

Padrões de Projeto

Prof. Marum Simão Filho

Agenda

- Padrão Iterator
- Padrão Proxy
- Padrão State



Padrão Iterator

- Motivação:
 - Dois restaurantes diferentes.
 - Deseja-se unificar as empresas.
 - Cada uma possui um cardápio diferente.
 - Uma especializada em Cafés da Manhã
 - A Panquecaria
 - Outra especializada em Almoços
 - O Restaurante



2 Coleções de Objetos

Menu Restaurante

Sanduiche Vegetariano 2,99
Alface e Pão Integral

Sopa do dia 2,80
Tigela de sopa com torradas

Cachorro Quente 1,50
Salsicha com molho e queijo

Legumes Cozidos 2,50
Mistura de legumes

MENU PANQUECARIA

Desjejum de Panqueca 2,99
Panquecas com ovos mexidos

Desjejum Tradicional 2,99
Panquecas com ovos fritos e salsicha

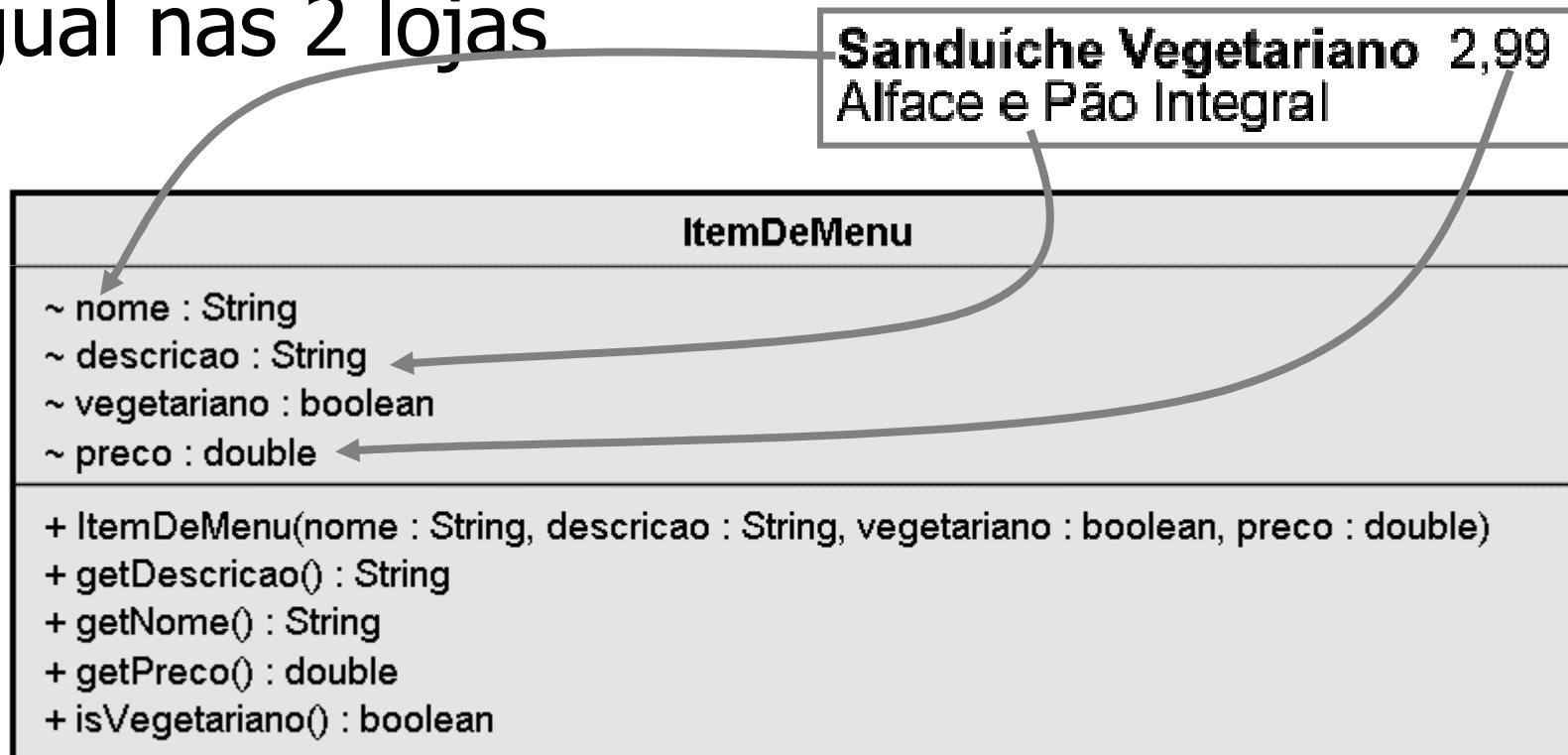
Panquecas Doces 3,49
Panquecas com geléia de amora

