Adapter

Padrão de Projeto

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educaçãoo, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Campus Rio Grande
Divisão de Computação

Agenda

Motivação

Um cenário onde existe a necessidade de criar uma classe que envolva outra é quando possuímos uma classe que implementa uma interface, mas precisamos que ela implemente outra.

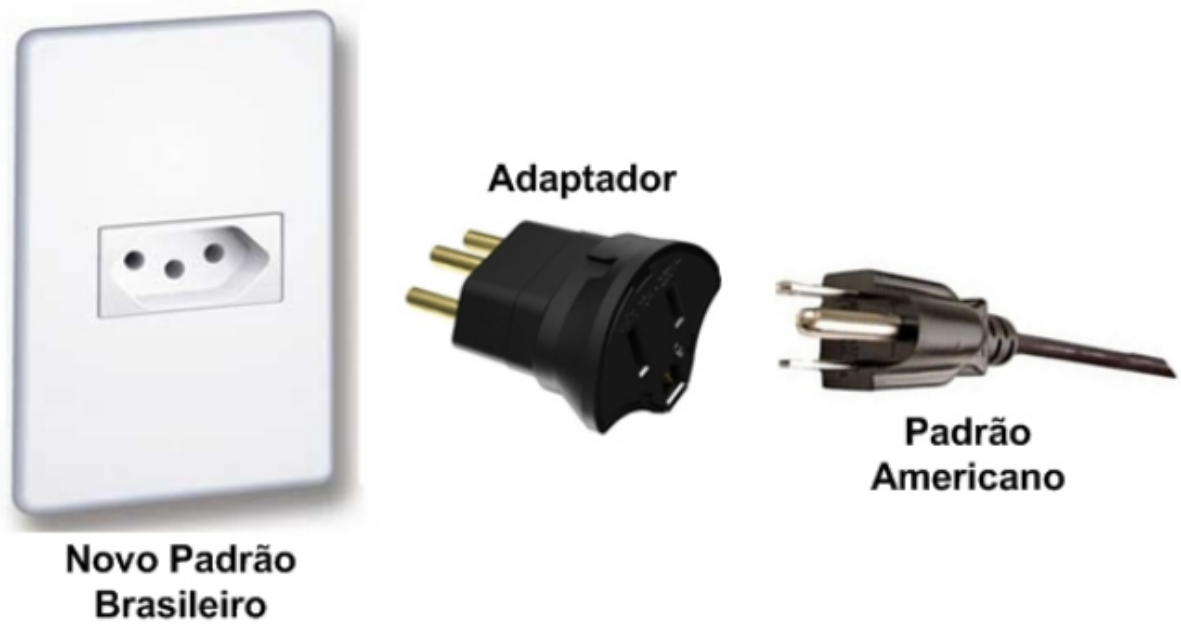
Um exemplo do mundo físico ocorre quando compramos um notebook nos Estados Unidos e precisamos ligá-lo aqui nas tomadas do Brasil.

Motivação

- No caso, o notebook americano possui uma fonte que possui a interface das tomadas americanas, porém precisamos que ele possua a que permita que ele seja ligado aqui no Brasil.

Todos sabem que esse problema não é difícil de resolver, bastando um adaptador que encaixe em uma entrada no padrão brasileiro e possua uma entrada no padrão americano.

Motivação



Motivação

Ok, é assim que funciona no mundo real, mas e os adaptadores orientados a objetos?

Bem, nossos adaptadores O.O desempenham o mesmo papel dos seus equivalentes no mundo real: eles recebem uma interface e a adaptam para algo que o cliente está esperando.

