APRESENTAÇÃO

Grupo 04 POO

Wandeson José dos Santos

Allan Jorge Alves de Arruda

Guilherme da Silva Lira

Jhonathan de souza Barbosa

Wallace Guilherme Correia Imperial

Cristian Matheus Galindo de Brito

Paulo Montenegro Campos

Paulo Emanuel Madeira de Freitas

Temas

-Bridge

-Composite

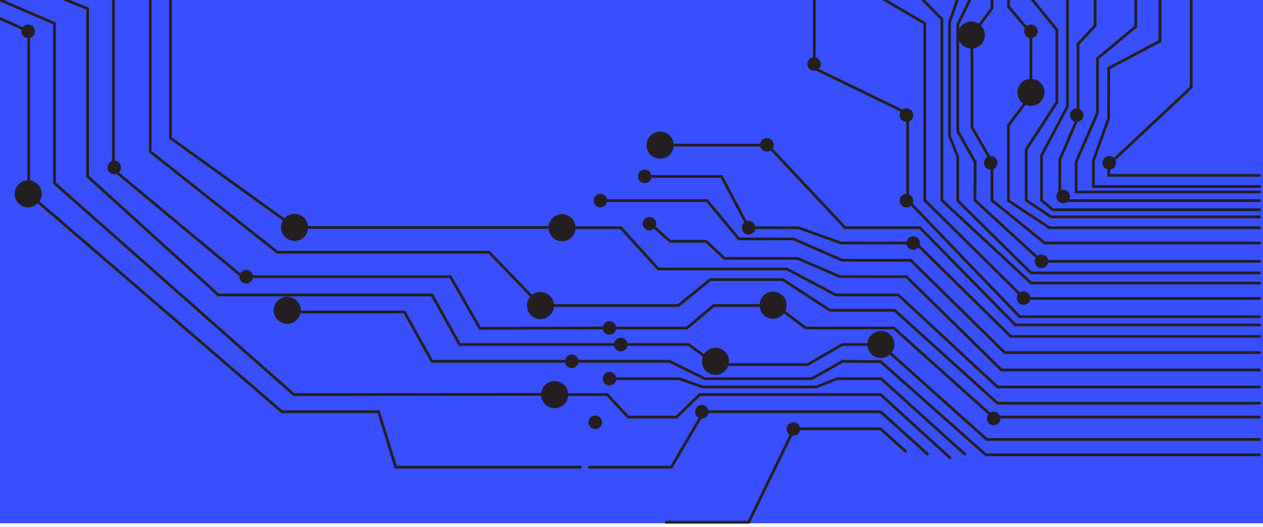
-Object Adapter

CCO 03 MA BV - POO

UNIFG

LAUREATE INTERNATIONAL UNIVERSITIES

BRIDGE



Definição

O padrão estrutural Bridge, que também pode ser chamado de handle/body, tem como intenção desacoplar uma abstração de sua implementação de tal modo que elas possam variar de maneira independente.

Motivação

-O uso de herança para permitir diversas implementações de uma abstração pode não ser flexível o suficiente.

Além disso...

-Interface e implementação ficam definitivamente ligadas, mas não podem ser usadas de modo independente.

Paulo Freitas

BRIDGE

Aplicabilidade

Quando o Bridge deve ser usado? Algumas situações são:
-Para evitar uma conexão permanente entre abstração e implementação.
-Para estender abstrações e suas implementações através de subclasses de modo independente.
-Para evitar que mudanças na implementação atinjam clientes.
-Para evitar a proliferação de classes.

Padrão UML

No padrão UML do Bridge temos:
-Abstraction: Que define a interface da abstração e mantém uma referência para o objeto Implementor.
-Refined Abstraction: Que estende a interface definida pela abstração.

Paulo Freitas

Padrão UML

-Implementor: Que define a interface para as classes de implementação. O Implementor não precisa ser igual, e geralmente é diferente da interface da abstração. Além disso, enquanto a implementação define apenas operações primitivas, a interface da abstração define operações de alto nível baseadas nessas operações primitivas.

-ConcreteImplementor: Que implementa a interface Implementor.

