AULA 12 – PADRÃO ADAPTER

GSI020 – Programação Orientada a Objetos II

Prof. Dr. Murillo G. Carneiro
mgcarneiro@ufu.br

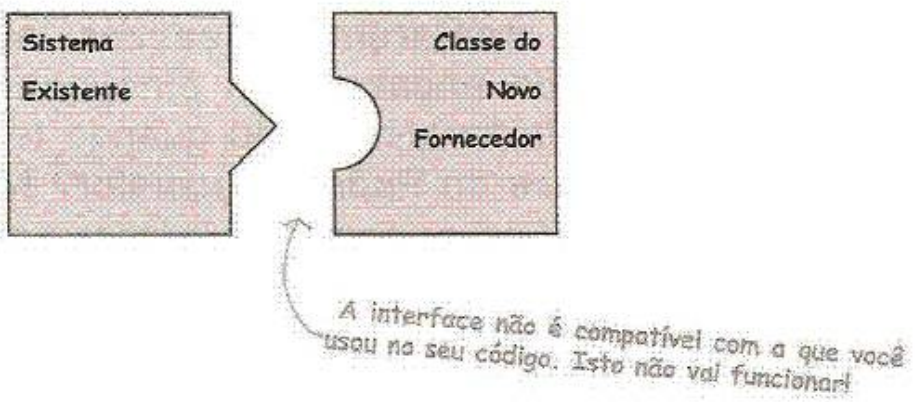
UFU

Material baseado nos slides cedidos pelo Prof. Rafael D. Araújo (FACOM/UFU)

Objetivo da aula

- Entender o funcionamento do padrão de projeto *Adapter (Adaptador)*.

Motivação



- Imagine que sua aplicação utilize uma biblioteca e que, após algum tempo, algumas interfaces dessa biblioteca foram alteradas (novos parâmetros inseridos, por exemplo).
- Como fazer a interface de um objeto parecer o que ela de fato não é?

Adaptador

- É um padrão de projeto de propósito **estrutural**
- Existem dois escopos do padrão Adapter
  - *Class adapter*
  - *Object adapter*
- Propõe uma solução para “encaixar” (adaptar) chamadas de uma interface em outra.
  - *Muito utilizado para criar compatibilidade de bibliotecas*
  - *Também conhecido como Wrapper*

Intenção

- Converter a interface de uma classe em outra interface.
- O *Adapter* permite que classes com interfaces incompatíveis trabalhem em conjunto.

Intenção

Um cliente implementado para uma interface específica (interface-alvo)

requisicao()