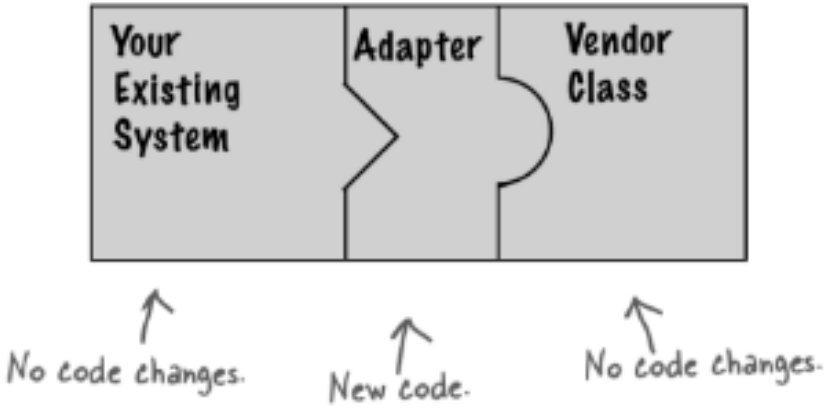
Adaptadores orientados a objetos

Digamos que você tenha que encaixar uma nova biblioteca de classes, adquirida de outro fornecedor, em um sistema de software já existente, mas as interfaces projetadas por esse novo fornecedor sejam diferentes das adotadas pelo fornecedor anterior:

Figura : Sistema Existente vs Classe do Novo Fornecedor

Adaptadores orientados a objetos

Muito bem, você não quer resolver o problema mudando todo o código existente (e nem pode mudar o código do novo fornecedor). O que fazer? Bem, você pode escrever uma classe que adapte a interface do novo fornecedor ao formato que o sistema esperar encontrar.



Se comunica com a interface do novo fornecedor para atender as suas solicitações

Motivação

A ideia do padrão Adapter não é muito diferente de um adaptador de tomadas. Criamos uma classe que implementa a interface necessária, a qual precisamos nos adaptar, a "tomada nova".

- Internamente essa classe é composta por uma referência a um objeto que implementa a outra interface, a "tomada antiga".
- Dessa forma, o Adapter traduz as chamadas da interface que ele implementa para a classe que ele encapsula.

Motivação

Isso nem sempre é trivial, pois normalmente não é somente o nome do método que muda, mas outras questões muitas vezes inesperadas.

Exemplo

Se anda como um pato e grasna como um pato, então talvez seja um peru envelopado em um adaptador de pato...

Exemplo

Chegou a hora de ver um **adaptador** em ação. Lembra-se dos nossos patos? Vamos examinar agora uma versão ligeiramente simplificada das interfaces e classes Duck:

```
1  public interface Duck {
2      public void quack();
3      public void fly();
4  }
```

Aqui está uma subclasse de Duck, (MallardDuck).

```
1  public class MallardDuck implements Duck {
2      public void quack(){
3          System.out.println("Quack");
4      }
5      public void fly(){
6          System.out.println("I'm flying");
7      }
8  }
```

Exemplo

E agora chegou o momento de conhecer a ave mais nova da turma:

```java
public interface Turkey {
    public void gobble();
    public void fly();
}
```

```java
public class WildTurkey implements Turkey {
    public void gobble(){
        System.out.println("Gobble gobble");
    }

    public void fly(){
        System.out.println("I'm flying a short distance");
    }
}
```

Exemplo

Agora, digamos que os objetos Duck estejam em falta e você queira usar alguns objetos Turkey em seu lugar.

Obviamente, não há como usar os perus do jeito que estão, porque eles têm uma interface diferente.

Exemplo

Portanto, vamos escrever um Adaptador:

```
1  public class TurkeyAdapter implements Duck {
2      Turkey turkey;
3
4      public TurkeyAdapter(Turkey turkey){
5          this.turkey = turkey;
6      }
7
8      public void quack(){
9          this.turkey.gobble();
10     }
11
12     public void fly(){
13         for (int i = 0; i < 5; i++) {
14             this.turkey.fly();
15         }
16     }
17 }
```

