



# Padrões de Projeto (cont.)

---

Prof<sup>a</sup>. Rachel Reis  
rachel@inf.ufpr.br



# Padrões de Projeto - Exemplos

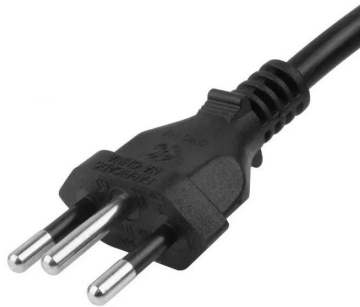
- Exemplos:

Criação	Estrutural	Comportamental
<ul style="list-style-type: none"><li>• Abstract factory</li><li>• Builder</li><li>• Factory Method</li><li>• Prototype</li><li>• Singleton</li></ul>	<ul style="list-style-type: none"><li>• Adapter</li><li>• Bridge</li><li>• Composite</li><li>• Decorator</li><li>• Façade</li><li>• Flyweight</li><li>• Proxy</li></ul>	<ul style="list-style-type: none"><li>• Chain of responsibility</li><li>• Command</li><li>• Interpreter</li><li>• Iterator</li><li>• Mediator</li><li>• Memento</li><li>• Observer</li><li>• Etc.</li></ul>



# Adapter - Analogia

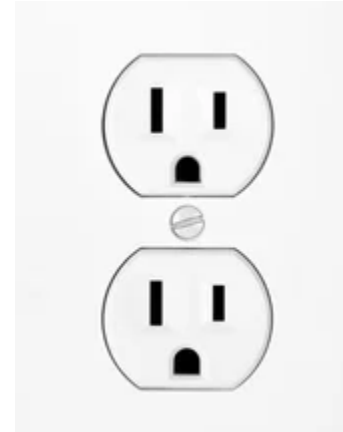
---



Tomada de notebook



Adaptador



Tomada



# Sobre o Adapter

---

- É um padrão de projeto estrutural (organiza classes e objetos para criar estruturas maiores) também conhecido como *wrapper*.
- Faz exatamente o que um adaptador da vida real faz.
- Evita a dependência com códigos externos.
- Utiliza o conceito de interface.



# Sobre o Adapter

---

- Lista de atributos usadas pelo livro GOF para a descrição dos padrões de projeto:

<ul style="list-style-type: none"><li>• <b>Nome</b></li><li>• <b>Intenção</b></li><li>• Motivação</li><li>• Aplicabilidade</li><li>• <b>Estrutura</b></li><li>• Participantes</li></ul>	<ul style="list-style-type: none"><li>• Colaborações</li><li>• Consequências</li><li>• <b>Implementação</b></li><li>• Exemplo de código</li><li>• Usos conhecidos</li><li>• Padrões relacionados</li></ul>
---	--

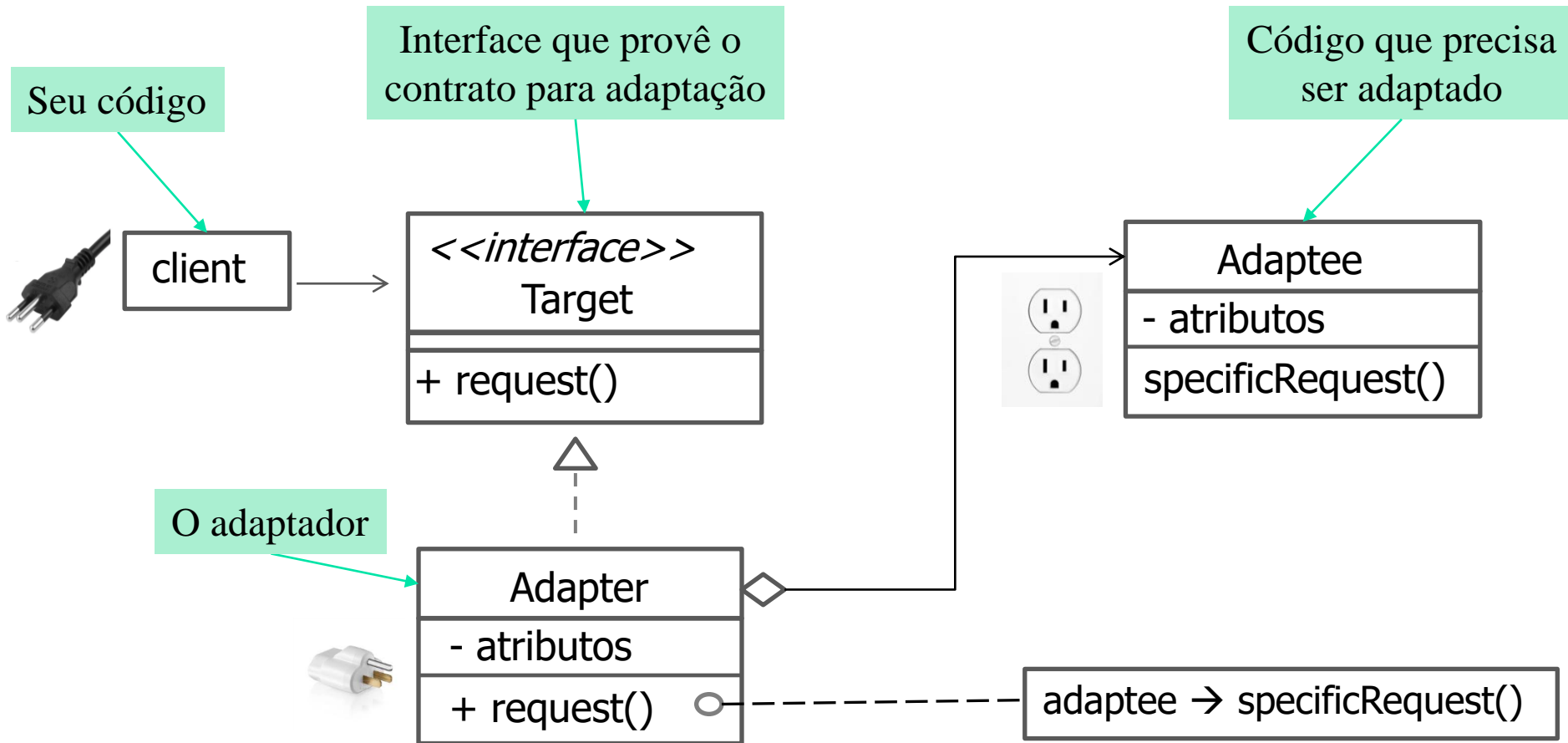


# Adapter - Intenção

---

- Converter a interface de uma classe em outra interface esperada pelos clientes.
- Permite que certas classes trabalhem em conjunto, pois de outra forma seria impossível por causa de suas interfaces incompatíveis.