Interface alvo

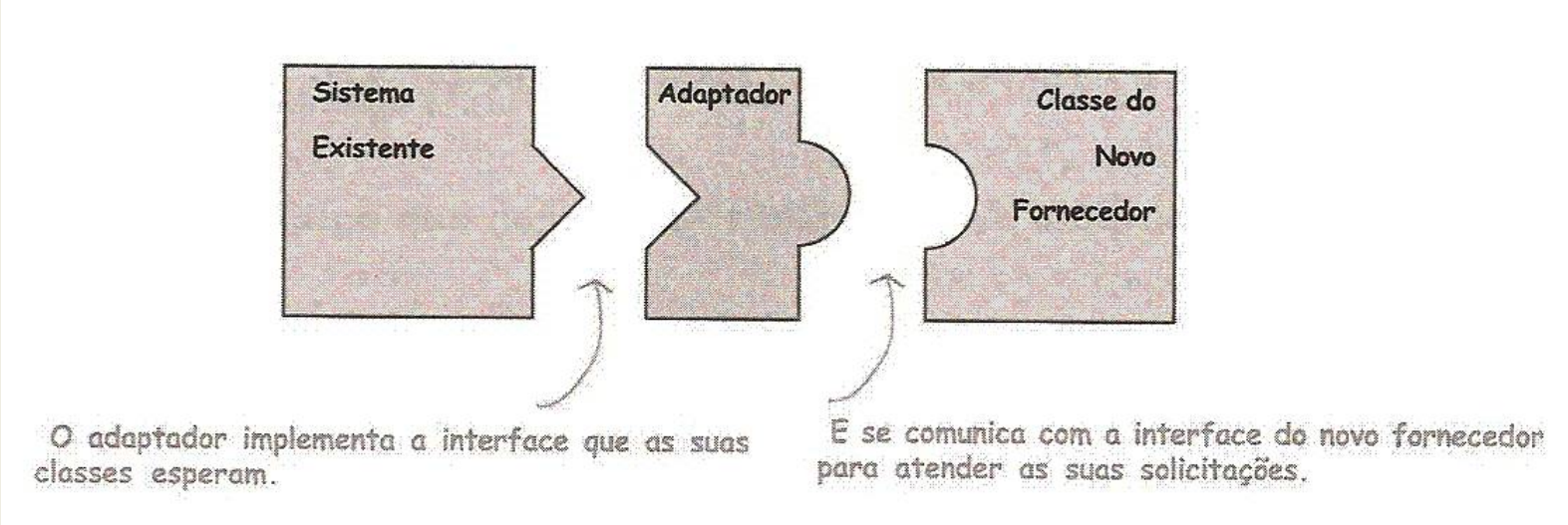
Adaptador

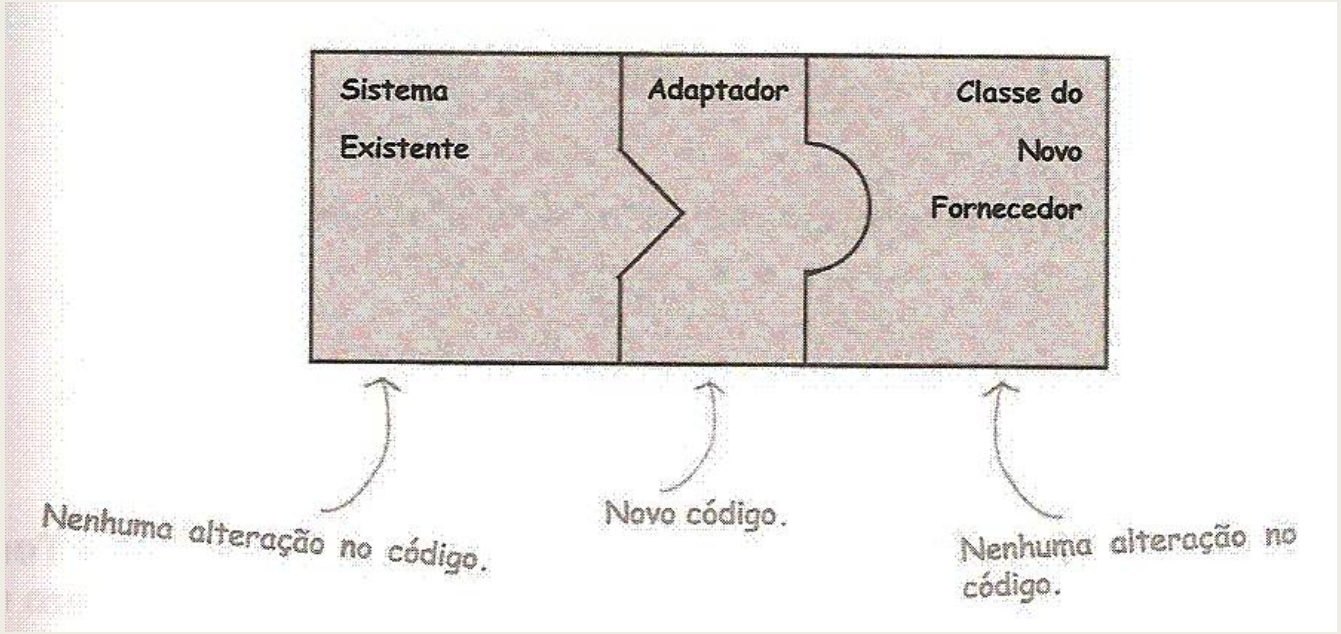
requisicaoTraduzida()