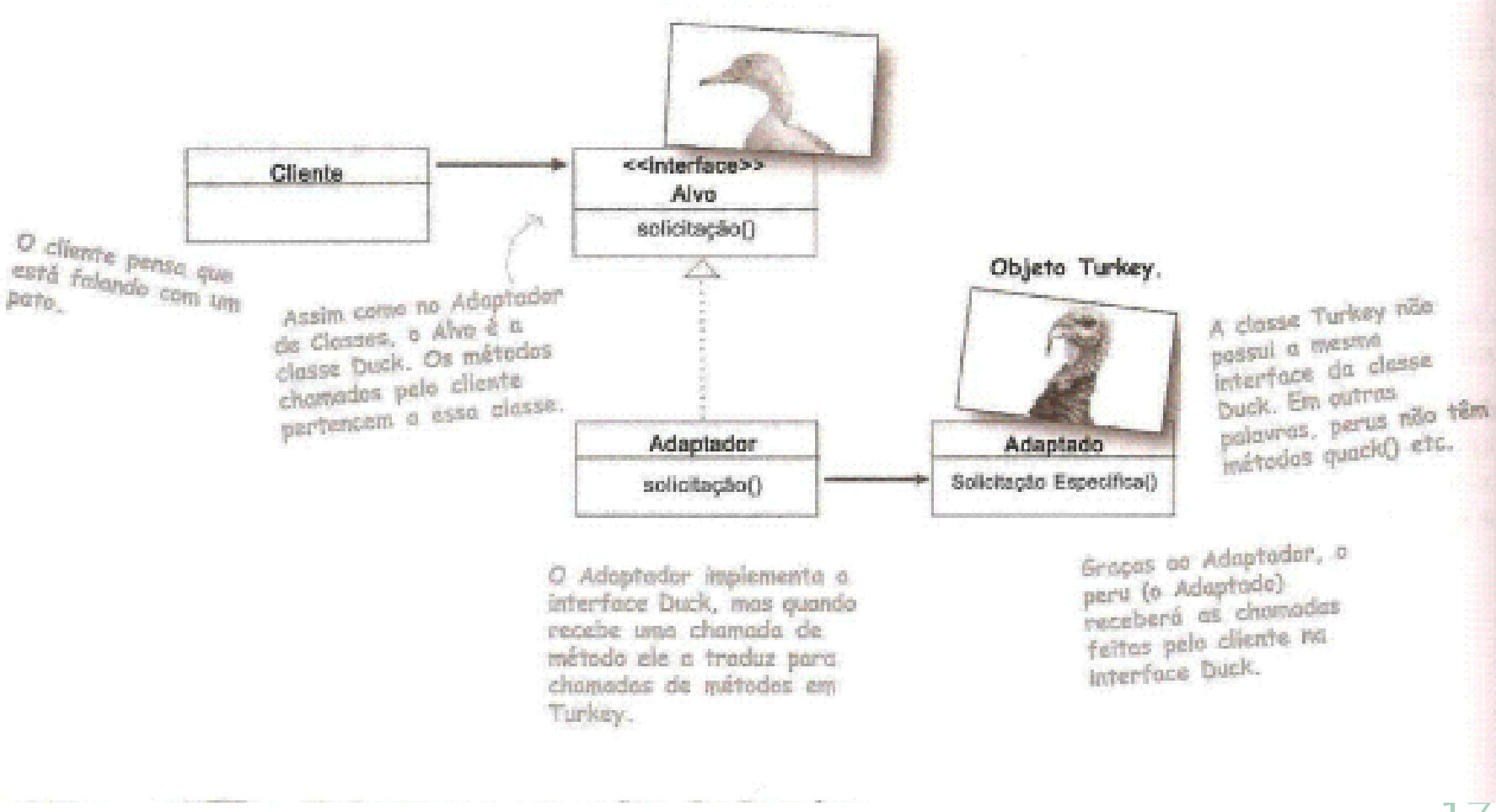
Exemplo

```
1  public class DuckTestDrive {
2      public static void main (String[] args){
3          MallardDuck duck = new MallardDuck();
4
5          WildTurkey turkey = new WildTurkey();
6          Duck turkeyAdapter = new TurkeyAdapter(turkey);
7
8          System.out.println("The Turkey says....");
9          turkey.gobble();
10         turkey.fly();
11
12         System.out.println("\nThe Duck says...");
13         testDuck(duck);
14
15         System.out.println("\nThe TurkeyAdapter says....");
16         testDuck(turkeyAdapter);
17     }
18     static void testDuck(Duck duck){
19         duck.quack();
20         duck.fly();
21     }
22 }
```

Exemplo

1. O cliente faz uma solicitação ao adaptador chamando um método nele através da interface-alvo
2. O adaptador traduz a solicitação para uma ou mais chamadas de métodos no adaptado usando a interface desse objeto
3. O cliente recebe os resultados da chamada sem jamais perceber que há um adaptador fazendo a tradução

Exemplo


Cliente
<<interface>>
Alvo
solicitação()
O cliente pensa que está falando com um pato.
Assim como no Adaptador de Classes, o Alvo é a classe Duck. Os métodos chamados pelo cliente pertencem a essa classe.
Objeto Turkey.
Adaptador
solicitação()
Adaptado
Solicitação Específica()
A classe Turkey não possui a mesma interface da classe Duck. Em outras palavras, perus não têm métodos quack() etc.
O Adaptador implementa a interface Duck, mas quando recebe uma chamada de método ele a traduz para chamadas de métodos em Turkey.
Graças ao Adaptador, o peru (o Adaptado) receberá as chamadas feitas pelo cliente na interface Duck.