# Adapter - Estrutura





# Exemplo: Aplicação de Pagamento

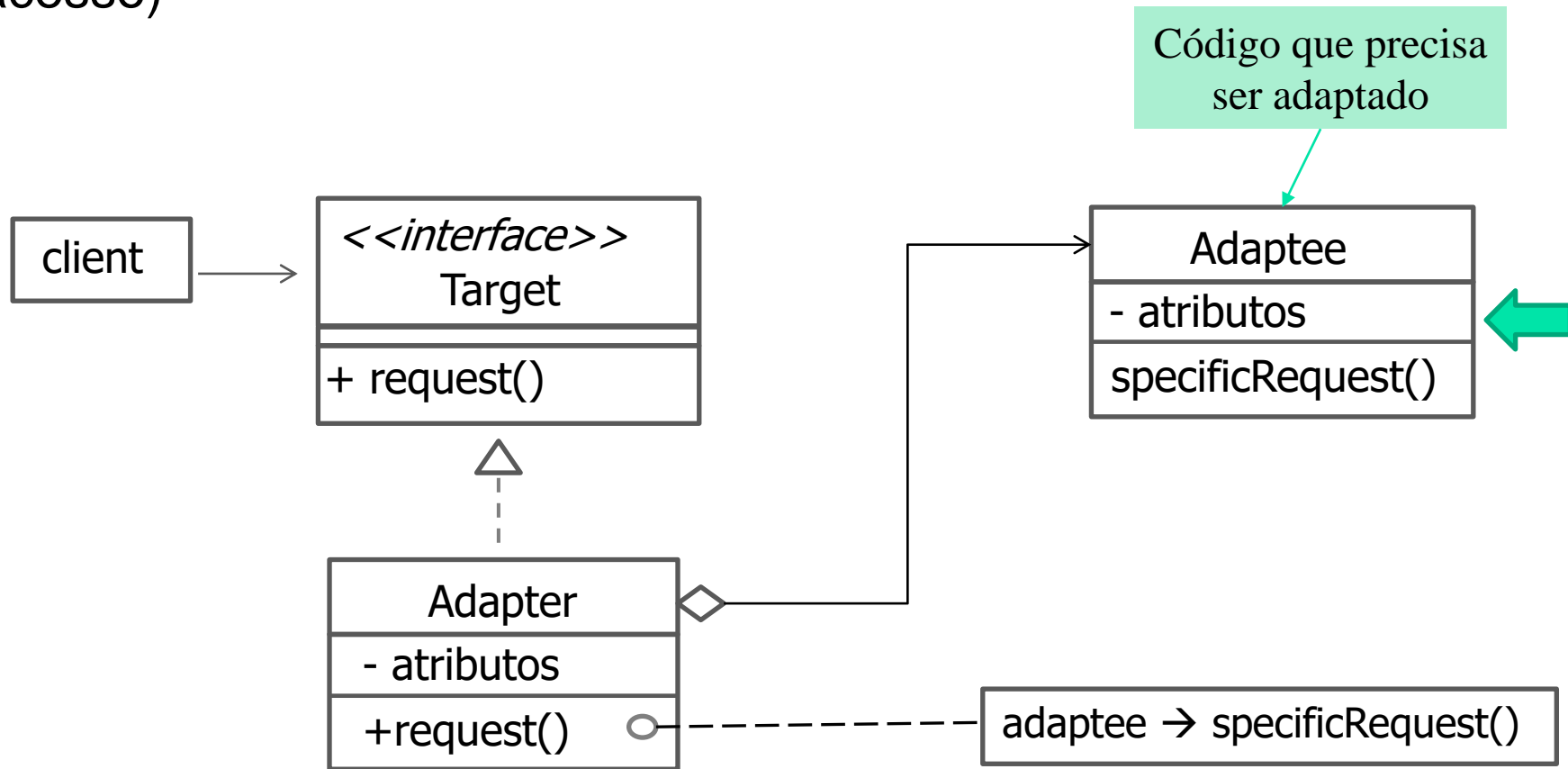
---

- Desenvolva uma aplicação que ofereça a opção de pagamento no cartão de crédito.



# Aplicação - Estrutura

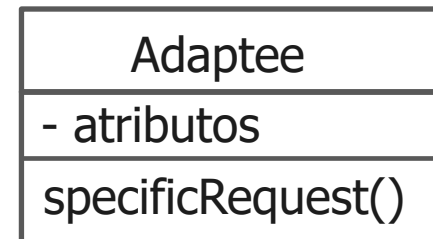
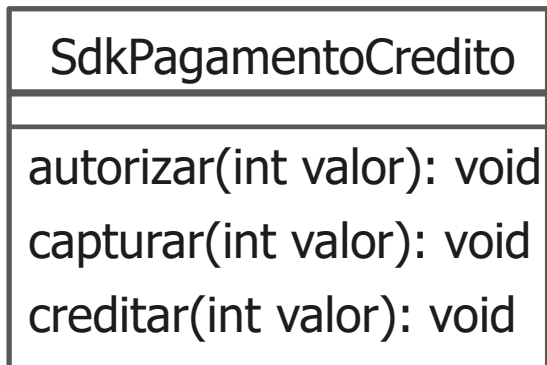
- Classe Adaptee: classe do fornecedor (você não tem acesso)






# Exemplo: Aplicação de Pagamento

- Classe do fornecedor (você não tem acesso) chamada de SdkPagamentoCredito.