Paulo Freitas

Guilherme Lira

Consequências

As consequências de se usar o Bridge são:

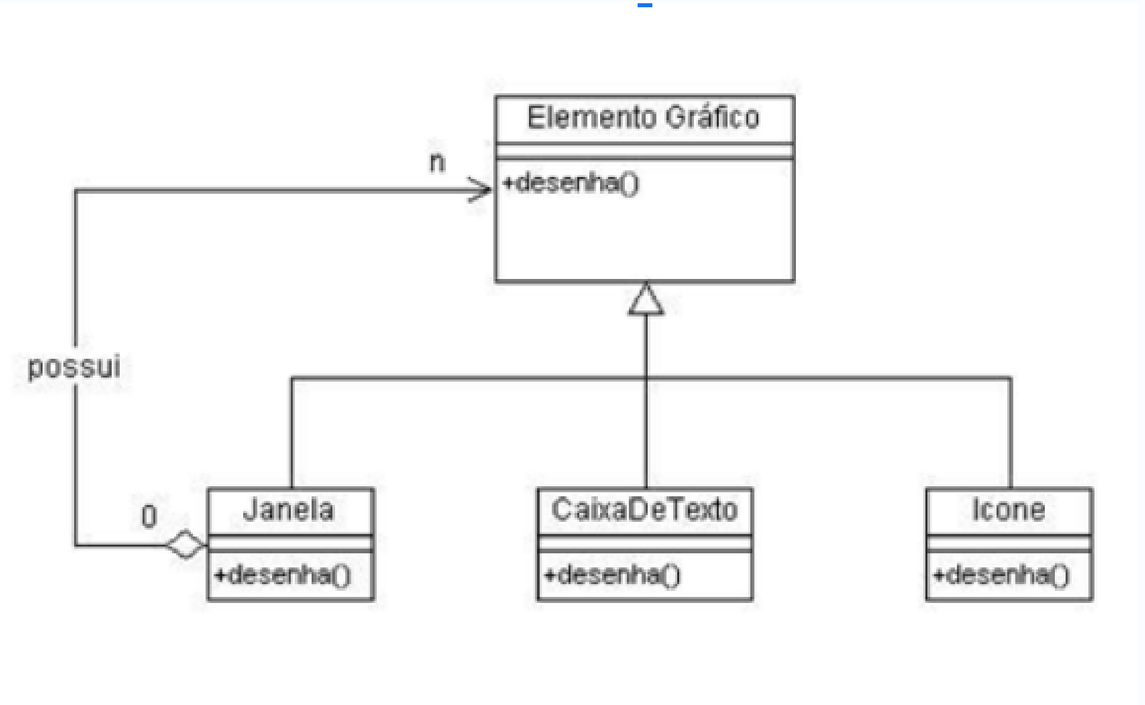
-Interface fica separada de implementação.

-Melhora as hierarquias de abstração e implementação (evolução independente).

E...

-Esconde detalhes de implementação dos clientes.

Estrutura



Exemplo de Código Parte 1

```java
//? Interface
public interface Aparelho {
    //? Metodos que iram ser usados pelas classes que implementam essa interface
    void ligar();
    void desligar();
    void aumentarVolume();
    void diminuirVolume();
}

//? Aparelho é uma interface usada para qualquer tipo de interface
public class Radio implements Aparelho {
    //? Oabstrações da classe Radio
    private boolean ligado = false;
    private int volume = 10;
    private int canal = 13;

    //? Métodos usados para ligar e desligar o Radio
    @Orverride
    public void ligar() { ligado = true; }

    @Orverride
    public void desligar() { ligado = false; }

    //? Métodos para aumenta e dimnuir o volume
    @Orverride
    public void aumentarVolume () { volume = volume + 1; }

    @Orverride
    public void diminuirVolume () { volume = volume - 1; }
}


public interface Controle {
    void ligar();
    void desligar();
    void aumentarVolume();
    void diminuirVolume();
}
```