Estrutura

A estrutura do padrão Adapter é similar à dos padrões **Proxy** e **Decorator**, sendo que a principal diferença é que a classe que está sendo encapsulada não possui a mesma interface da que a está envolvendo.

- É justamente esse fato que permite a adaptação entre as interfaces.

Estrutura

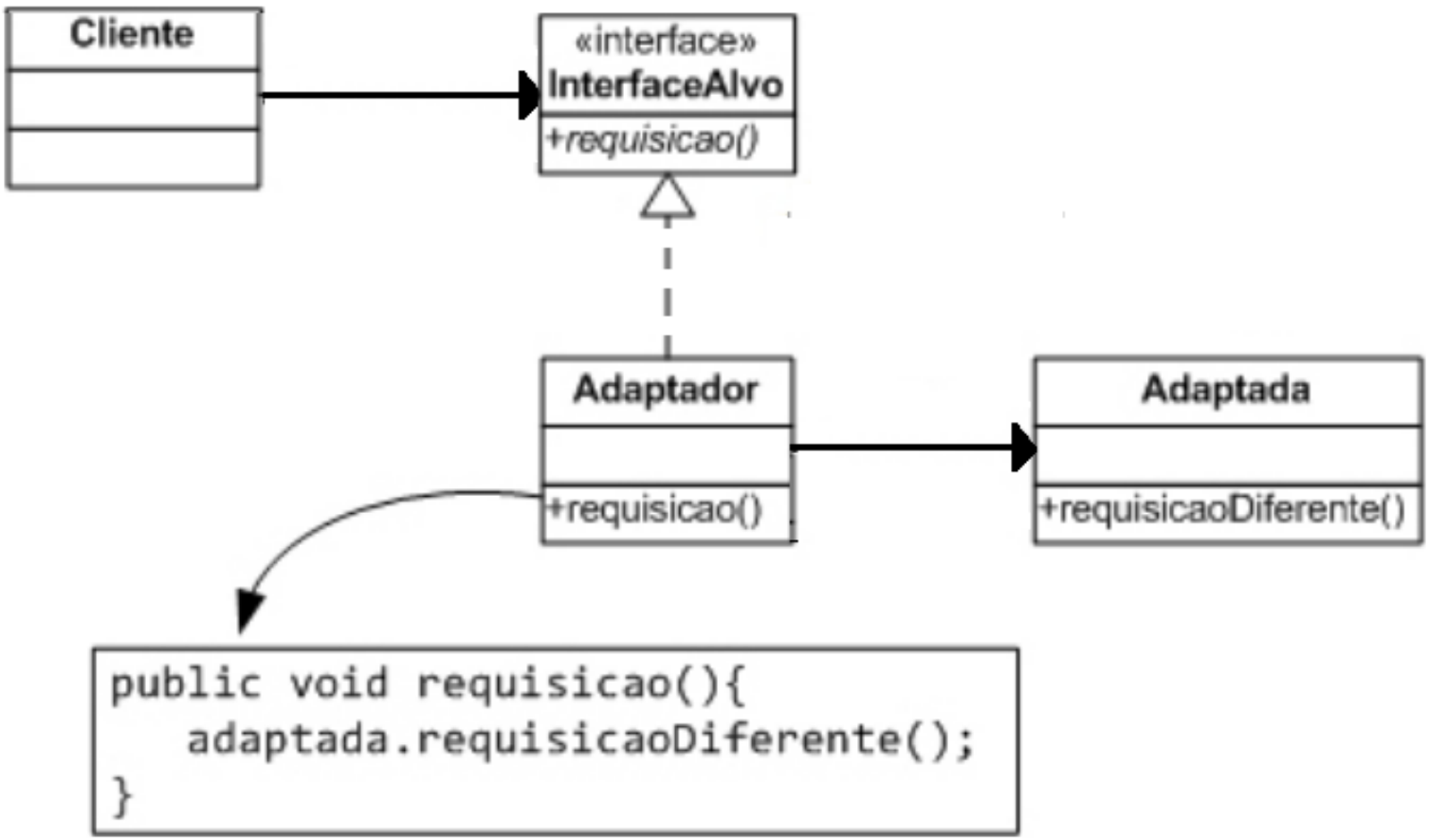


Figura : Estrutura

Estrutura

Nesse diagrama temos uma classe **Cliente** que depende de alguma forma de uma **InterfaceAlvo**.