Código que precisa ser adaptado



```
public class SdkPagamentoCredito
{
    public void autorizar(int valor){
        // reservar o dinheiro
    }

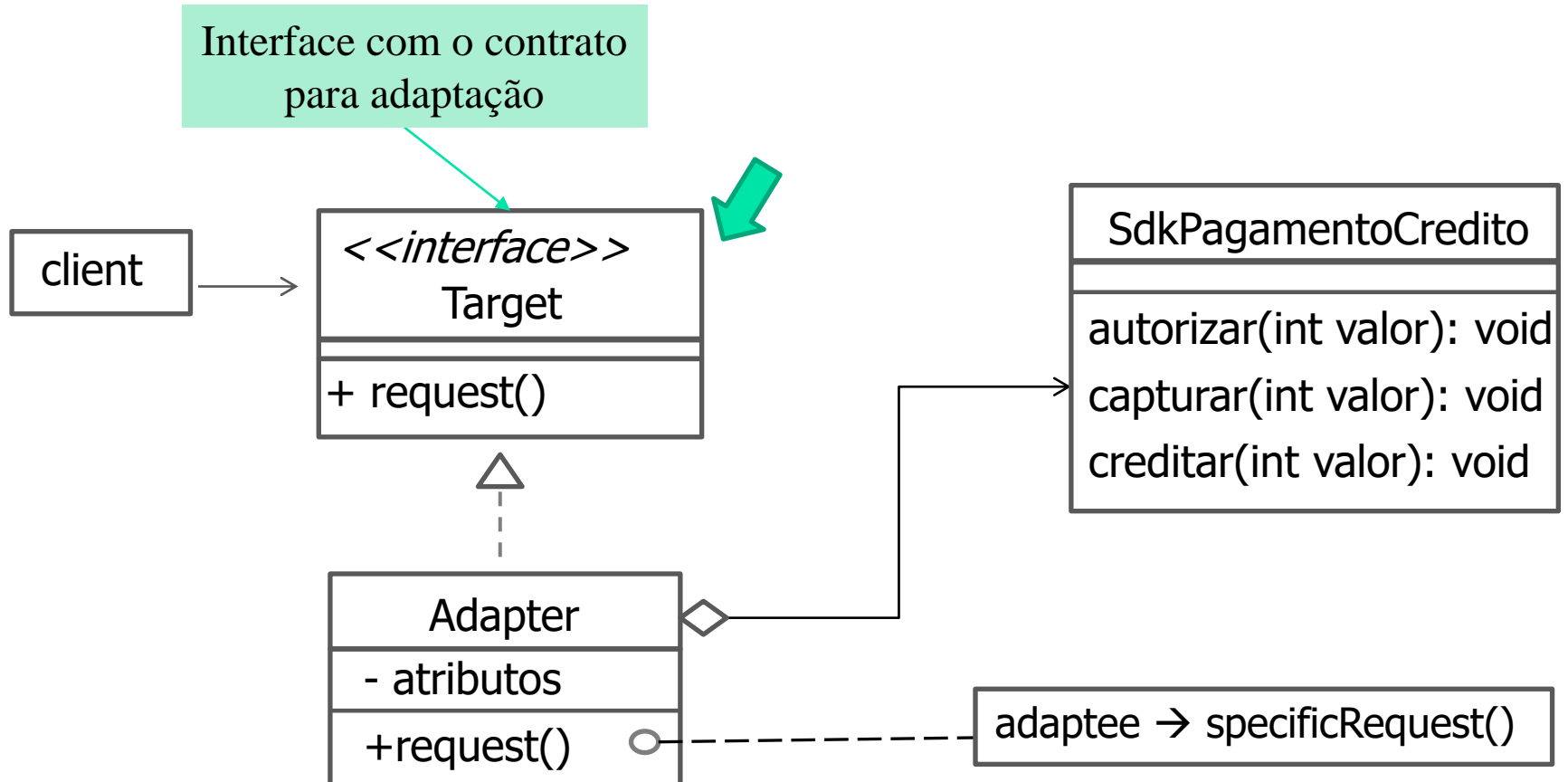
    public void capturar(int valor){
        // efetuar a cobrança
    }

    public void creditar(int valor){
        // extornar dinheiro
    }
}
```

SdkPagamentoCredito
autorizar(int valor): void
capturar(int valor): void
creditar(int valor): void

# Aplicação de Pagamento - Adapter

- Interface Target: provê os métodos para adaptação.

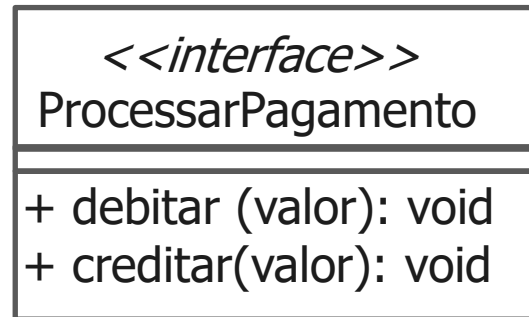
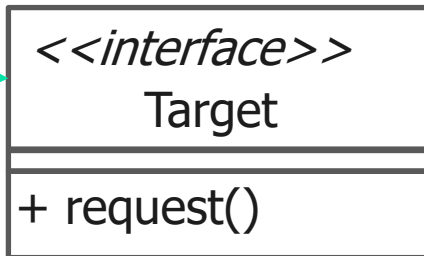




# Exemplo: Aplicação de Pagamento

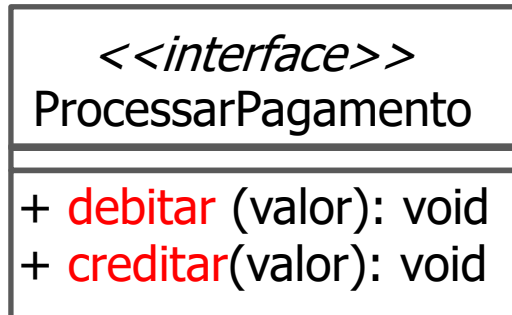
- Interface Target: provê os métodos para adaptação.

