



Padrões de Projeto (cont.)

Prof^a. Rachel Reis
rachel@inf.ufpr.br



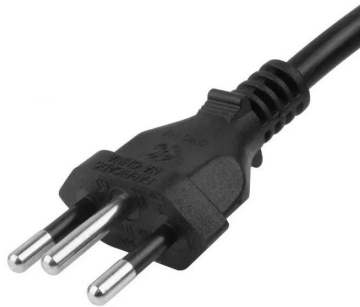
Padrões de Projeto - Exemplos

- Exemplos:

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



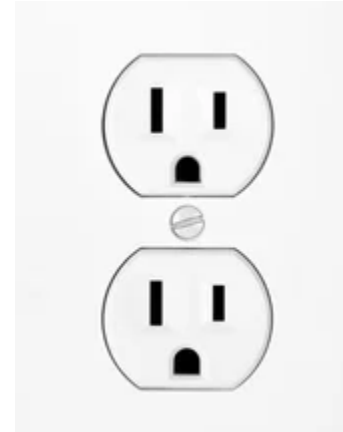
Adapter - Analogia



Tomada de notebook



Adaptador



Tomada



Sobre o Adapter

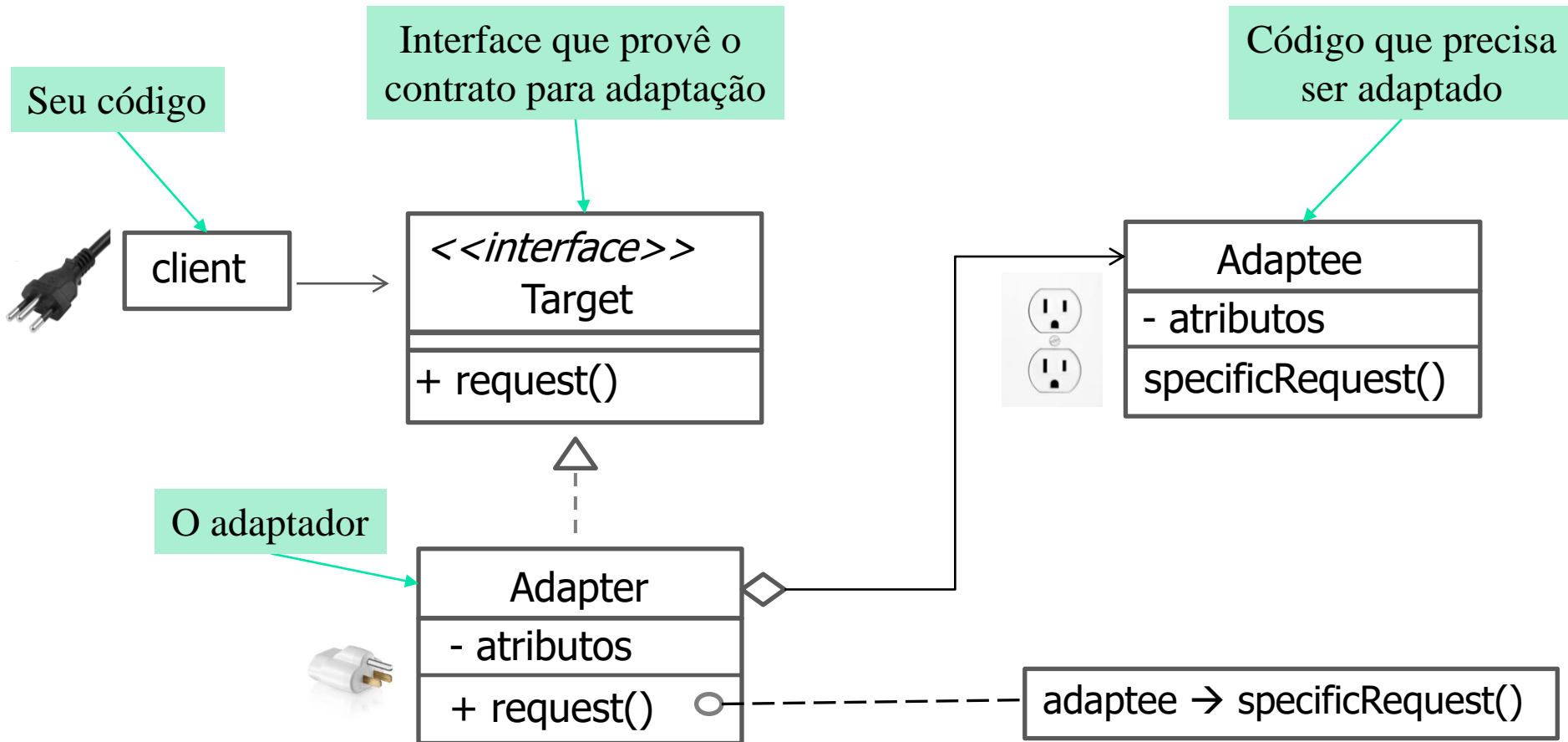
- É um padrão de projeto estrutural (organiza classes e objetos para criar estruturas maiores) também conhecido como *wrapper*.
- Faz exatamente o que um adaptador da vida real faz.
- Evita a dependência com códigos externos.
- Utiliza o conceito de interface.



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.