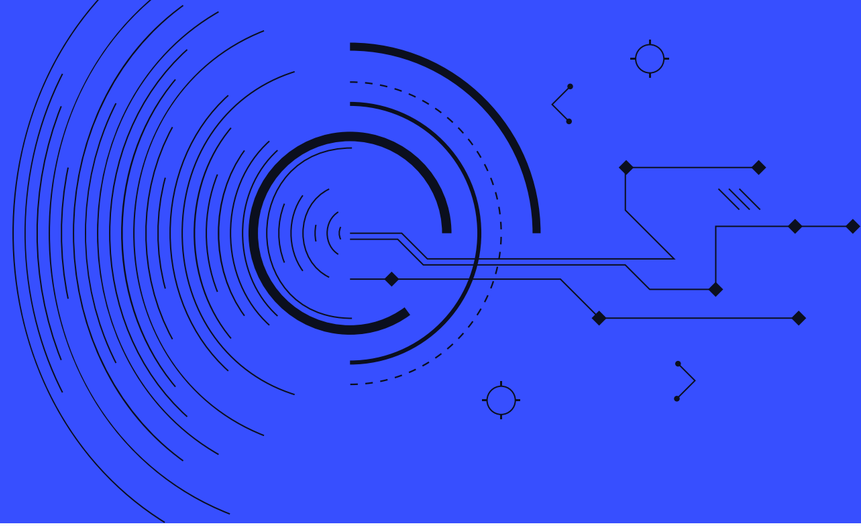
Guilherme Lira

Exemplo de Código Parte 2

```java
public class ControleBasico implements Controle {
    protected Aparelho aparelho;

    public ControleBasico(Aparelho aparelho) {
        this.aparelho = aparelho;
    }

    //? Metodos do controle remoto chamam os metodos da classe do Radio
    @Orverride
    public void ligar() {
        aparelho.ligar();
    }
    @Orverride
    public void desligar() {
        aparelho.desligar();
    }

    @Orverride
    public void aumentarVolume() {
        aparelho.aumentarVolume();
    }
    @Orverride
    public void diminuirVolume() {
        aparelho.diminuirVolume();
    }
}
```

Guilherme Lira

COMPOSITE



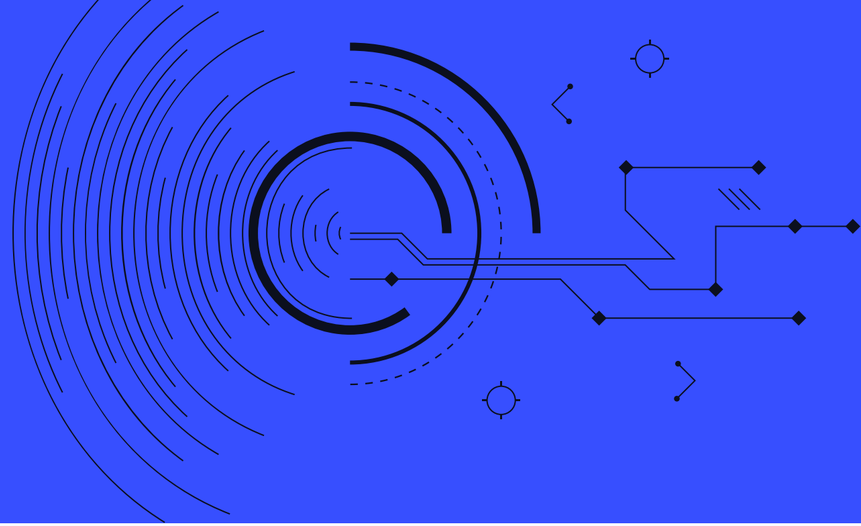
Definição

O padrão estrutural Composite (também chamado de Contêiner) permite que se componha objetos em estruturas de árvores e então trabalhe com essas estruturas como se elas fossem objetos individuais.

O Composite se tornou uma solução bastante popular para a maioria dos problemas que exigem a construção de uma estrutura em árvore. Esse padrão estrutural tem como grande recurso a capacidade de executar métodos recursivamente em toda a estrutura da árvore e resumir os resultados.