Interface que meu  
código precisa



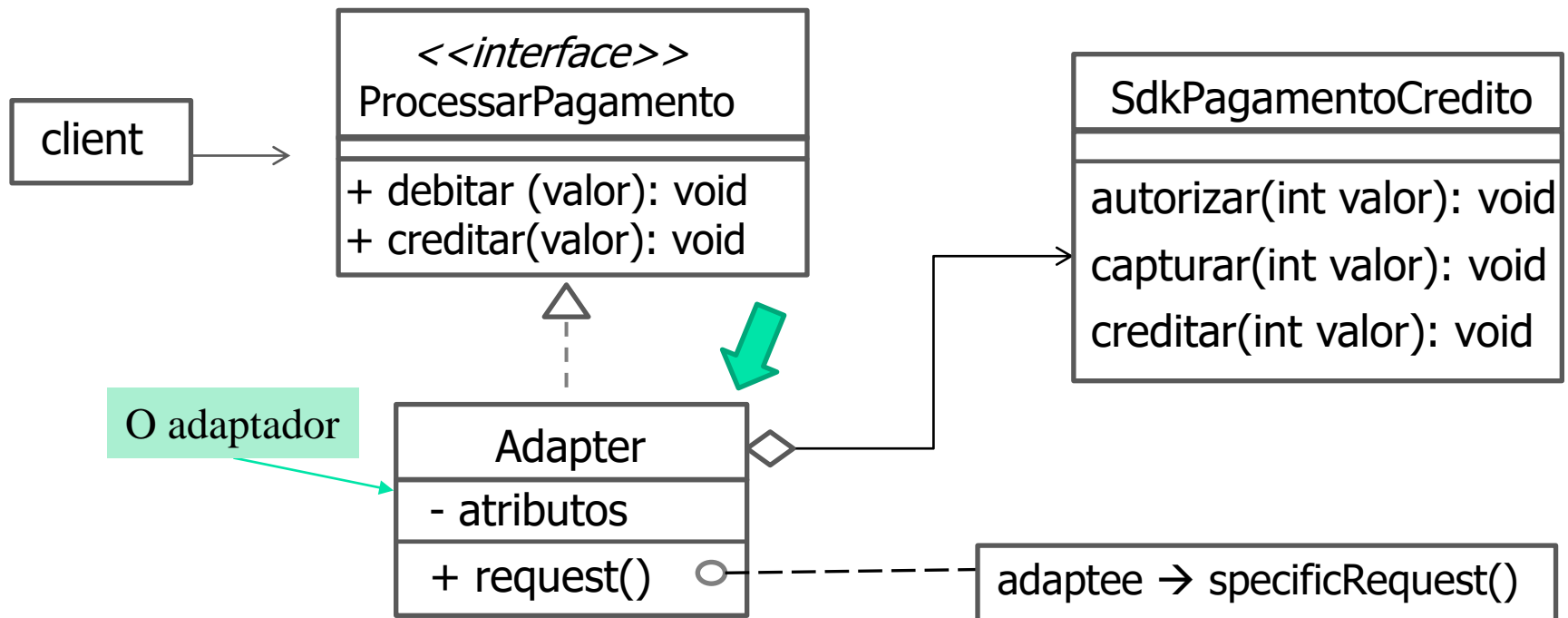
```
public interface ProcessarPagamento
{
    public void debitar(int valor);

    public void creditar(int valor);
}
```



# Aplicação de Pagamento - Adapter

- Classe Adapter: classe interna (você pode alterar, é sua!!!)

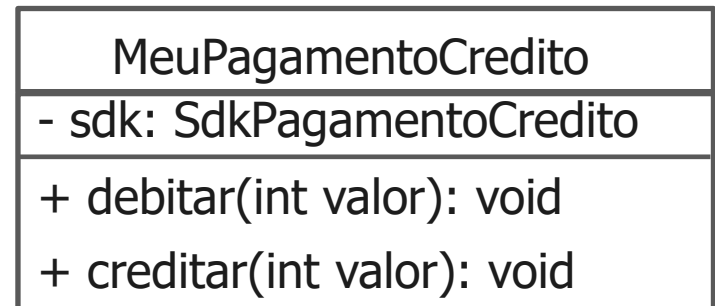
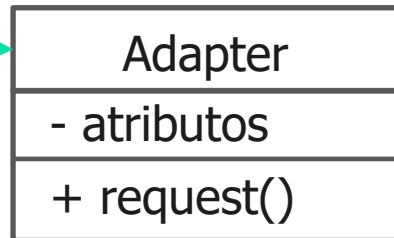




# Exemplo: Aplicação de Pagamento

- Classe interna (você pode alterar, é sua!!!)