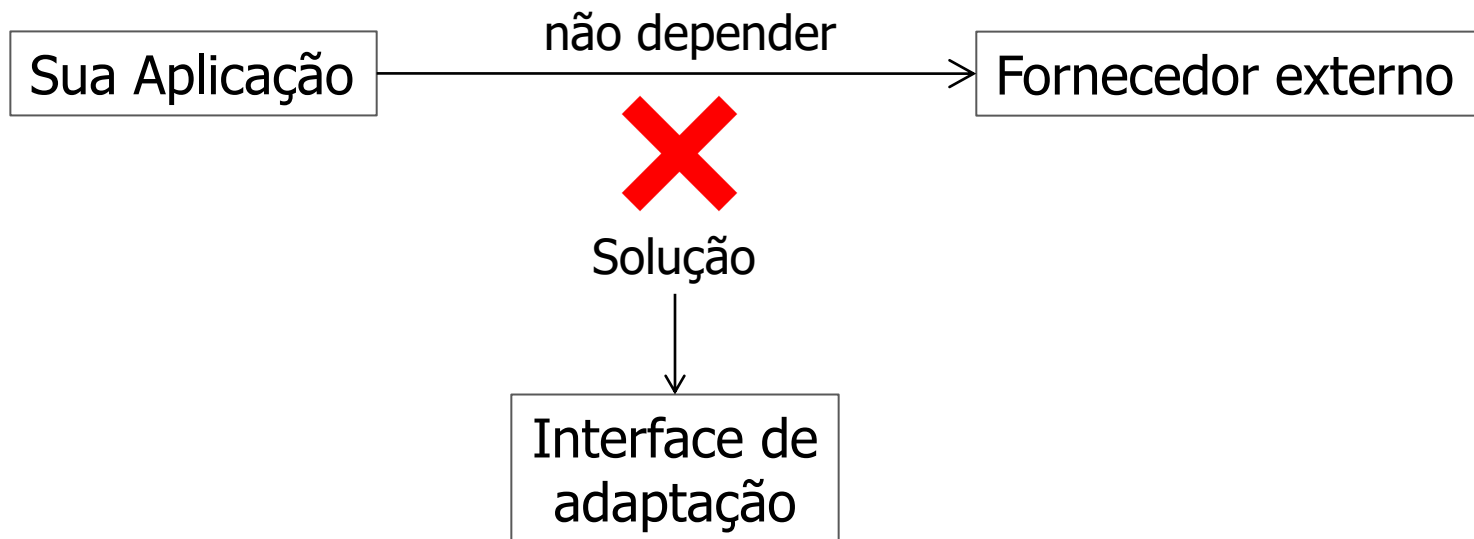
Adapter - Estrutura





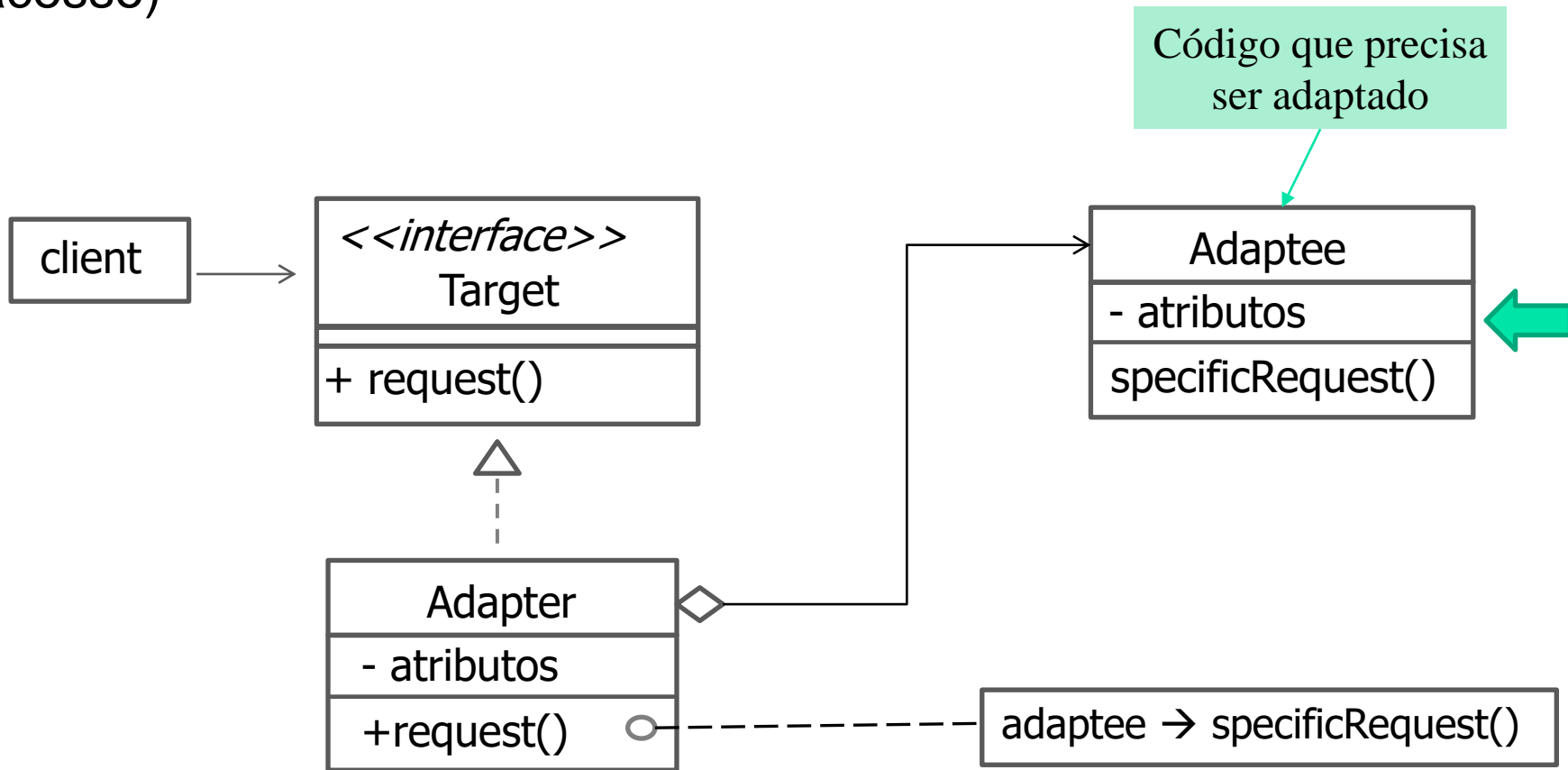
Exemplo: Aplicação de Pagamento

- Aplicação que oferece a opção de pagamento no cartão de crédito.