Jhonatha

COMPOSITE



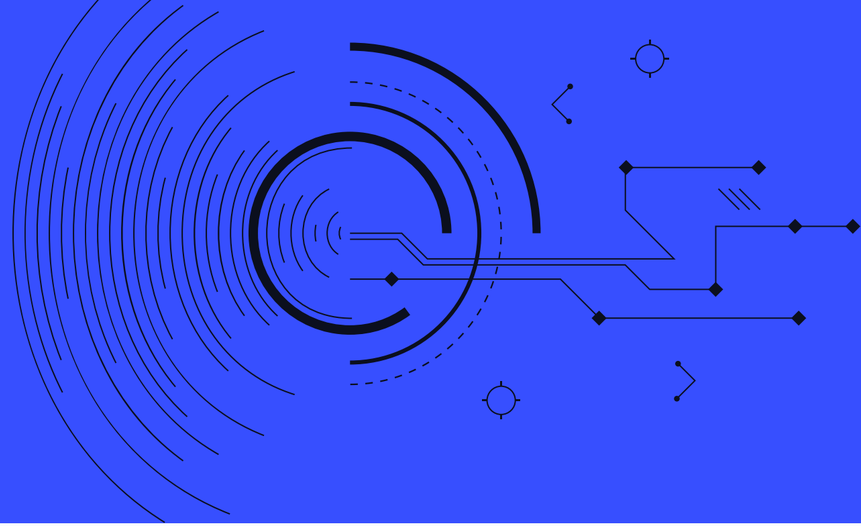
Motivação

Para abordar a motivação do Composite de maneira mais simples, é adequado trazer um exemplo da vida. Como exemplo, produtos que podem ser comprados por unidade ou em caixas. O comprador por ir ao supermercado e comprar um único produto, que irá contar com um código de barras. No entanto, o comprador pode comprar uma caixa desse mesmo produto, que por sua vez, também contará com um código de barras.

O supermercado trata uma estrutura inteira de objetos como um único objeto com um código de barras e um preço, porém, se o cliente quiser, também pode obter um único produto de dentro da caixa e realizar a compra da mesma forma. Ambos, caixa e produto, possuem seus próprios códigos de barras.

Jhonatha

COMPOSITE

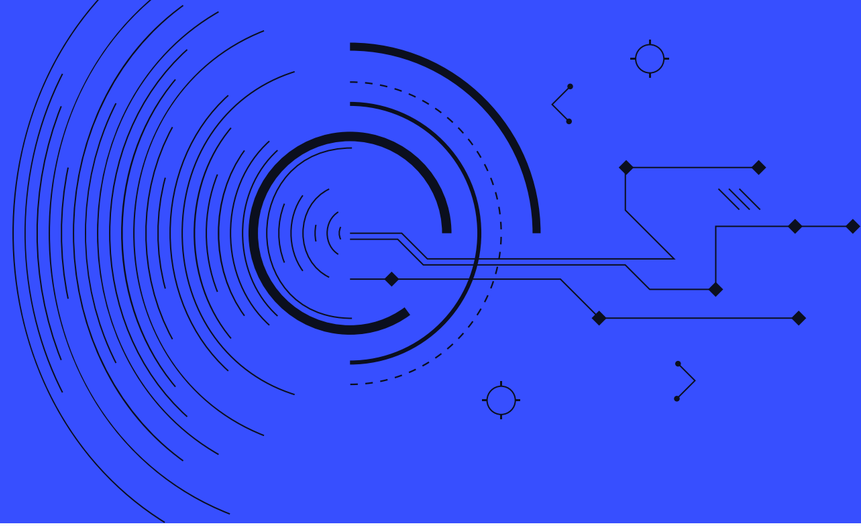


Motivação

O Composite é exatamente isso: pode tratar de um único objeto ou uma estrutura de objetos da mesma maneira. Isso se consegue através de uma interface em comum entre objetos compostos (caixa) e objetos folha (produtos).

Jhonatha

COMPOSITE



Aplicabilidade

Quando o Composite deve ser usado?
Segundo Rocha (2003), deve-se usar Composite sempre que for necessário tratar um determinado conjunto de objetos como um objeto individual. Além disso, Gamma et al. (2000) acrescentam que o Composite também deve ser utilizado quando se deseja determinar hierarquias partes-todo de objetos, ou seja, associar componentes a seus agregados.

Jhonatha

Padrão UML

Client: Que trabalha com todos os elementos através da interface componente. Como resultado, o Client pode trabalhar da mesma forma tanto com elementos simples como elementos complexos da árvore.

Interface Component: Que descreve operações que são comuns tanto para elementos simples como para elementos complexos da árvore.