O adaptador

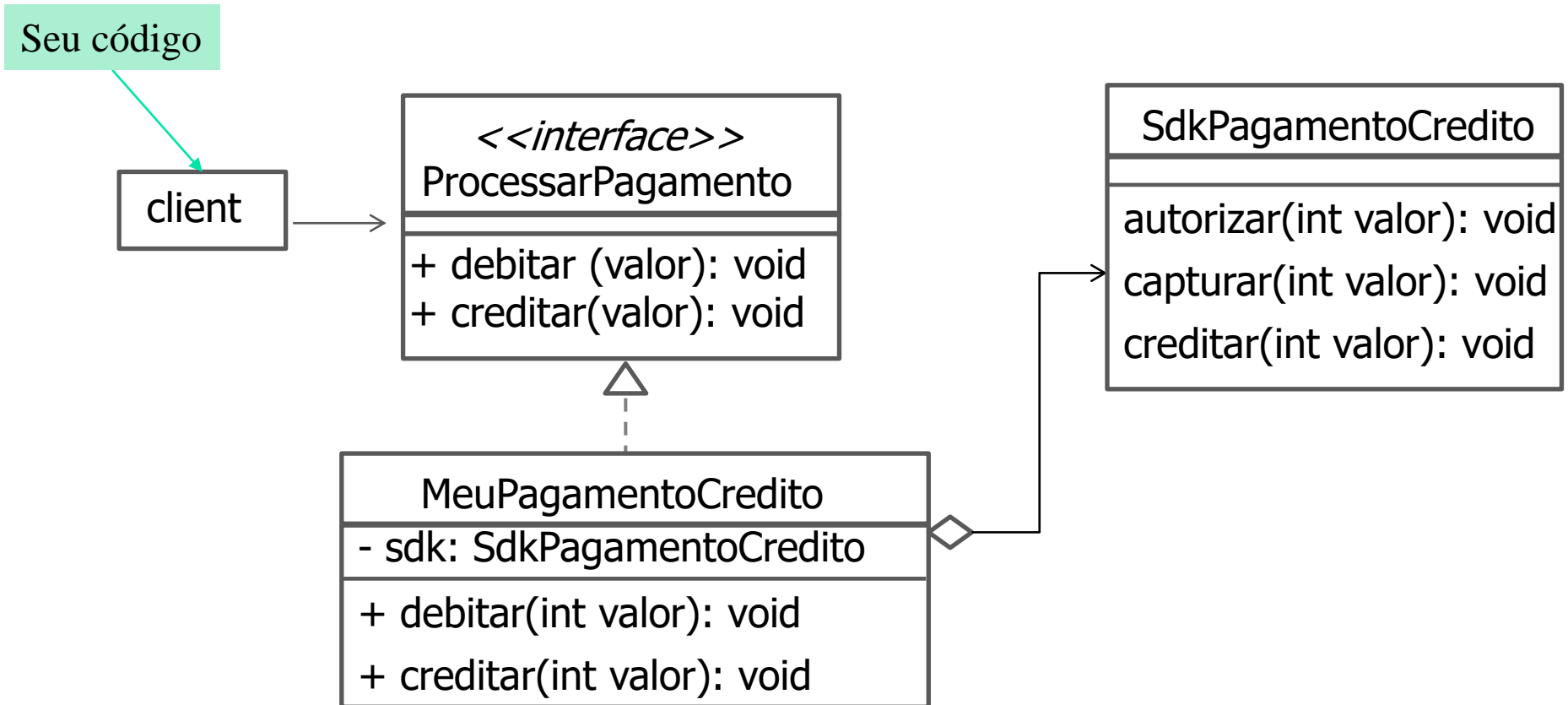




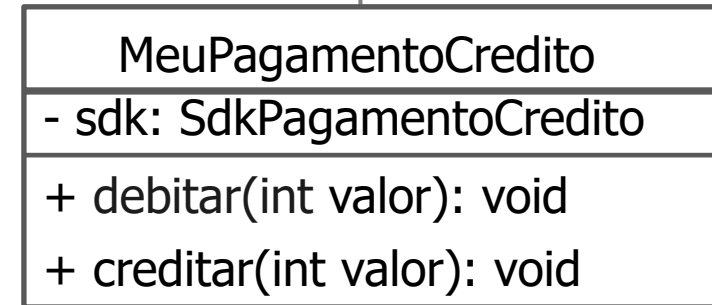
```
public class MeuPagamentoCredito implements ProcessarPagamento{  
    // Atributo do tipo da classe do fornecedor  
    private SdkPagamentoCredito sdk;  
  
    public MeuPagamentoCredito(){  
        // Instância da classe do fornecedor  
        sdk = new SdkPagamentoCredito();  
    }  
  
    public void debitar(int valor){  
        // Métodos da classe do fornecedor  
        sdk.autorizar(valor);  
        sdk.capturar(valor);  
    }  
  
    public void creditar(int valor){  
        // Método da classe do fornecedor  
        sdk.creditar(valor);  
    }  
}
```

MeuPagamentoCredito
- sdk: SdkPagamentoCredito
+ debitar(int valor): void
+ creditar(int valor): void

# Aplicação de Pagamento - Adapter



```
public class Principal
{
    public static void main(String []args)
    {
        ProcessarPagamento credito = new MeuPagamentoCredito();
        credito.debitar(240);
    }
}
```





## Para praticar...

---

- Pense em uma aplicação em que o padrão Adapter poderia ser usado. Em seguida, elabore a estrutura do padrão.



# Padrões de Projeto - Exemplos

---

- Exemplos:

Criação	Estrutural	Comportamental
<ul style="list-style-type: none"><li>• Abstract factory</li><li>• Builder</li><li>• Factory Method</li><li>• Prototype</li><li>• Singleton</li></ul>	<ul style="list-style-type: none"><li>• Adapter</li><li>• Bridge</li><li>• Composite</li><li>• Decorator</li><li>• Façade</li><li>• Flyweight</li><li>• Proxy</li></ul>	<ul style="list-style-type: none"><li>• Chain of responsibility</li><li>• Command</li><li>• Interpreter</li><li>• Iterator</li><li>• Mediator</li><li>• Memento</li><li>• <b>Observer</b></li><li>• Etc.</li></ul>



# Observer

---

- Lista de atributos usadas pelo livro GOF para a descrição dos padrões de projeto:

<ul style="list-style-type: none"><li>• <b>Nome</b></li><li>• <b>Intenção</b></li><li>• Motivação</li><li>• Aplicabilidade</li><li>• <b>Estrutura</b></li><li>• Participantes</li></ul>	<ul style="list-style-type: none"><li>• Colaborações</li><li>• Consequências</li><li>• <b>Implementação</b></li><li>• Exemplo de código</li><li>• Usos conhecidos</li><li>• Padrões relacionados</li></ul>
---	--



# Observer - Intenção

---

- Define uma dependência um para muitos entre objetos, de modo que, quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados.

Objeto observado

Métodos: adicionar, remover, notificar

Observador

Observador

Observador

...