Aplicação - Estrutura

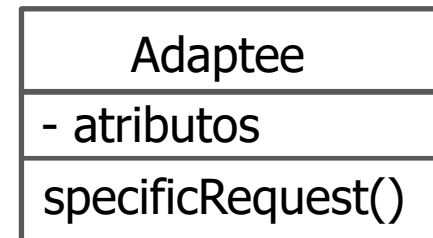
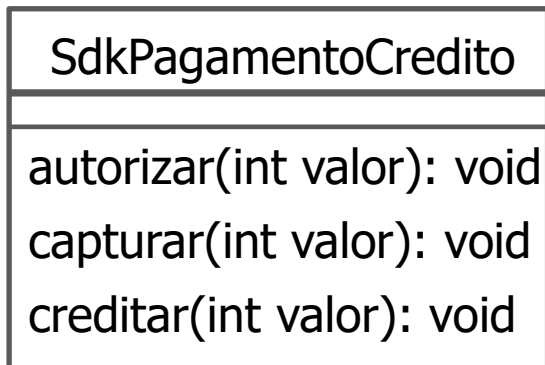
- Classe Adaptee: classe do fornecedor (você não tem acesso)






Exemplo: Aplicação de Pagamento

- Classe do fornecedor (você não tem acesso)



Código que precisa
ser adaptado



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

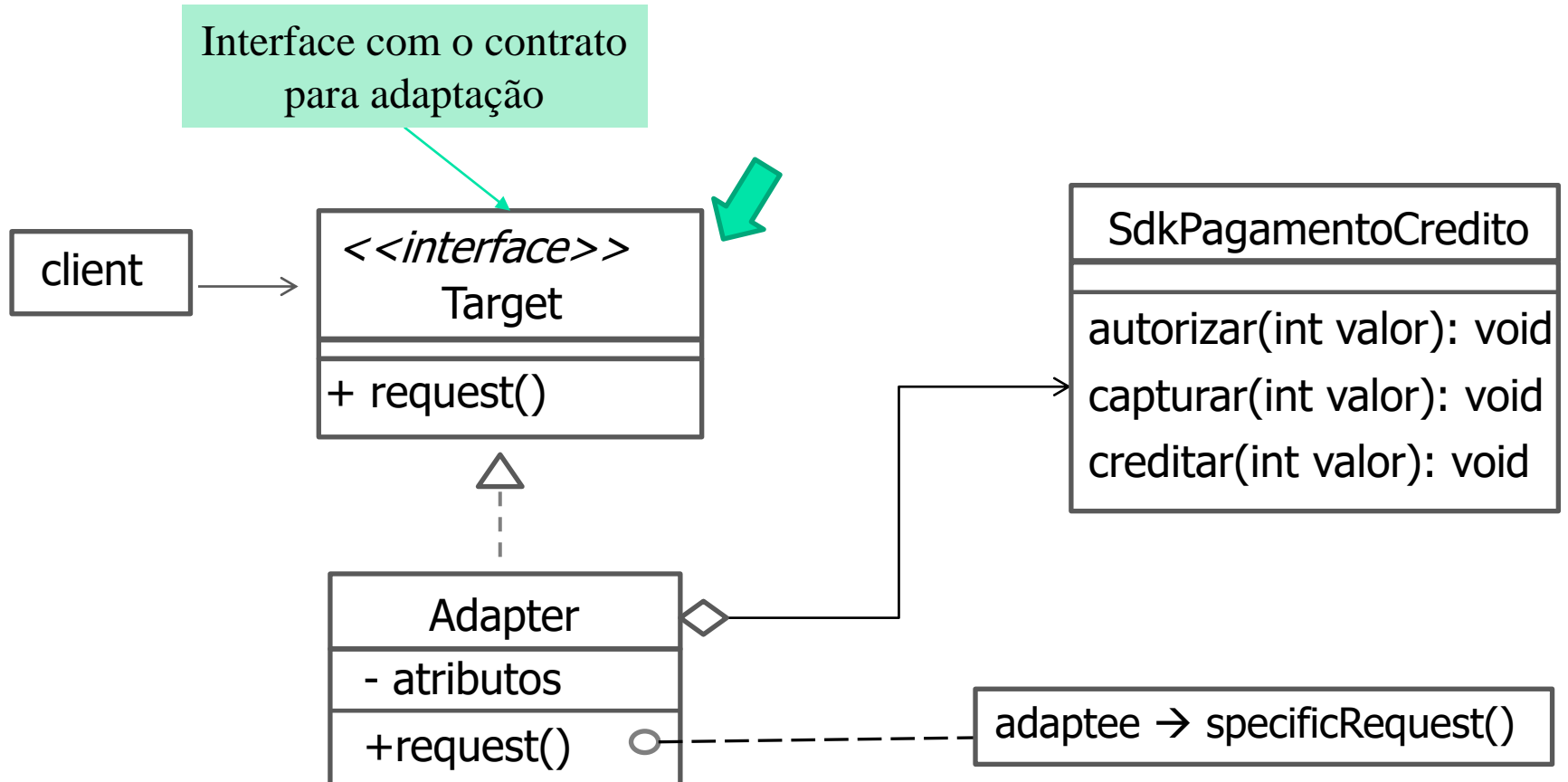
    public void capturar(int valor){
        // efetuar a cobrança
    }

    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.