Leaf: Que é um elemento básico de uma árvore que não tem subelementos. Geralmente, componentes Leaf acabam fazendo boa parte do verdadeiro trabalho, uma vez que não tem mais ninguém para delegá-lo. recebe chamadas do cliente através da interface do adaptador e as traduz em chamadas para o objeto encobrido do serviço em um formato que ele possa entender.

Wallace

Consequências

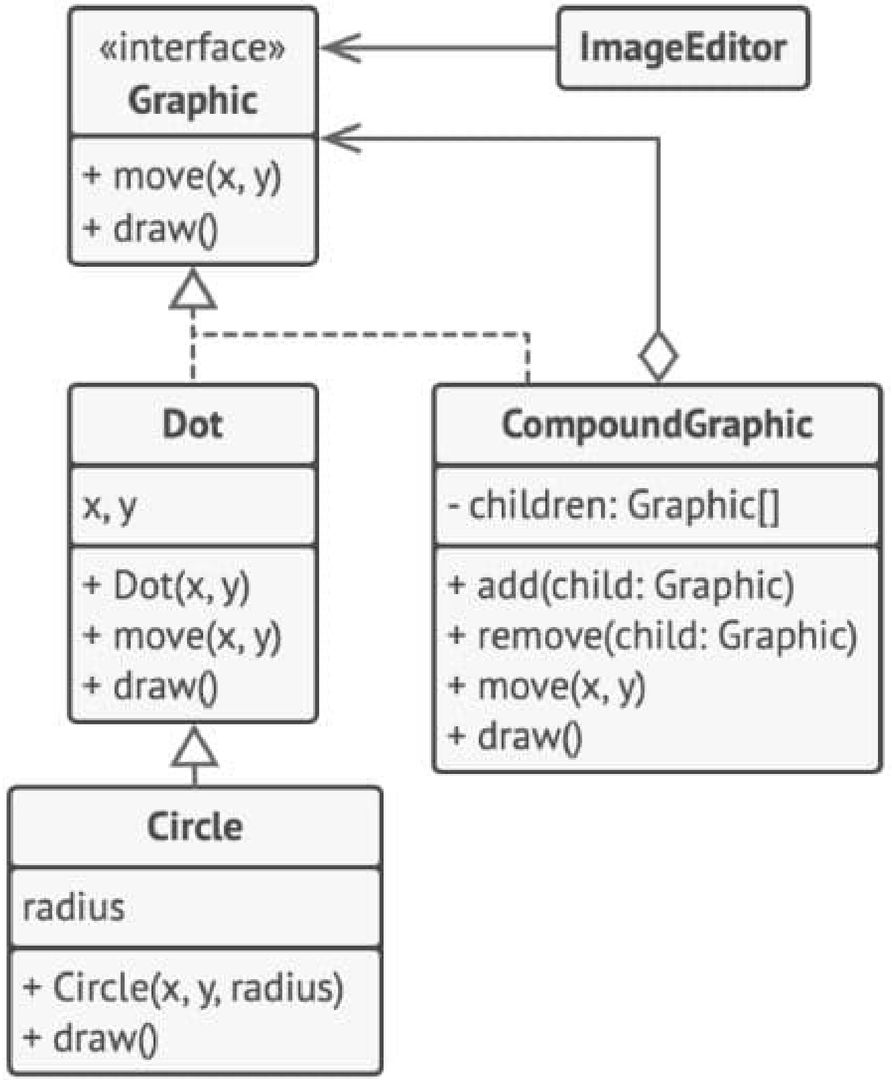
-Definição de hierarquias de classe que consistem de objetos primitivos e objetos compostos. Os objetos primitivos podem compor objetos mais complexos, os quais, por sua vez, também podem compor outros objetos, e assim por diante, recursivamente.

-Torna mais fácil de acrescentar novas espécies de componentes. Novas subclasses definidas, Composite ou Leaf, funcionam automaticamente com as estruturas existentes e o código do cliente. Os clientes não precisam ser alterados para tratar novas classes Component.