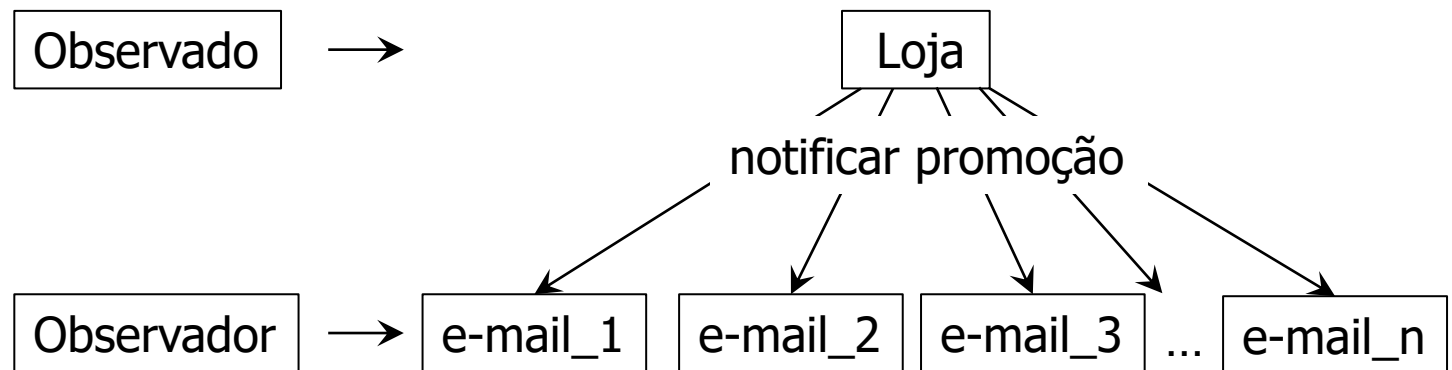
Observador

→ dependentes



# Padrão Observer - Ideia

- Funciona como uma *newsletter* (e-mail informativo) de um site.







# Sobre o Observer

---

- É um padrão de projeto comportamental (classes e objetos interagem e distribuem responsabilidades na aplicação) .
- Implementados com dois tipos de objetos: objetos observados (*observable*) e objetos observadores (*observer*).
- Objetos observados (*observable*) possuem referência para todos os seus observadores (*observer*).

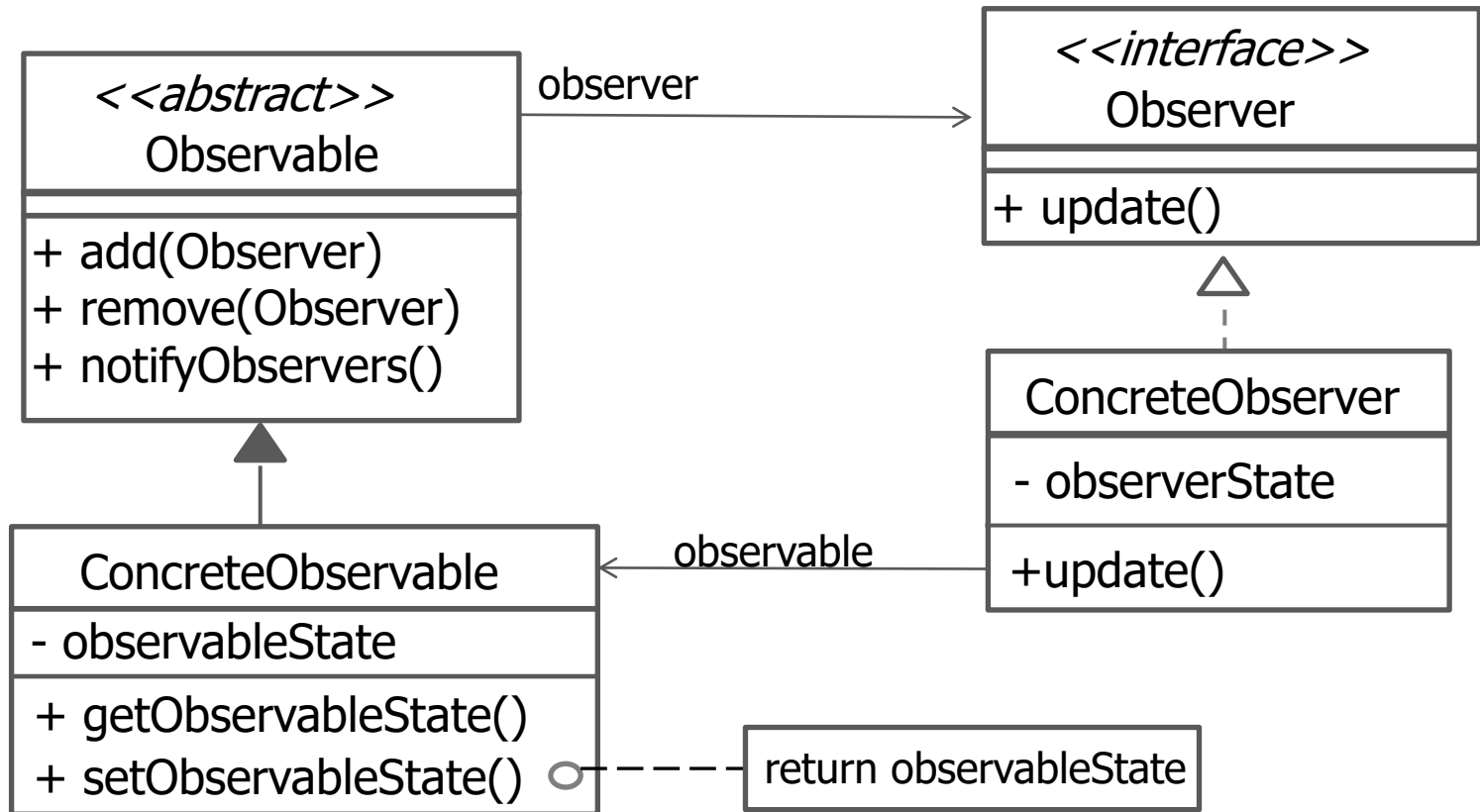


# Sobre o Observer

---

- Objetos observados (*observable*) podem adicionar, remover e notificar todos os observadores (*observer*) quando seu estado muda.
- Objetos observadores (*observer*) devem ter meios para receber as notificações de seu observado (*observable*). Normalmente, isso é feito por meio de um método.
- Utiliza o conceito de interface e classe abstrata.