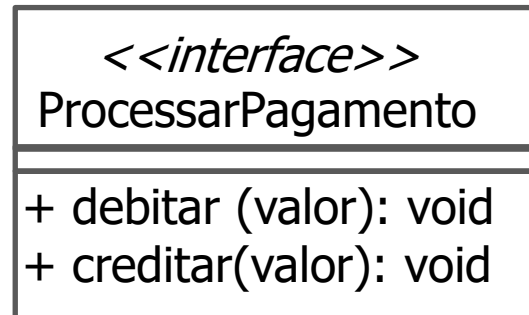
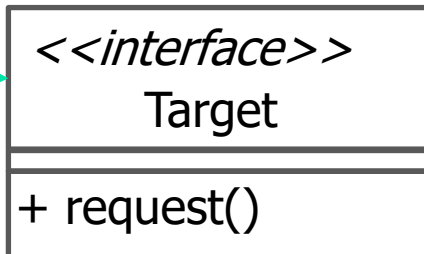




Exemplo: Aplicação de Pagamento

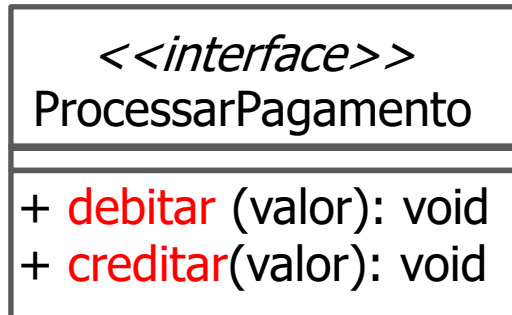
- Interface Target: provê os métodos para adaptação.

Interface que meu
código precisa



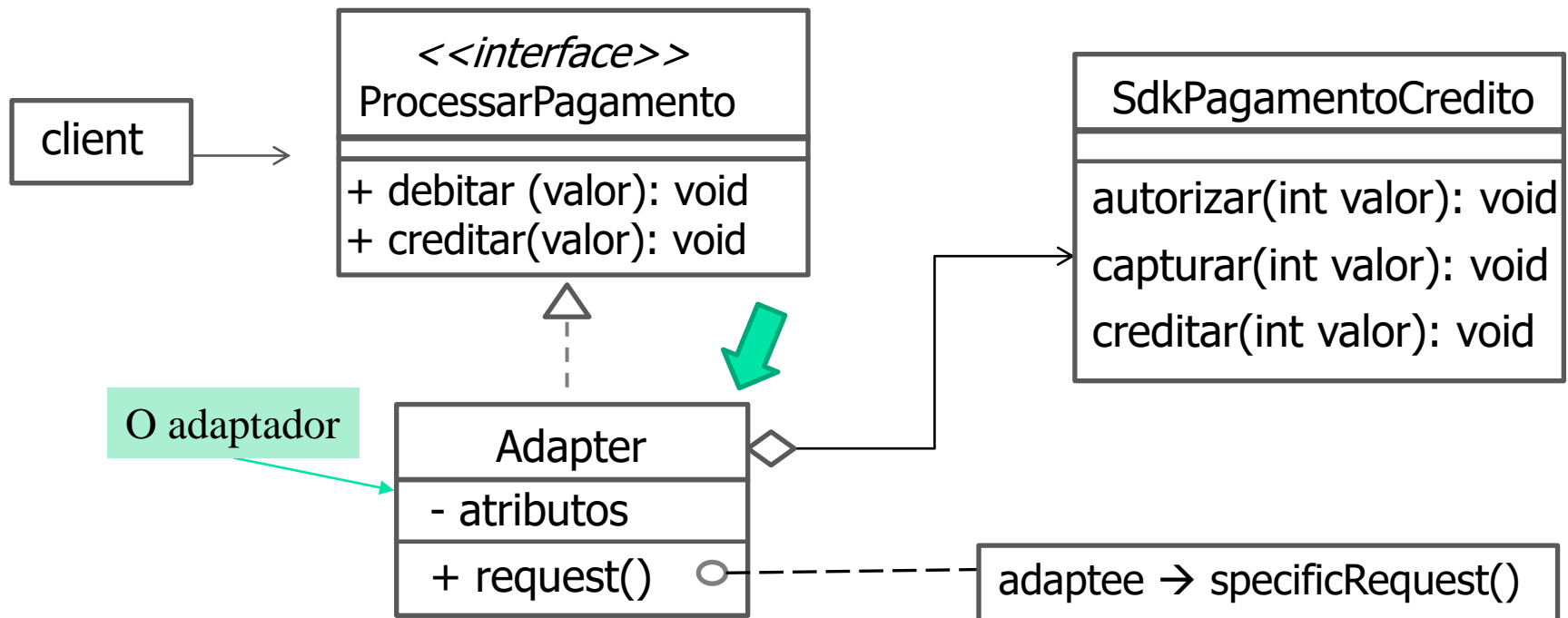
```
public interface ProcessarPagamento
{
    public void debitar(int valor);

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



Aplicação de Pagamento - Adapter

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

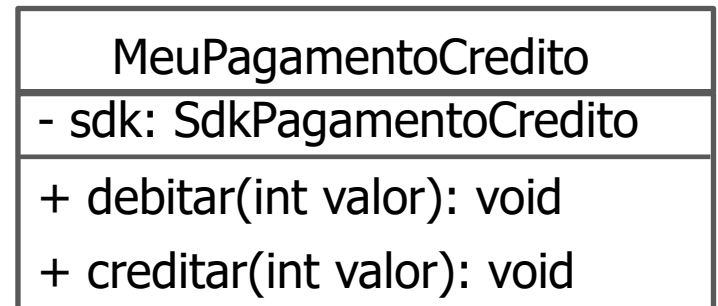
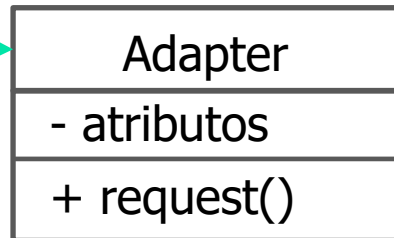




Exemplo: Aplicação de Pagamento

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

O adaptador



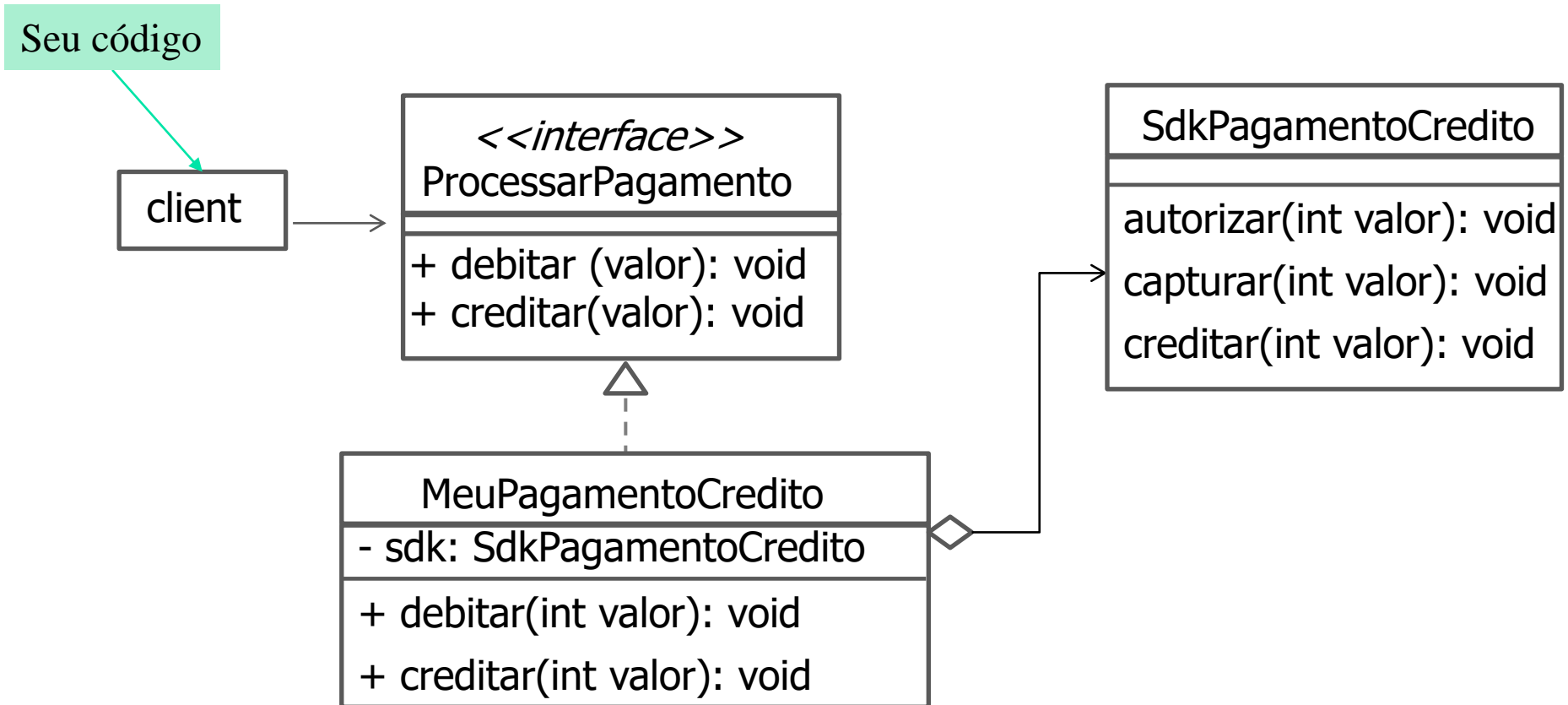
```
public class MeuPagamentoCredito implements ProcessadorPagamento
{
    private SdkPagamentoCredito sdk = new SdkPagamentoCredito();