Waffles 3,59
Waffles com geléia de mocotó

ItemDeMenu

O Item de Menu

- A classe que representa cada item de menu é igual nas 2 lojas



Os Menus Semelhantes

- Os menus são semelhantes

| | |
|-----------------------------|------|
| <i>Menu Restaurante</i> | |
| Sanduche Vegetariano | 2,99 |
| Alface e Pão Integral | |
| Sopa do dia | 2,80 |
| Tigela de sopa com torradas | |
| Cachorro Quente | 1,50 |
| Salsicha com molho e queijo | |
| Legumes Cozidos | 2,50 |
| Mistura de legumes | |

ArrayList<ItemDeMenu>

| | |
|--------------------------------------|------|
| MENU PANQUECARIA | |
| Desjejum de Panqueca | 2,99 |
| Panquecas com ovos mexidos | |
| Desjejum Tradicional | 2,99 |
| Panquecas com ovos fritos e salsicha | |
| Panquecas Doces | 3,49 |
| Panquecas com geléia de amora | |
| Waffles | 3,59 |
| Waffles com geléia de mocotó | |

ItemDeMenu[]

- Diferença
 - A armazenagem dos objetos ItemDeMenu

Menu Restaurante

Sanduíche Vegetariano 2,99
Alface e Pão Integral

Sopa do dia 2,80
Tigela de sopa com torradas

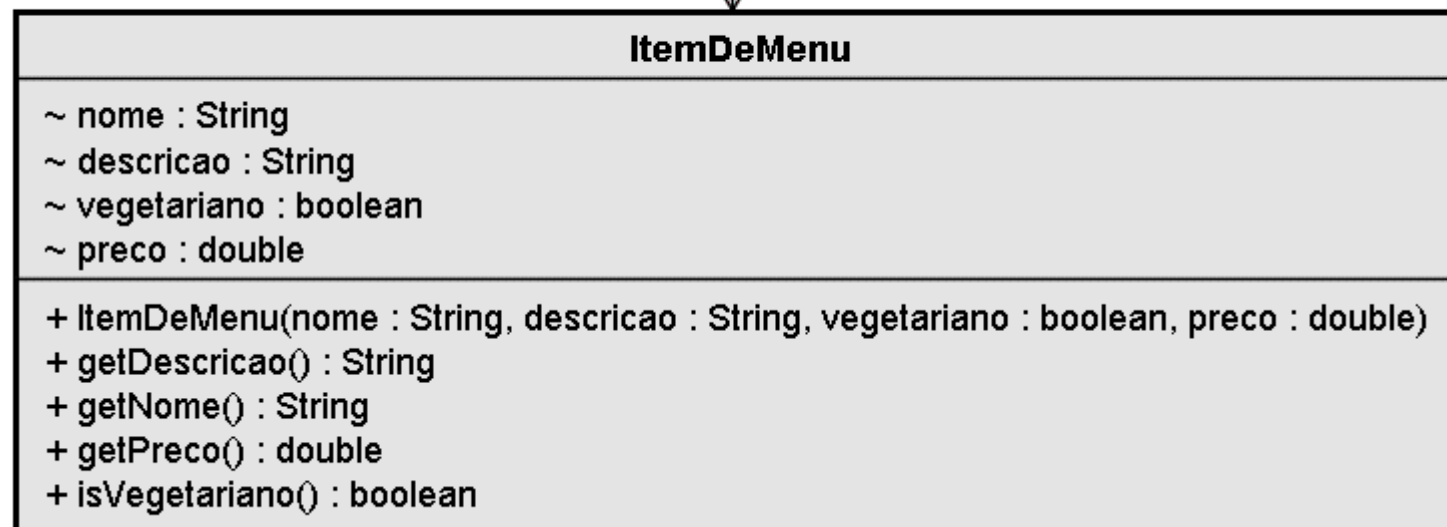
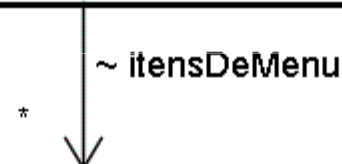
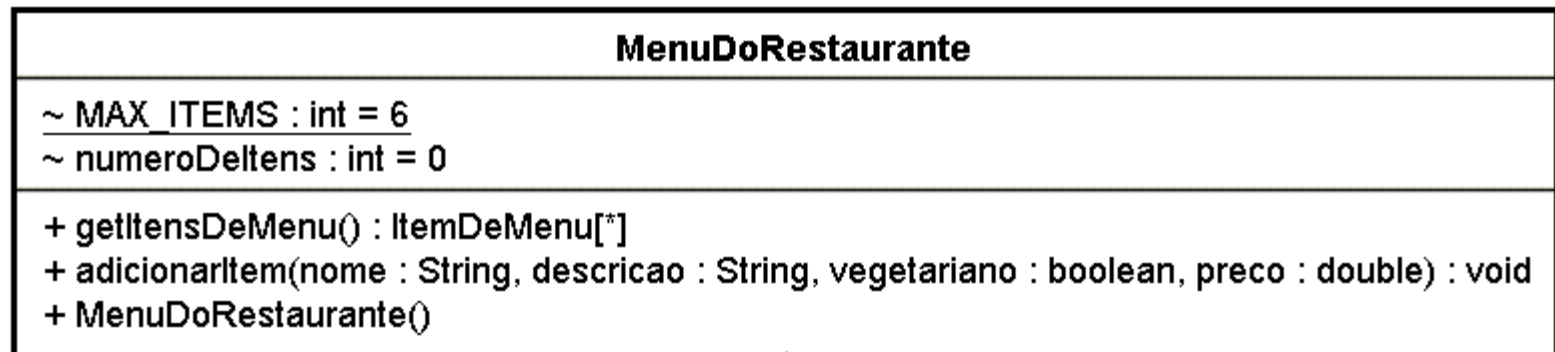
Cachorro Quente 1,50
Salsicha com molho e queijo