Consequências

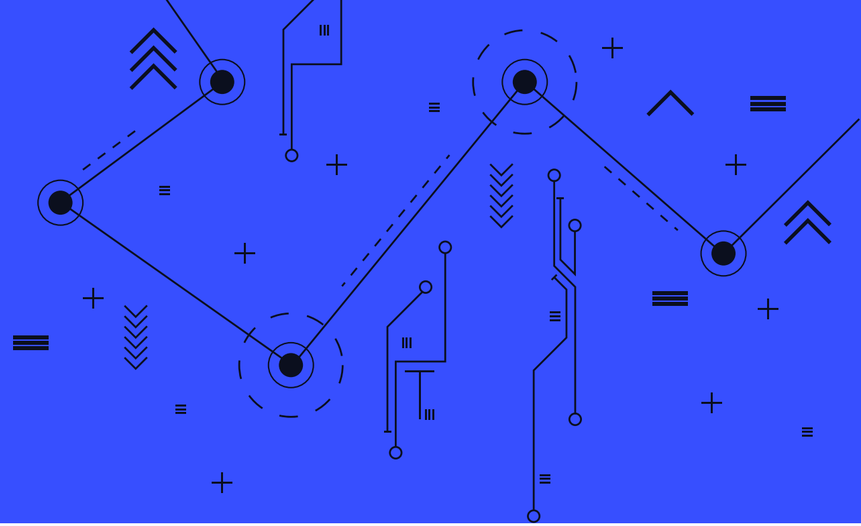
-Pode tornar o projeto excessivamente genérico. A desvantagem de facilitar o acréscimo de novos componentes é que isso torna mais difícil restringir os componentes de uma composição. Algumas vezes, você deseja uma composição que tenha somente certos componentes. Com Composite, você não pode confiar no sistema de tipos para garantir a obediência a essas restrições. Ao invés disso, terá que usar verificações e testes em tempo de execução.

Estrutura



Exemplo do editor de formas geométricas.

OBJECT ADAPTER

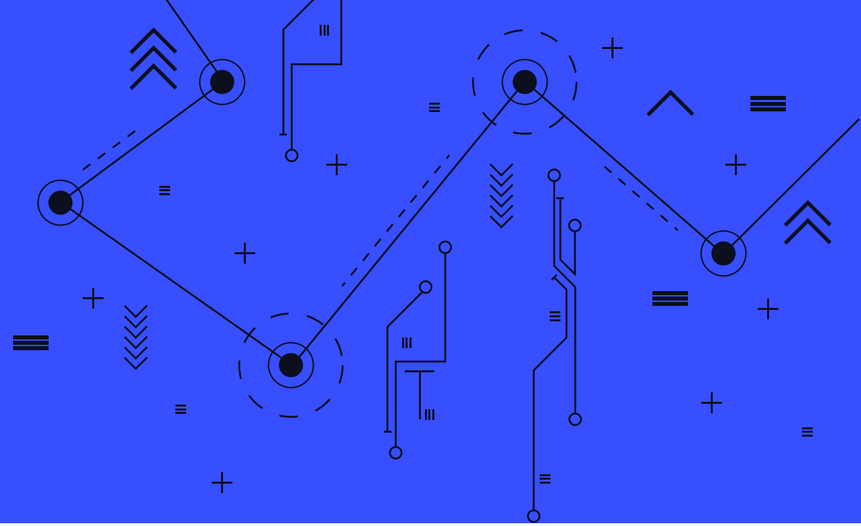


Definição

O padrão Adapter converte a interface de uma classe para outra interface que o cliente deseja encontrar, traduzindo solicitações do formato requerido pelo usuário para o formato compatível com a classe Adapter e as redirecionando. Dessa forma, o Adapter permite que classes com interfaces incompatíveis trabalhem juntas.