    public void debitar(int valor){
        sdk.autorizar(valor);
        sdk.capturar(valor);
    }

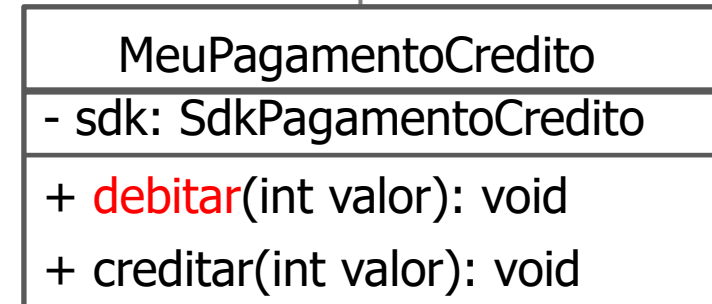
    public void creditar(int valor){
        sdk.creditar(valor);
    }
}
```

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

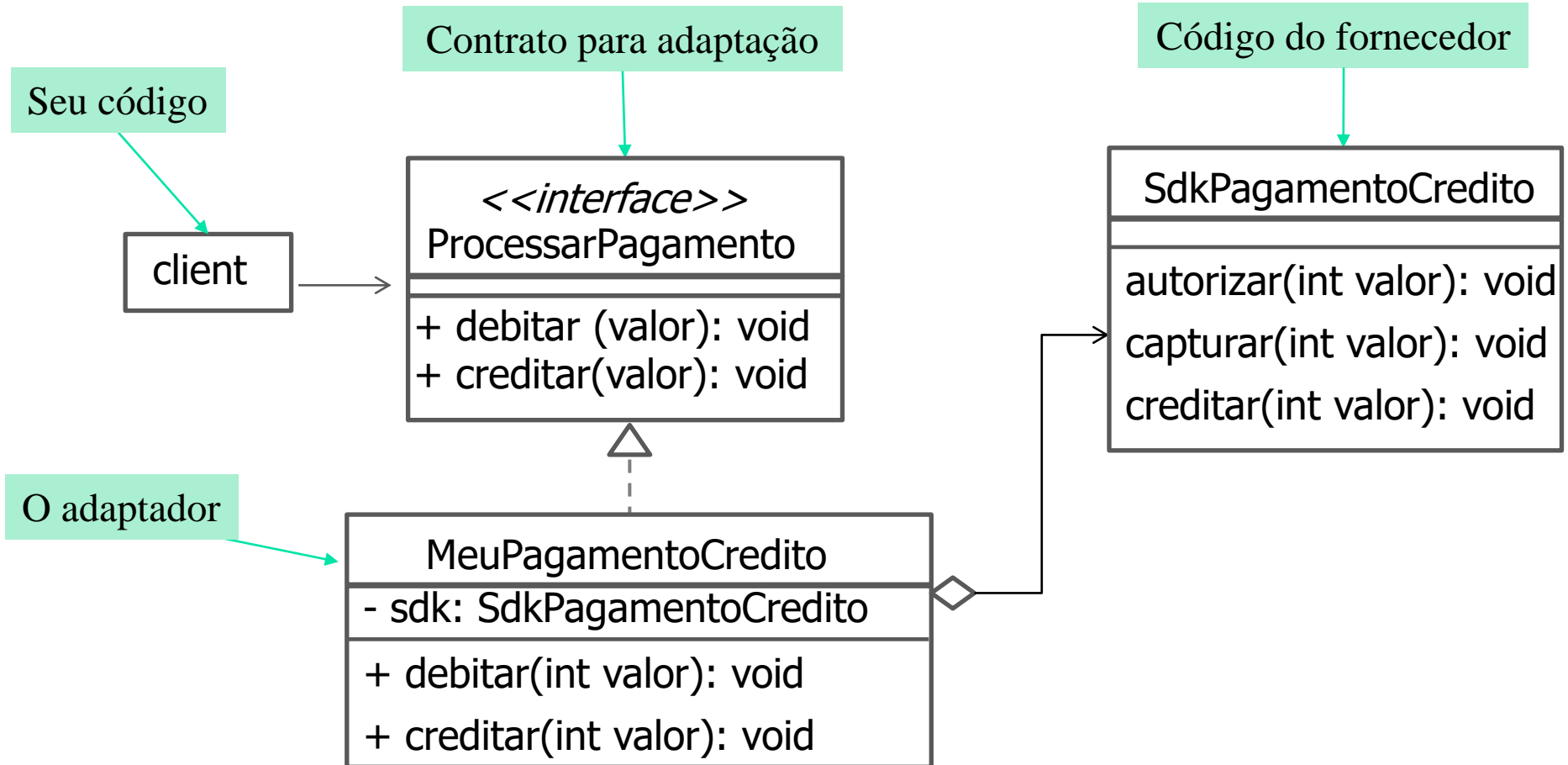
Aplicação de Pagamento - Adapter



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



Aplicação de Pagamento - Adapter





Padrões de Projeto - Exemplos