Legumes Cozidos 2,50
Mistura de legumes



ItemDeMenu[]

O Menu do Restaurante



MenuDoRestaurante.java

```
public class MenuDoRestaurante {
    static final int MAX_ITEMS = 6;
    int numeroDeItens = 0;
    ItemDeMenu[] itensDeMenu;

    public ItemDeMenu[] getItensDeMenu() {
        return itensDeMenu;
    }

    public void adicionarItem(
        String nome, String descricao, boolean vegetariano, double preco) {
        ItemDeMenu itemDeMenu =
            new ItemDeMenu(nome, descricao, vegetariano, preco);
        if (numeroDeItens >= MAX_ITEMS) {
            System.out.println("Menu está cheio");
        } else {
            itensDeMenu[numeroDeItens] = itemDeMenu;
            numeroDeItens++;
        }
    }

    public MenuDoRestaurante() {
        itensDeMenu = new ItemDeMenu[MAX_ITEMS];
        adicionarItem("Canja", "Canja", false, 3.99);
        adicionarItem("Waffles", "Waffles", true, 3.59);
    }
}
```



| | |
|-----------------------------|------|
| <i>Menu Restaurante</i> | |
| Sanduíche Vegetariano | 2,99 |
| Alface e Pão Integral | |
| Sopa do dia | 2,80 |
| Tigela de sopa com torradas | |
| Cachorro Quente | 1,50 |
| Salsicha com molho e queijo | |
| Legumes Cozidos | 2,50 |
| Mistura de legumes | |

ItemDeMenu[]

MENU PANQUECARIA

Desjejum de Panqueca 2,99
Panquecas com ovos mexidos

Desjejum Tradicional 2,99
Panquecas com ovos fritos e salsicha

Panquecas Doces 3,49
Panquecas com geléia de amora

Waffles 3,59
Waffles com geléia de mocotó



ArrayList<ItemDeMenu>

O Menu da Panquecaria

MenuDaPanquecaria

- itensDeMenu : ArrayList<ItemDeMenu>


+ getItensDeMenu() : ArrayList<ItemDeMenu>

+ adicionarItem(nome : String, descricao : String, vegetariano : boolean, preco : double) : void

+ MenuDaPanquecaria()

MenuDaPanquecaria.java

```
public class MenuDaPanquecaria {  
  
    ArrayList itensDeMenu;  
  
    public ArrayList getItensDeMenu() {  
        return itensDeMenu;  
    }  
  
    public void adicionarItem(  
        String nome, String descricao, boolean vegetariano, double preco) {  
        ItemDeMenu itemDeMenu = new ItemDeMenu(nome, descricao, vegetariano, preco);  
        itensDeMenu.add(itemDeMenu);  
    }  
  
    public MenuDaPanquecaria() {  
        itensDeMenu = new ArrayList();  
        adicionarItem("Panqueca Café da Manhã", "Panqueca com ovos", true, 2.99);  
        adicionarItem("Waffles", "Waffles", true, 3.59);  
    }  
}
```



ArrayList<ItemDeMenu>



Contrata-se Garçonete



■ Requisitos

- Deve ser capaz de interagir com menus em Java de diversas naturezas e apresentá-los.