Wandeson

OBJECT ADAPTER

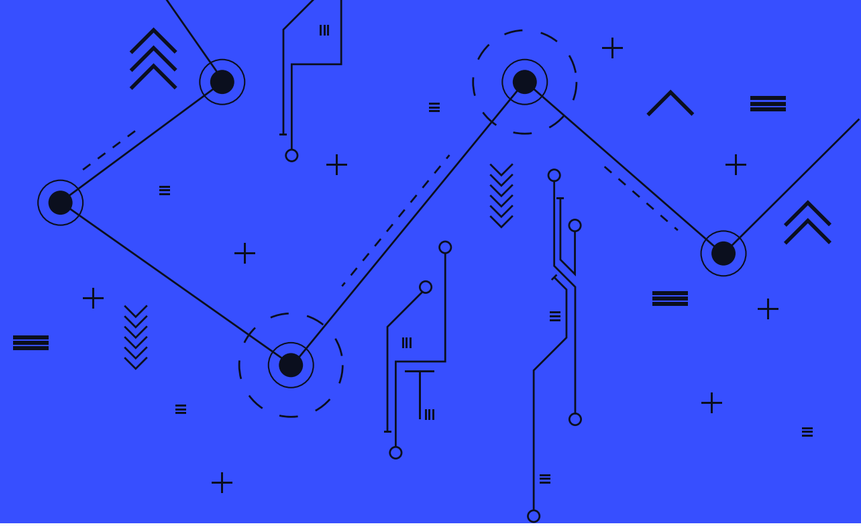


Motivação

Muitas vezes uma classe que poderia ser reaproveitada acaba não sendo reutilizada justamente por sua interface não ser correspondente à interface específica de um domínio requerido por uma aplicação.

Wandeson

OBJECT ADAPTER



Aplicabilidade

-Quando se deseja utilizar uma classe existente, porém sua interface não corresponde à interface que é necessária.
-Quando o desenvolvedor quiser criar classes reutilizáveis que cooperem com classes não relacionadas ou não-previstas, ou seja, classes que não possuem necessariamente interfaces compatíveis.

-Quando é necessário utilizar muitas subclasses existentes, porém, impossível de adaptar as interfaces criando subclasses para cada uma. Um adaptador de objeto pode adaptar a interface de sua classe mãe (exclusivamente para Object Adapter).

Wandeson

Padrão UML

Client: Que é uma classe que contém a lógica de negócio do programa existente.
Client Interface: Que descreve um protocolo que outras classes devem seguir para ser capaz de colaborar com o código cliente.

Service: Que é alguma classe útil (geralmente de terceiros ou código legado). O cliente não pode usar essa classe diretamente pois ela tem uma interface incompatível.

Adapter: Que é uma classe que é capaz de trabalhar tanto com o cliente quanto o serviço: ela implementa a interface do cliente enquanto encobre o objeto do serviço. O adaptador

Allan

Consequências

-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.

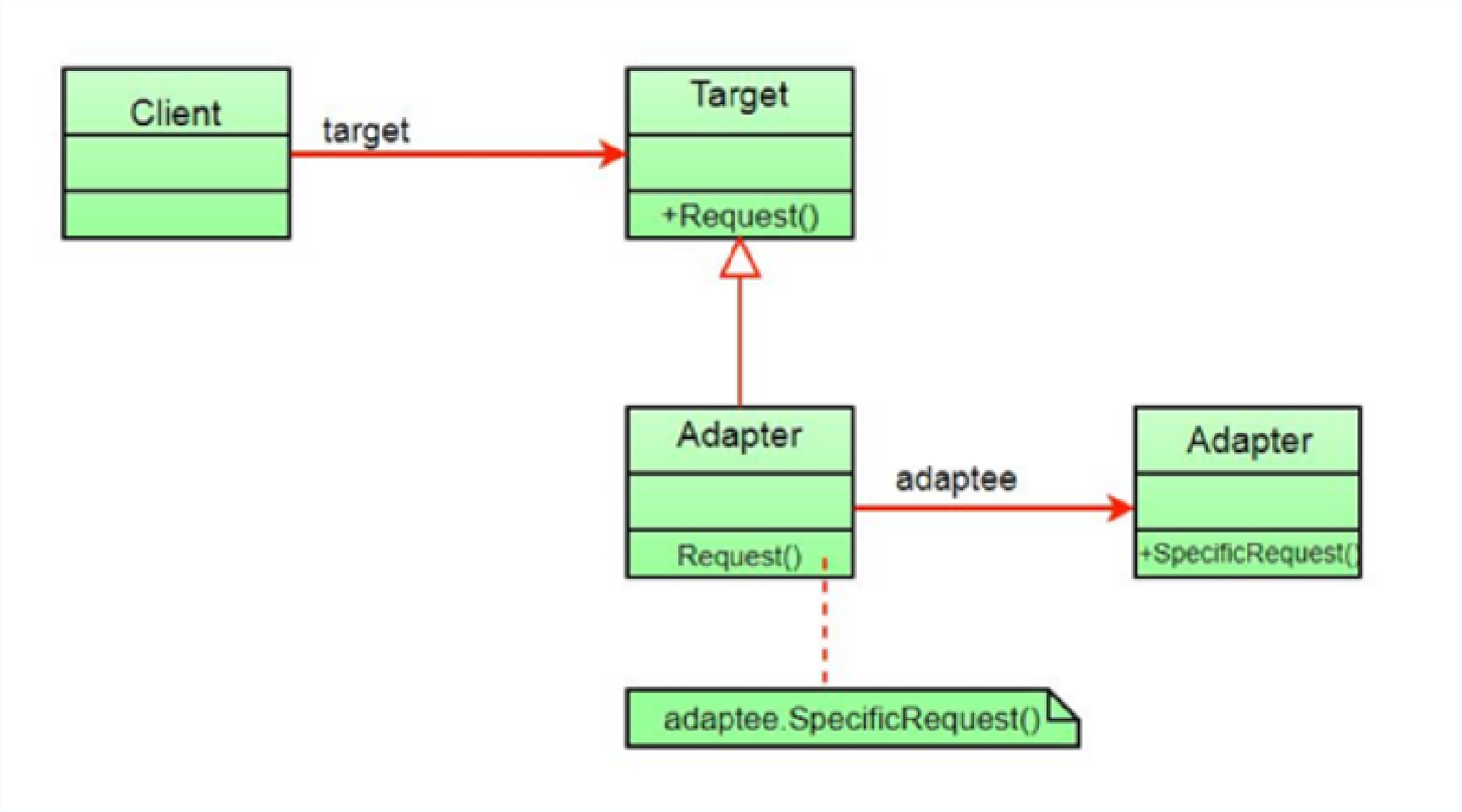
- Permite que a classe Adapter substitua algum comportamento da classe Adaptee, uma vez que Adapter é uma subclasse de Adaptee;

