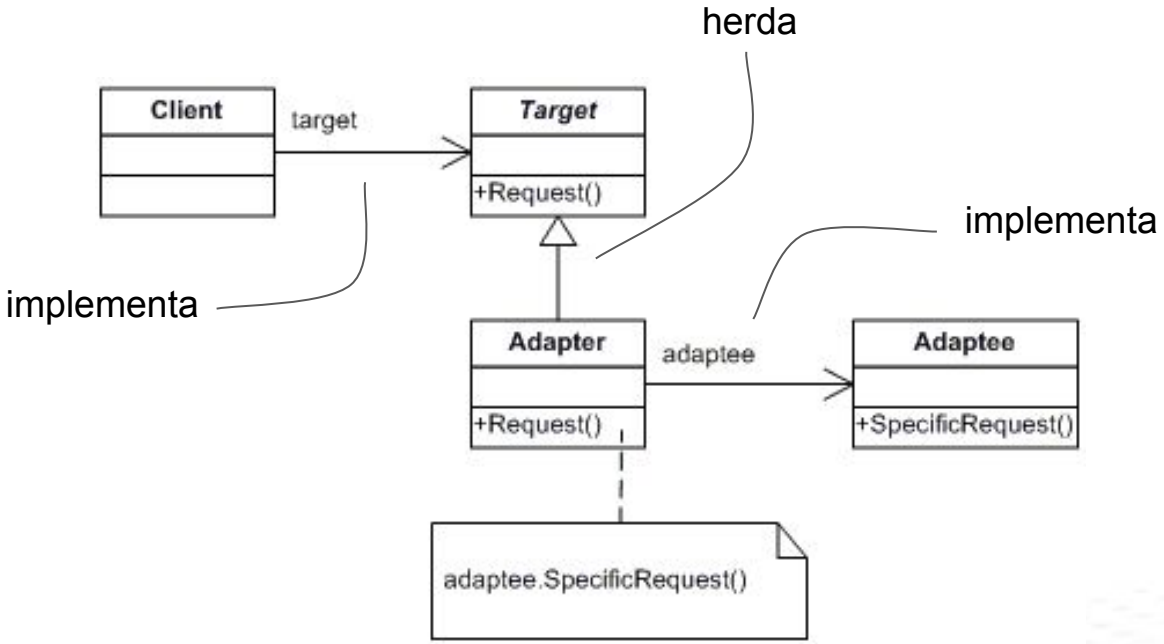
Um padrão adapter atua como um conector entre duas interfaces incompatíveis que não podem ser conectadas diretamente. Um adapter envolve uma classe existente com uma nova interface para que se torne compatível com a interface do cliente.

O principal motivo por trás do uso desse padrão é converter uma interface existente em outra interface que o cliente espera ou que pode ser utilizada por uma atualização do software ou produto.

Propósito: estrutural

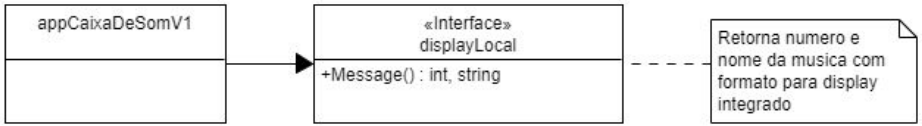
O Adapter é um padrão de projeto estrutural que permite objetos com interfaces incompatíveis colaborarem entre si, ou seja, permite que você crie uma classe de meio termo que serve como um tradutor entre seu código e a classe antiga, uma classe de terceiros, ou qualquer outra classe com uma interface estranha.

Diagrama genérico



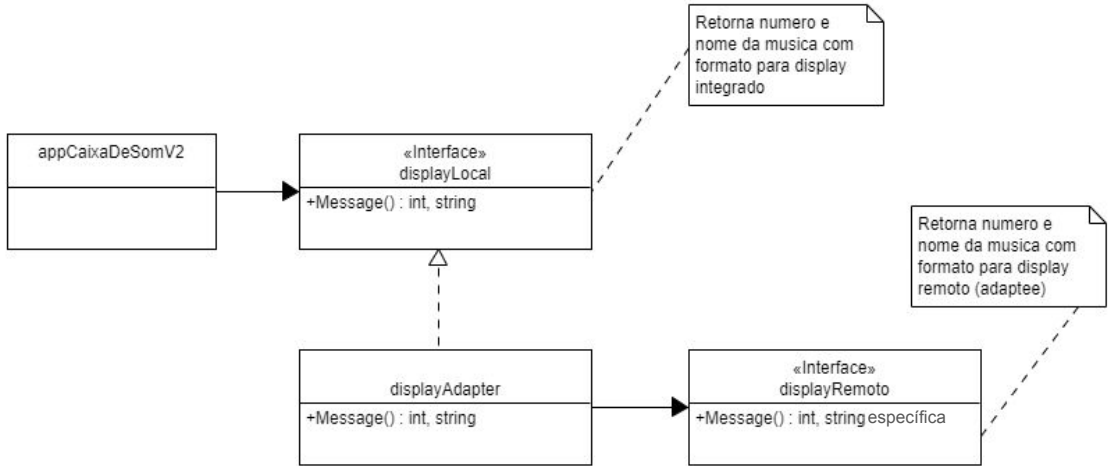
Problema

Uma empresa produz dispositivos de reprodução de áudio (caixas de som) que possuem um display integrado que informa o título do áudio que está sendo reproduzido (CaixaV1)



Porém, em um certo momento a fabricante tem a ideia de criar uma caixa de som V2, mais tecnológica que possui um display que funciona de maneira separada, um display remoto com conexão sem fio à caixa. Tal mudança fez com que a mensagem antes enviada ao display integrado não fosse mais reconhecida pelo display remoto devido a limitação resultante da separação dos hardwares.

Como opção para o funcionamento da caixa V2, o dev poderia recriar o código desde o início para que tudo funcionasse corretamente, o que gastaria mais tempo e recursos da empresa… Ou ele poderia pegar o código já existente da versão 1 e **adaptar** a mensagem de uma maneira que a mesma pudesse ser recebida e apresentada pelo novo display, assim criando o software V2.



Consequências

Com o uso do adapter não foi necessária a criação de um novo código, apenas de uma nova classe que adaptava a interface antiga para uma nova que era compatível com o novo display.

Essa nova classe permite uma melhor reutilização do código caso seja necessária para futuras especificações.

Implementação

Linguagem de programação: C++

bibliotecas:

<iostream> - manipulação do fluxo padrão da linguagem C++

<memory> - necessária para a criação do ponteiro único (displayPtr) usando a função unique_ptr que é responsável pela comunicação entre DisplayLocal e DisplayAdapter.

Implementação

main.cpp

```
#include <iostream>
#include <memory>

using namespace std;


//localDisplay interface

class LocalDispla
{
    public:
    enum Color
    {
        Red = 0,
        Blue,
        Green,
        Yellow
    };

    virtual void Message(Color color, string message) = 0;
};


// RemoteDisplay lib
// RemoteDisplay.hpp
class RemoteDisplay

{
    public:
        void MsgRed(string message) const;
        void MsgBlue(string message) const;
        void MsgGreen(string message) const;
        void MsgYellow(string message) const;
```

MOSTRAR O CÓDIGO

Referências Bibliográficas:

- https://www.baeldung.com/java-adapter-pattern
- https://ssaurel.medium.com/implement-the-adapter-design-pattern-in-java-f9adb6a8828f
- https://jucimarjr.github.io/zelda/templates/design_pattern/adapter.html

Ferramentas utilizadas:

- Desenhar diagrama: https://www.umlet.com
- IDE C++: www.onlinegbd.com