Interface adaptada

O adaptador implementa a interface-alvo e possui uma instância do adaptado

Intenção


Sistema
Existente
Adaptador
Classe do
Novo
Fornecedor
O adaptador implementa a interface que as suas classes esperam.
E se comunica com a interface do novo fornecedor para atender as suas solicitações.


Sistema
Existente
Adaptador
Classe do
Novo
Fornecedor
Nenhuma alteração no código.
Novo código.
Nenhuma alteração no código.

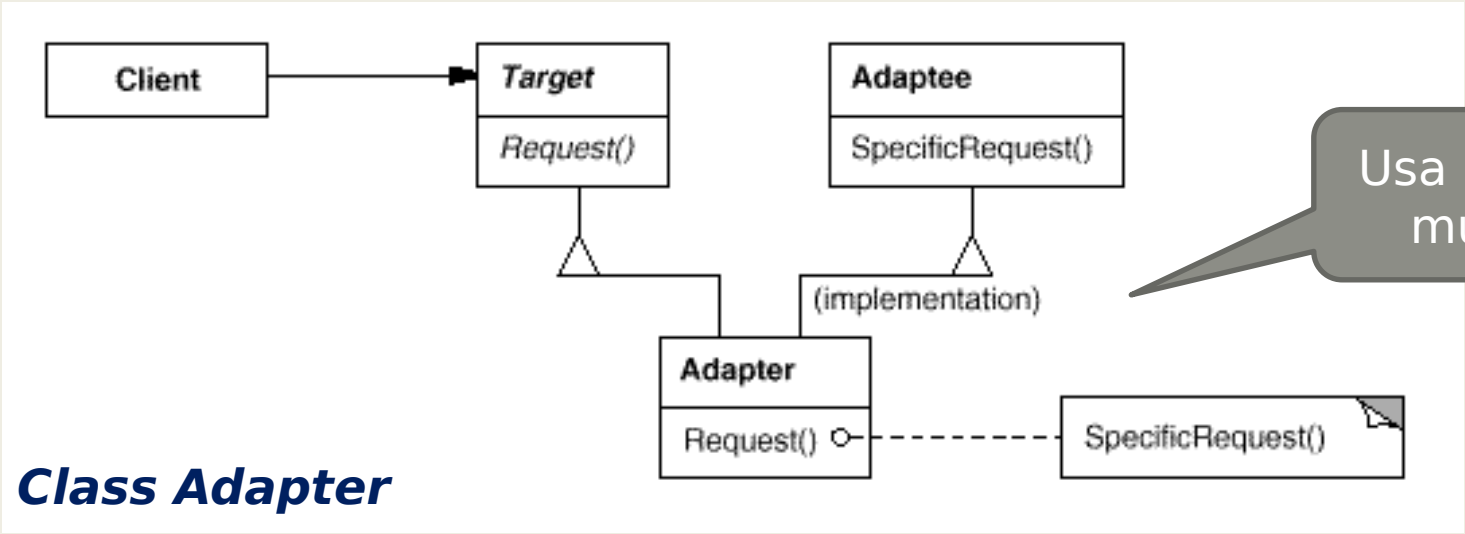
Quando usar

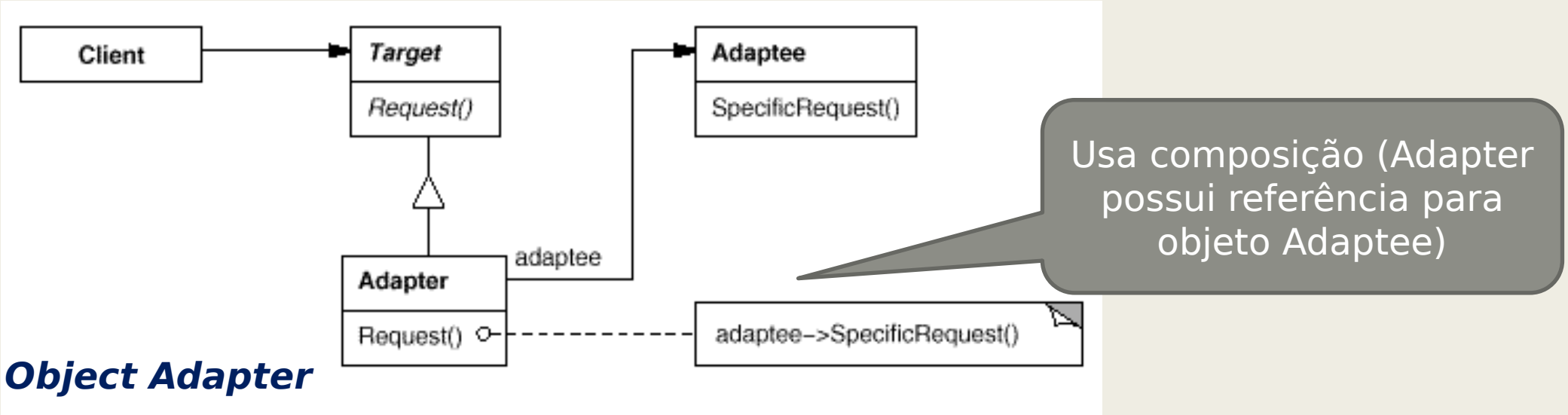
- Precisa-se usar uma classe existente, mas sua interface não corresponde à interface de que necessita
- Deseja-se criar uma classe reutilizável que coopere com classes não relacionadas ou não-previstas, ou seja, classes sem interfaces compatíveis
- Pretende-se usar várias subclasses existentes, porém, é impraticável adaptar estas interfaces criando subclasses para cada uma. Um adaptador pode adaptar a interface da sua classe mãe (*object adapter*)

Elementos participantes

- **Target (alvo)**: define a interface utilizada pelo *Client*.
- **Client (cliente)**: colabora com objetos compatíveis com a interface-alvo.
- **Adaptee (adaptado)**: define uma interface existente que necessita ser adaptada.
- **Adapter (adaptador)**: adapta a interface do adaptado à interface-alvo.

Estrutura


Client
Target
Request()
Adaptee
SpecificRequest()
(implementation)
Adapter
Request()
SpecificRequest()
Usa herança múltipla
Class Adapter


Client
Target
Request()
Adaptee
SpecificRequest()
adaptee
Adapter
Request()
adaptee->SpecificRequest()
Usa composição (Adapter possui referência para objeto Adaptee)
Object Adapter

Benefícios

(Class Adapter)

- *Permite sobrescrever métodos no adaptador*
- *Só precisa criar um único objeto (o adaptador) para conseguir acessar o objeto adaptado*

(Object Adapter)

- *permite a um único Adaptador trabalhar com muitos Adaptados (inclusive subclasses)*

Desvantagens

(Class adapter)