- Sendo assim, para tornar possível ela acessar uma classe **Adaptada** que não implementa essa abstração, ela precisa utilizar a classe **Adaptador**, que irá receber chamadas de método conforme a **InterfaceAlvo** e irá chamar os métodos correspondentes na classe **Adaptada**.

Estrutura

É importante ficar claro que nem sempre a adaptação é simples de ser feita.

- Questões comuns que precisam ser traduzidas incluem a nomenclatura de classes e métodos, além da conversão de parâmetros e retornos.
- Questões mais complexas podem envolver diferentes formas de interagir com as classes.
  - Por exemplo, o que é um parâmetro de método em uma pode ser um atributo passado no construtor na outra, ou ainda, o que é o retorno em uma pode ser um atributo populado em um parâmetro passado para o método na outra.

Estrutura

Porém, a maior dificuldade é quando existem diferenças semânticas entre as diferentes APIs.

- Nesse caso, normalmente é necessária a ajuda de especialistas do domínio para compreender as diferenças e possibilitar a tradução.

Aplicabilidade

- Você quiser usar uma classe existente, mas sua interface não corresponde à interface de que necessita;
- Você quiser usar uma classe reutilizável que coopere com classes não relacionadas ou não-previstas, ou seja, classes que não necessariamente tenham interfaces compatíveis;

Trabalho

Imagine uma aplicação que interaja com o serviço de SMS de operadoras de celular.

Considere que a versão inicial da aplicação possua uma interface para interagir com o serviço de apenas uma operadora.

Trabalho

A figura a seguir apresenta a interface **SMSSender** que é utilizada pela aplicação para o envio de mensagens de sms's.

```
public interface SMSSender {
    public boolean sendSMS(SMS sms);
}

public class SMS {
    private String destino;
    private String origem;
    private String texto;
    //getters e setters omitidos
}
```

Figura : Interface Usada pelo Sistema para Envio de mensagens

Trabalho

Essa classe possui o método **sendSMS()** que recebe como parâmetro uma instância da classe **SMS**, também apresentada na figura, e retorna um valor booleano dizendo se a mensagem foi enviada com sucesso ou não.

Trabalho

Problema Proposto

Ao evoluir a aplicação, foi necessário incorporar o serviço de envio de mensagens de uma outra operadora.

Não foi uma surpresa muito grande quando foi constatado que a API para o acesso a funcionalidade era completamente diferente.

Trabalho

A figura com a interface dessa nova API, **EnviadorSMS**, está representada a seguir.

```
public interface EnviadorSMS {
    public void enviarSMS(String destino, String origem, String[] msgs)
        throws SMSException;
}
```

Figura : Nova Interface Usada pelo Sistema para Envio de Mensagens

Trabalho

- **A primeira diferença está nos parâmetros**, que em vez de serem encapsulados em uma classe, no caso a SMS, são passado diretamente para o método.
- Outra diferença é que o **texto da mensagem não é mais uma string, mas um array de strings**.
  - A primeira API recebia um texto longo e dividia em várias mensagens se necessário, porém nessa outra API a mensagem precisa ser divida em trechos de 160 caracteres antes da chamada do método.
- Finalmente, **a nova API lança uma exceção para indicar uma falha, e não retorna um valor booleano para indicar o sucesso ou não.**

Trabalho

Para evitar que as classes cliente precisem ser alteradas e se acoplarem a uma nova API, decidiu-se criar um **Adapter** para traduzir as chamadas de uma interface para outra.

Descrição

Implemente o padrão Adapter para o problema proposto.

Desafio: + 0.5

Enviar, verdadeiramente, sms.

Adapter

Padrão de Projeto

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educaçãoo, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Campus Rio Grande
Divisão de Computação