- Exemplos:

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



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 ...

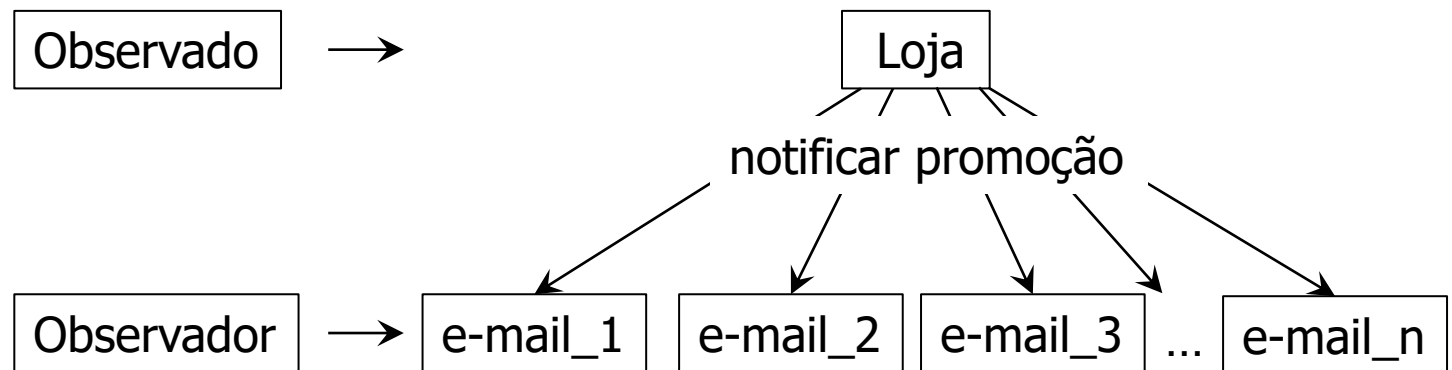
Observador

→ dependentes



Padrão Observer - Ideia

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





Sobre o Observer

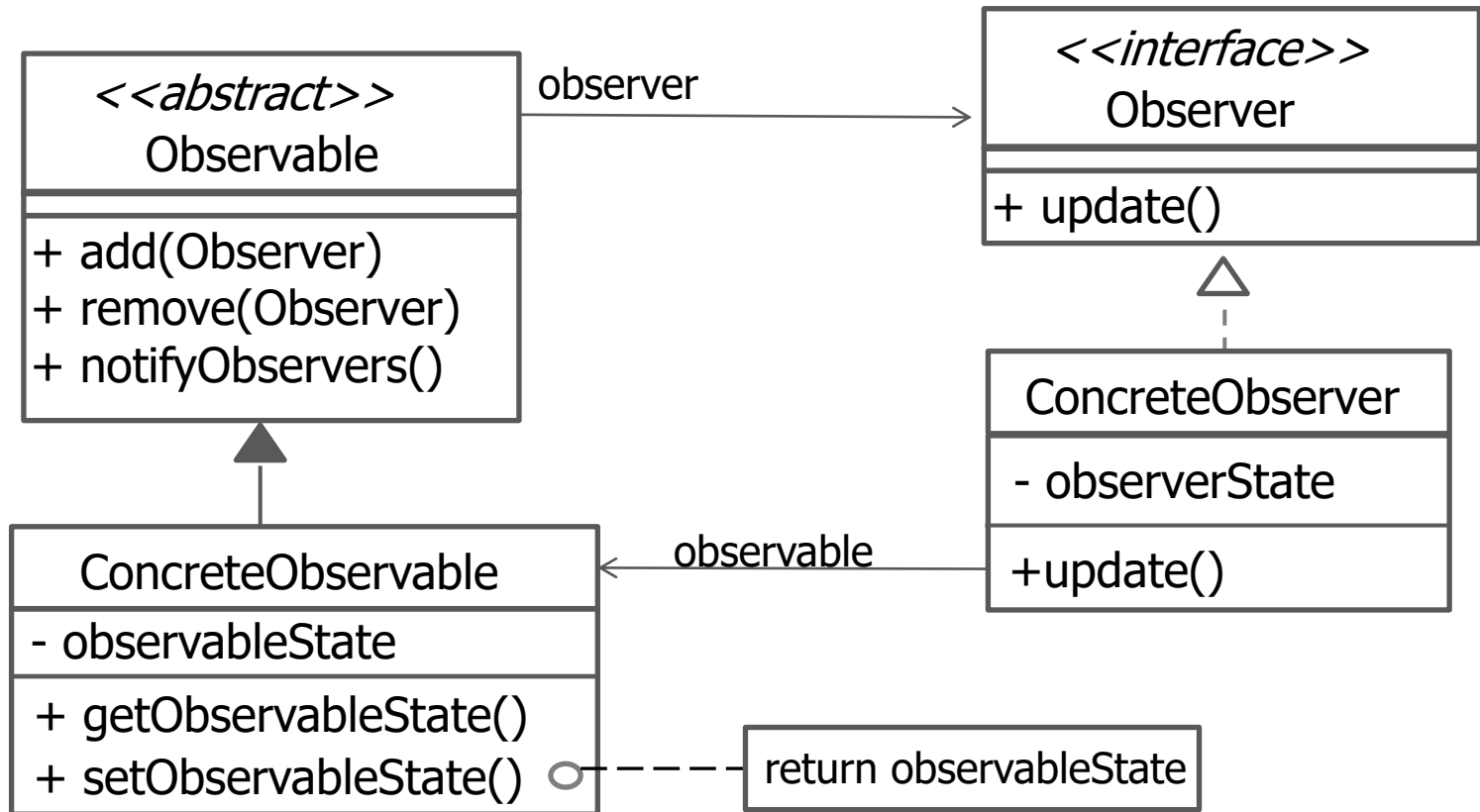
- É um padrão de projeto comportamental (classes e objetos interagem e distribuem responsabilidades na aplicação) .