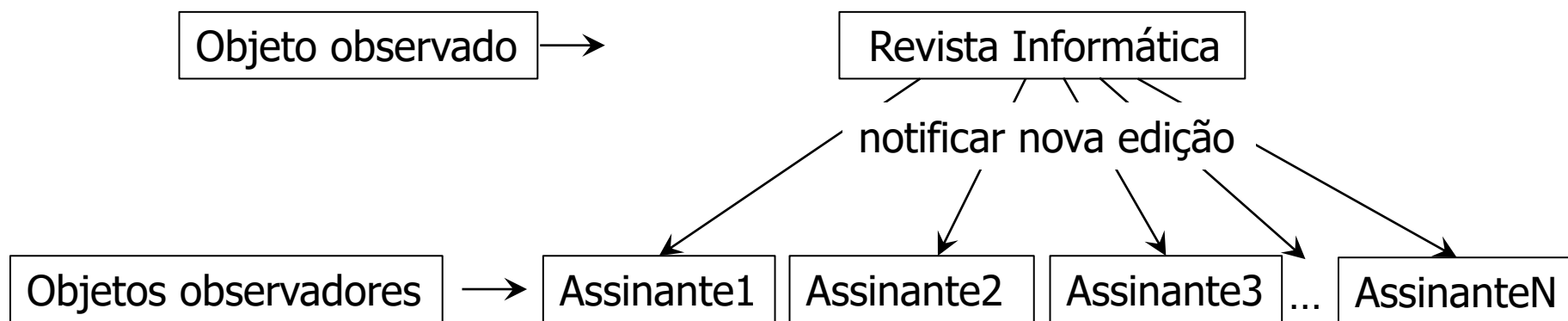
# Observer - Estrutura



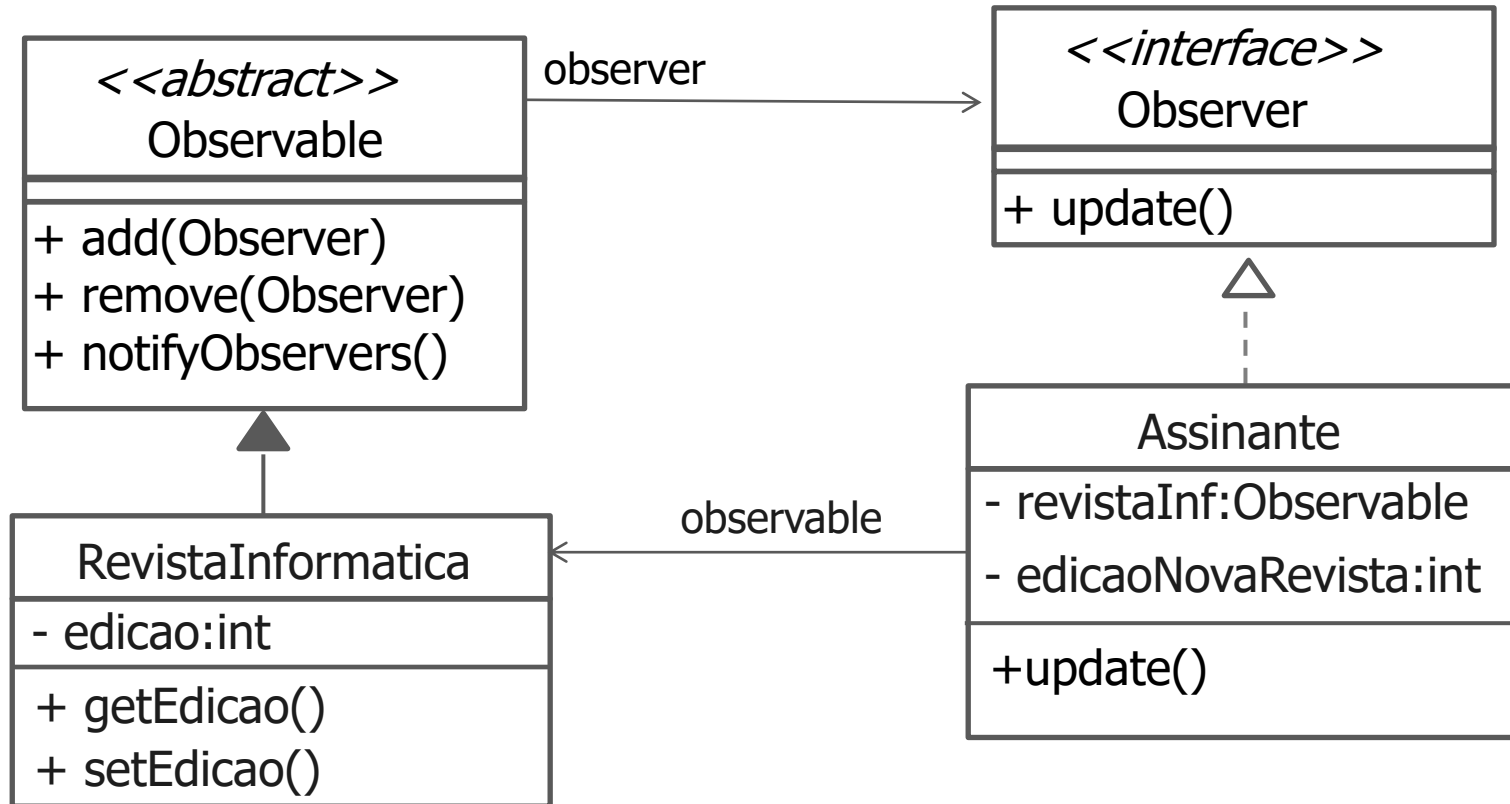


# Exemplo: Revista Informática

- Aplicação que notifica os assinantes de novas edições da Revista de Informática .



# Revista Informática - Observer



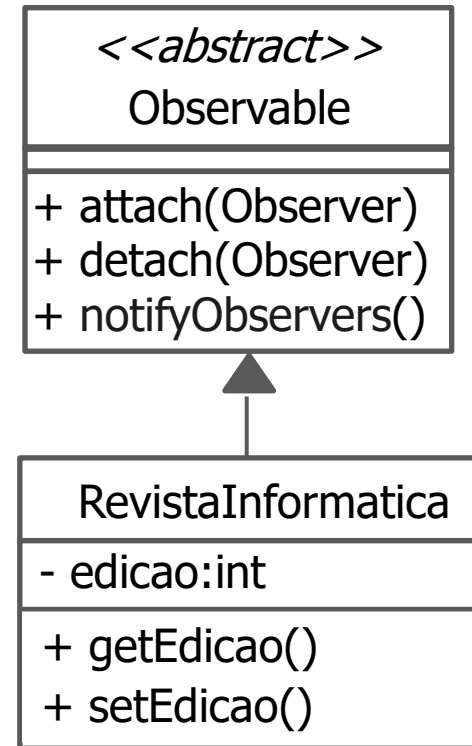
```

public class RevistaInformatica extends Observable
{
    private int edicao;

    public int getEdicao(){
        return this.edicao;
    }

    public void setEdicao (int novaEdicao){
        if(novaEdicao > 0)
        {
            this.edicao = novaEdicao;
            /* chamada do método para
            notificar os observadores */
            notifyObservers();
        }
    }
}