- Introduz somente um objeto, e não é necessário endereçamento indireto adicional por ponteiros para chegar até a classe Adaptee.

- Permite a um único Adapter trabalhar com muitos Adaptees, ou seja, o Adaptee em si e todas as suas subclasses (caso existam);

- Torna-se mais difícil redefinir um comportamento de uma classe Adaptee. Ele exigirá a criação de subclasses de Adaptee e fará com que a classe Adapter seja referência a subclasse, ao invés da classe Adaptee em si.

Estrutura


Client
target
Target
+Request()
Adapter
Request()
adaptee
Adapter
+SpecificRequest()
adaptee.SpecificRequest()

Exemplo de Código: Listagem 1: Exemplo de implementação do Padrão Adapter

```
public class TomadaDeDoisPinos {
 public void ligarNaTomadaDeDoisPinos() {
  System.out.println("Ligado na Tomada de Dois Pinos");
 }
}
public class TomadaDeTresPinos {
 public void ligarNaTomadaDeTresPinos() {
  System.out.println("Ligado na Tomada de Tres Pinos");
 }
}
public class AdapterTomada extends TomadaDeDoisPinos {
 private TomadaDeTresPinos tomadaDeTresPinos;
 public AdapterTomada(TomadaDeTresPinos tomadaDeTresPinos) {
  this.tomadaDeTresPinos = tomadaDeTresPinos;
 }
 public void ligarNaTomadaDeDoisPinos() {
  tomadaDeTresPinos.ligarNaTomadaDeTresPinos();
 }
}
```

Allan

Listagem 2: Exemplo de execução do padrão adapter

```
public class Teste {

 public static void main(String args[]) {
  TomadaDeTresPinos t3 = new TomadaDeTresPinos();
  AdapterTomada a = new AdapterTomada(t3);
  a.ligarNaTomadaDeDoisPinos();
 }
}
```

REFERÊNCIAS

CCO3 BV MA 2020

- https://refactoring.guru/pt-br/design-patterns/adapter
- https://www.devmedia.com.br/padrao-de-projeto-adapter-em-java/26467]
- https://refactoring.guru/pt-br/design-patterns/composite
- https://medium.com/@gbbigardi/arquitetura-e-desenvolvimento-de-software-parte-8-composite-9d342d641a4a

OBRIGADO

CCO3 BV MA 2020

LINK DO VIDEO

CCO3 BV MA 2020

https://vimeo.com/manage/videos/551890901