- Implementados com dois tipos de objetos: objetos observados (*observable*) e objetos observadores (*observer*).
- Objetos observados (*observable*) possuem referência para todos os seus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

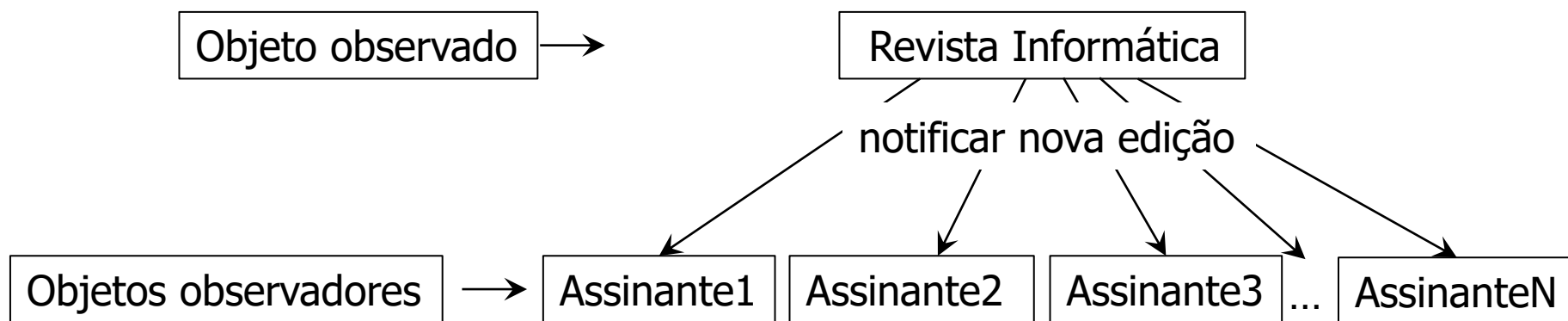
Observer - Estrutura



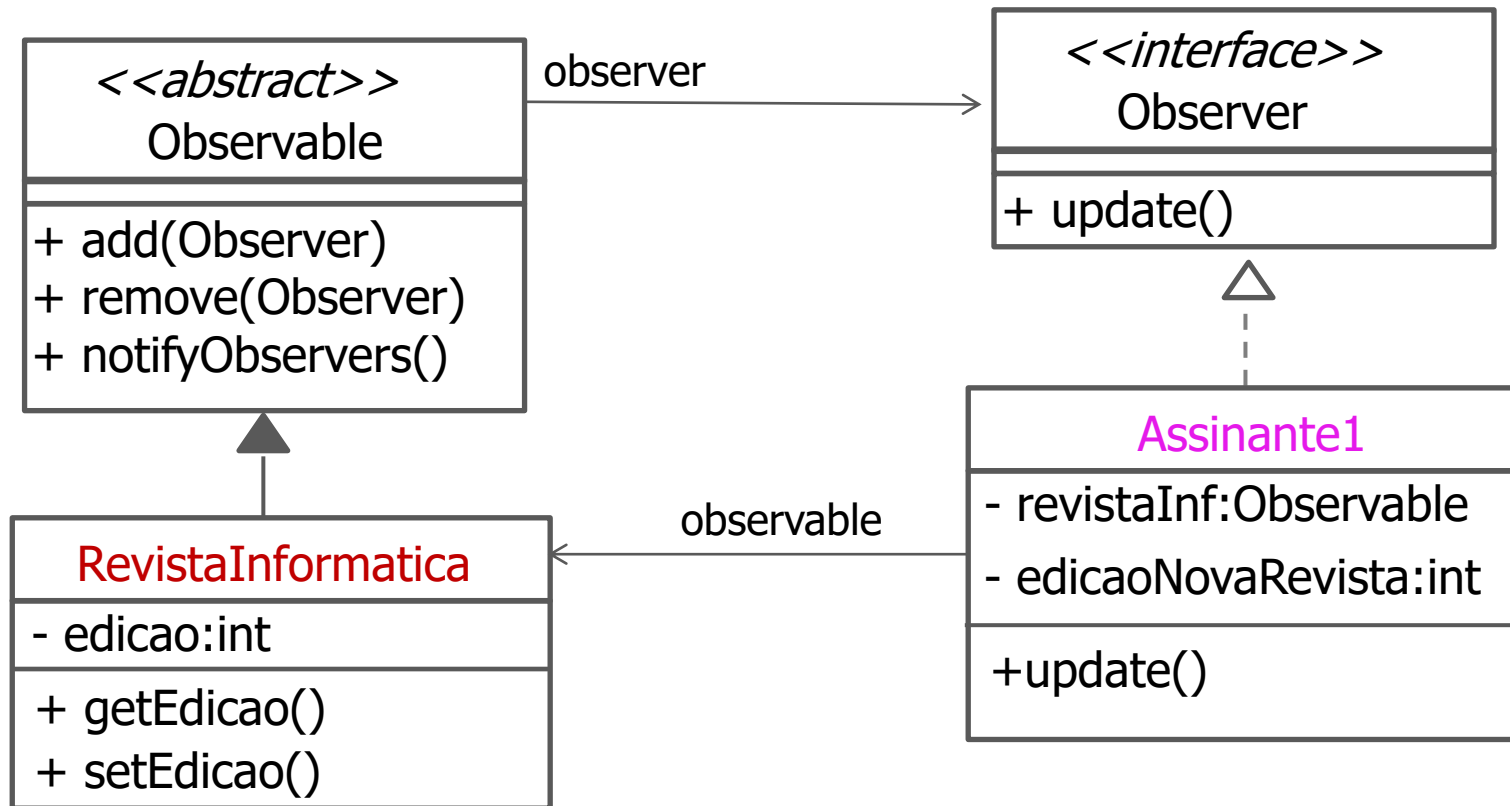


Exemplo: Revista Informática

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



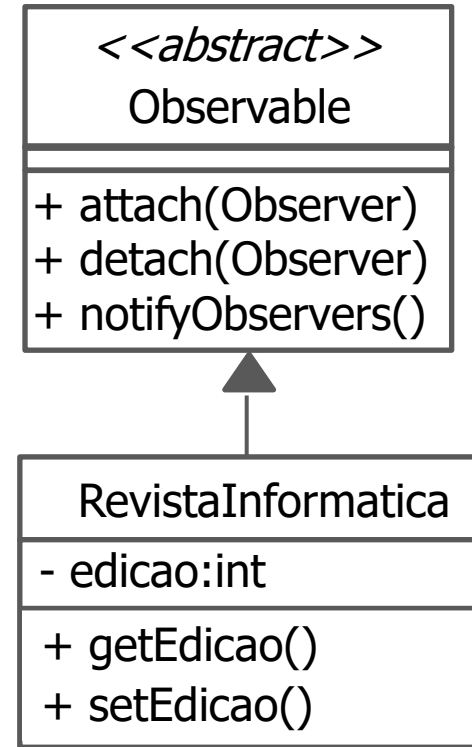
Revista Informática - Observer