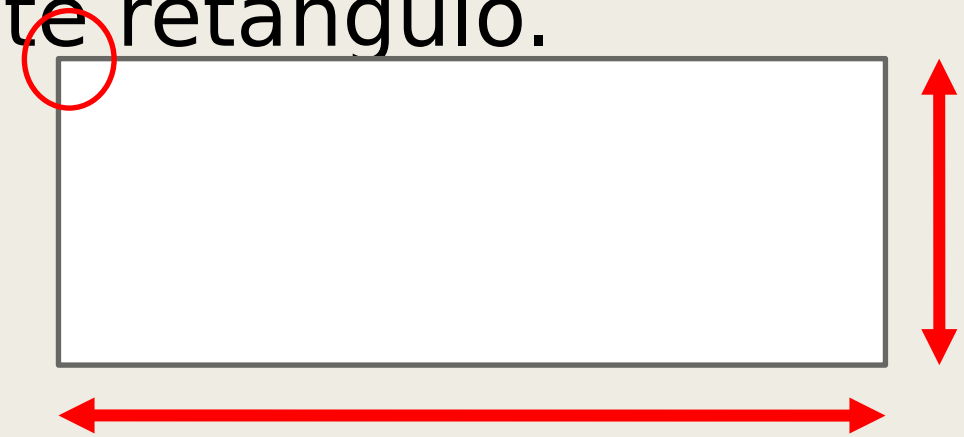
- *Não funciona quando é necessário adaptar uma classe e todas as suas subclasses (hierarquia de classes), pois utiliza herança*

(Object adapter)

- *Precisa-se de uma instância de cada objeto adaptado*
- *Mais difícil sobrescrever métodos*

Problema prático

- Imagine uma classe RetanguloLegado com um método desenhar que recebe um ponto (int x, int y) do canto superior esquerdo, o comprimento e a altura de um retângulo e imprime os 4 pontos dos cantos deste retângulo.

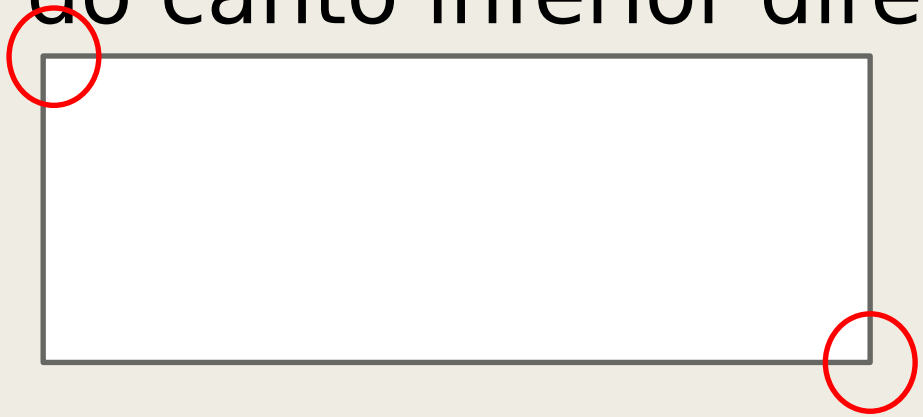


Problema prático

```
public class RetanguloLegado {

    /**
     * p1(x1, y1) .. p2(x1+comprimento, y1)
     * p3(x1, y1-altura) .. p4(x1+comprimento, y1-altura)
     */

    public void desenhar(int x1, int y1, int comprimento, int altura) {

        System.out.println("Método desenhar LEGADO");

        int p2x = x1 + comprimento;
        int p3y = y1 - altura;
        int p4x = x1 + comprimento;
        int p4y = y1 - altura;

        System.out.println("p1(" + x1 + ", " + y1 + ") .. p2(" + p2x + ", " + y1 + ")");
        System.out.println("p3(" + x1 + ", " + p3y + ") .. p4(" + p4x + ", " + p4y + ")");
    }
}
```

Problema prático

- No entanto, a aplicação (*client*) deseja passar apenas dois pontos : um do canto superior esquerdo e um do canto inferior direito



Problema prático

- Implemente a interface Figura com o método desenhar
  - *public void desenhar(int xSE, int ySE, int xID, int yID);*
- Implemente uma classe RetanguloAdapter utilizando o padrão de projeto *Object Adapter* para fazer a conversão.
- Implemente uma classe RetanguloClassAdapter utilizando o padrão de projeto *Class Adapter* para fazer a conversão.

Referências

- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995. Capítulo 4.