```



```
public class Assinante implements Observer
```

```
{
```

```
    private Observable revistaInf;
```

```
    private int edicaoNovaRevista;
```

```
    public Assinante(Observable revistaInfo){
```

```
        this.setRevistaInf(revistaInfo);
```

```
        revistaInf.add(this);
```

```
    }
```

```
// Métodos get/set
```

```
    public void update(Observable revistaInfo)
```

```
    {
        if(revistaInfo instanceof RevistaInformatica){
```

```
            RevistaInformatica rev = (RevistaInformatica) revistaInfo;
```

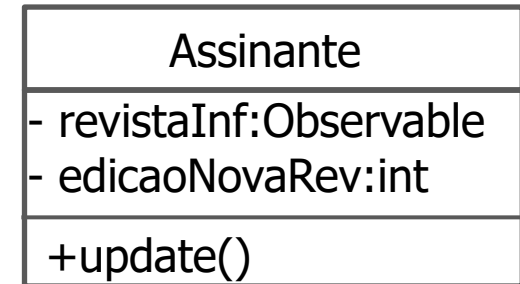
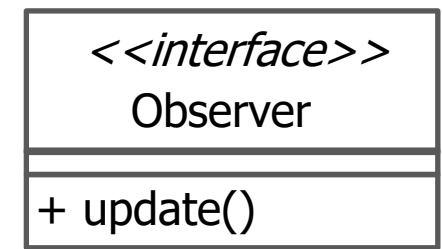
```
            this.edicaoNovaRevista = rev.getEdicao();
```

```
            S.o.p("Atenção! Edição " + edicaoNovaRevista + " disponível");
```

```
        }
```

```
    }
```

```
}
```



```
public class Principal
{
    public static void main(String []args)
    {
        int novaEdicao = 3;
        RevistaInformatica revista = new RevistaInformatica();
        Assinante assinante = new Assinante(revista);

        revista.setNovaEdicao(novaEdicao);
    }
}
```

A black computer monitor with a silver stand, displaying a white screen with black text. The text reads "Atenção! Edição 3 disponível!".

Atenção! Edição 3 disponível!





## Para praticar...

---

- Pense em uma aplicação em que o padrão Observer poderia ser usado. Em seguida, elabore a estrutura do padrão.



# Exemplo 1

---

- Que padrão de projeto utilizar?

Desenvolva uma aplicação para rede sociais que sinalize quando uma nova postagem é exibida no feed dos seus usuários. O feed é o local onde os usuários interagem e se mantêm atualizados com o conteúdo compartilhado por seus amigos e seguidores.



## Exemplo 2

---

- Que padrão de projeto utilizar?

Desenvolva uma aplicação para uma empresa de dispositivos eletrônicos personalizados. Essa empresa produz smarthpones, tablets e laptops, cada um com suas especificações. A empresa precisa de um mecanismo que facilite a produção desses dispositivos e garanta que as caraterísticas solicitadas pelos clientes sejam incorporadas.



## Exemplo 3

---

- Que padrão de projeto utilizar?

Desenvolva uma aplicação de streaming de música que permita que o usuário tenha acesso a diferentes serviços de música, como Spotify, Apple Music e YouTube Music usando uma única interface. Um streaming de música consiste na transmissão contínua de música pela Internet, sem que seja necessário baixar a música localmente.



## Para praticar...

---

- Monte a estrutura para os padrões de projeto mostrados nos Exemplos 1, 2 e 3.



# Refatoração (Refactoring)

---

- Processo de melhoria de código, sem que seja necessário criar novas funcionalidades.
- Objetivo: transformar um código mal feito/bagunçado em código limpo (simples, elegante e legível) .



# Técnicas de Refatoração

---

## 1) Código duplicado

- Princípio: toda informação deve ter um único endereço (“doença do copiar e colar”).
- O que fazer: separe o código duplicado e crie uma nova função ou classe.

## 2) Classes longas

- Uma classe não deve implementar mais de uma entidade ou algoritmo.
- O que fazer: isole os atributos e métodos que são afins e crie uma nova classe.



# Técnicas de Refatoração

---

## 3) Métodos longos

- Evite a escrita de métodos que ocupem mais de uma tela ou que realize mais de uma atividade.
- O que fazer: separe trechos que fazem atividades específicas e crie novos métodos.

## 4) Atributos da classe

- Atributos de classes nunca devem ser públicos.
- O que fazer: torne-os privados ou protegidos (no caso de herança) e crie métodos get/set. Se um atributo nunca for ser alterado, crie apenas o métodos get.





# Técnicas de Refatoração

---

## 5) Comentários

- Evite comentários redundantes ou para explicar o que o código faz (o código deve dizer).
- O que fazer: utilize comentários para explicar por que você tomou uma decisão ao invés de outra.



# Técnicas de Refatoração

---

## 6) Métodos com muitos parâmetros

- Métodos com 5, 6 ou mais parâmetros são confusos e indicam que não foi modelado.
- O que fazer: verifique se não é possível criar uma classe para representar os parâmetros.
  - Método:
    - cadastraPessoa("Jose", "Rua X", 23, "555-1111", etc)
  - Solução: criar classes separadas como Pessoa, Endereço



# Técnicas de Refatoração

---

## 7) Expressões complexas

- Não deixe expressões complexas soltas no código.
- O que fazer: crie métodos cujo nome digam o que a expressão faz.

➤ Exemplo, ao invés de:

```
if(ano % 4 == 0 && (ano % 100 != 0 || ano % 400 == 0)){...}
```

→ Que tal usar:

```
if(ehBissextto(ano)){...}
```