```
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 Assinante1 implements Observer
```

```
{
```

```
    Observable revistaInf;
```

```
    int edicaoNovaRevista;
```

```
    public Assinante1(Observable revistaInfo){
```

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

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

```
    }
```

```
// Implementar métodos get/set
```

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

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

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

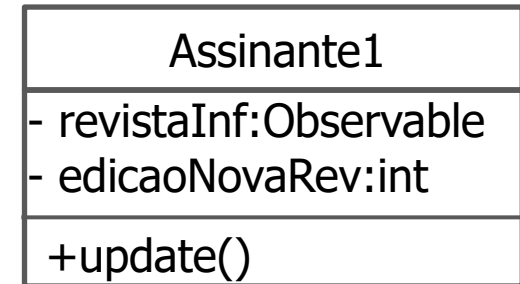
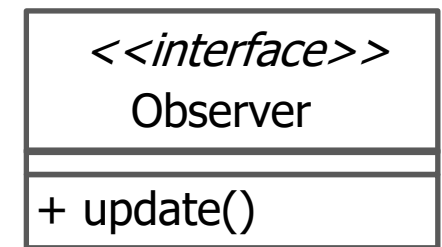
```
            this.edicaoNovaRevista = rev.getEdicao();
```

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

```
        }
```

```
    }
```

```
}
```



```
public class Principal
{
    public static void main(String []args)
    {
        int novaEdicao = 3;
        RevistaInformatica revistaInf = new RevistaInformatica();
        Assinante1 assinante1 = new Assinante1(revistaInf);

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



Para praticar...

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



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 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

4) 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.

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) Expressões complexas

- Não deixe expressões complexas soltas no código.
- O que fazer: crie funções 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)){...}
```



Técnicas de Refatoração

7) 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.