■ O Problema

- Ela deve saber trabalhar com Array e ArrayList.

Garçonete

```
printMenu()  
printMenuRestaurante()  
printMenuPanquecaria()
```



MENU PANQUECARIA

Desjejum de Panqueca 2,99
Panquecas com ovos mexidos

Desjejum Tradicional 2,99
Panquecas com ovos fritos e salsicha

Panquecas Doces 3,49
Panquecas com geléia de amora

Waffles 3,59
Waffles com geléia de mocotó



ArrayList<ItemDeMenu>

O que ela deve saber fazer

```
void printMenuDaPanquecaria() {  
    MenuDaPanquecaria menu = new MenuDaPanquecaria();  
    ArrayList itensPanquecaria = menu.getItemsDeMenu();  
  
    for (int i = 0; i < itensPanquecaria.size(); i++) {  
        ItemDeMenu item =  
            (ItemDeMenu) itensPanquecaria.get(i);  
        System.out.print(item.getNome() + " ");  
        System.out.print(item.getPreco() + " ");  
        System.out.println(item.getDescricao());  
    }  
}
```



O que ela deve saber fazer

```
void printMenuDoRestaurante() {  
    MenuDoRestaurante menu = new MenuDoRestaurante();  
    ItemDeMenu[] itensDoRestaurante =  
        menu.getItensDeMenu();  
  
    for (int i = 0; i < itensDoRestaurante.length; i++) {  
        ItemDeMenu item = itensDoRestaurante[i];  
        System.out.print(item.getNome() + " ");  
        System.out.print(item.getPreco() + " ");  
        System.out.println(item.getDescricao());  
    }  
}
```

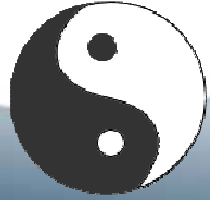
| Menu Restaurante | |
|-----------------------------|------|
| Sanduíche Vegetariano | 2,99 |
| Alface e Pão Integral | |
| Sopa do dia | 2,80 |
| Tigela de sopa com torradas | |
| Cachorro Quente | 1,50 |
| Salsicha com molho e queijo | |
| Legumes Cozidos | 2,50 |
| Mistura de legumes | |

+ ItemDeMenu[]



Breve análise Verdadeiro ou Falso?

- V** Estamos codificando para **implementações concretas** de MenuDoRestaurante e MenuDaPanquecaria e **não para uma interface**.
- V** Se quiséssemos passar a usar um **outro tipo de Menu** que implemente sua lista de itens através de Hashtable, teríamos que **modificar** o código da Garçonete.
- V** A **Garçonete precisa saber** como cada menu representa a sua coleção interna de itens, o que **viola o encapsulamento**.
- V** Temos **código duplicado**: o método printMenu() exige **dois laços separados** para iterar através dos dois tipos de menus. Se acrescentarmos um terceiro menu, teríamos mais um laço.



Relembrando o 1.º Princípio de Design

“Identifique os aspectos de seu aplicativo
que **variam** e **separe-os**
do que permanece igual”

O quê está variando?

Menu Restaurante

Sanduche Vegetariano 2,99
Alface e Pão Integral

Sopa do dia 2,80
Tigela de sopa com torradas

Cachorro Quente 1,50
Salsicha com molho e queijo

Legumes Cozidos 2,50
Mistura de legumes



ItemDeMenu[]

MENU PANQUECARIA

Desjejum de Panqueca 2,99
Panquecas com ovos mexidos

Desjejum Tradicional 2,99
Panquecas com ovos fritos e salsicha

Panquecas Doces 3,49
Panquecas com geleia de amora

Waffles 3,59
Waffles com geleia de mocotó



ArrayList<ItemDeMenu>

Garconete

printMenu()

printMenuRestaurante()

printMenuPanquecaria()

MENU PANQUECARIA

Desjejum de Panqueca 2,99
Panquecas com ovos mexidos

Desjejum Tradicional 2,99
Panquecas com ovos fritos e salsicha

Panquecas Doces 3,49
Panquecas com geleia de amora

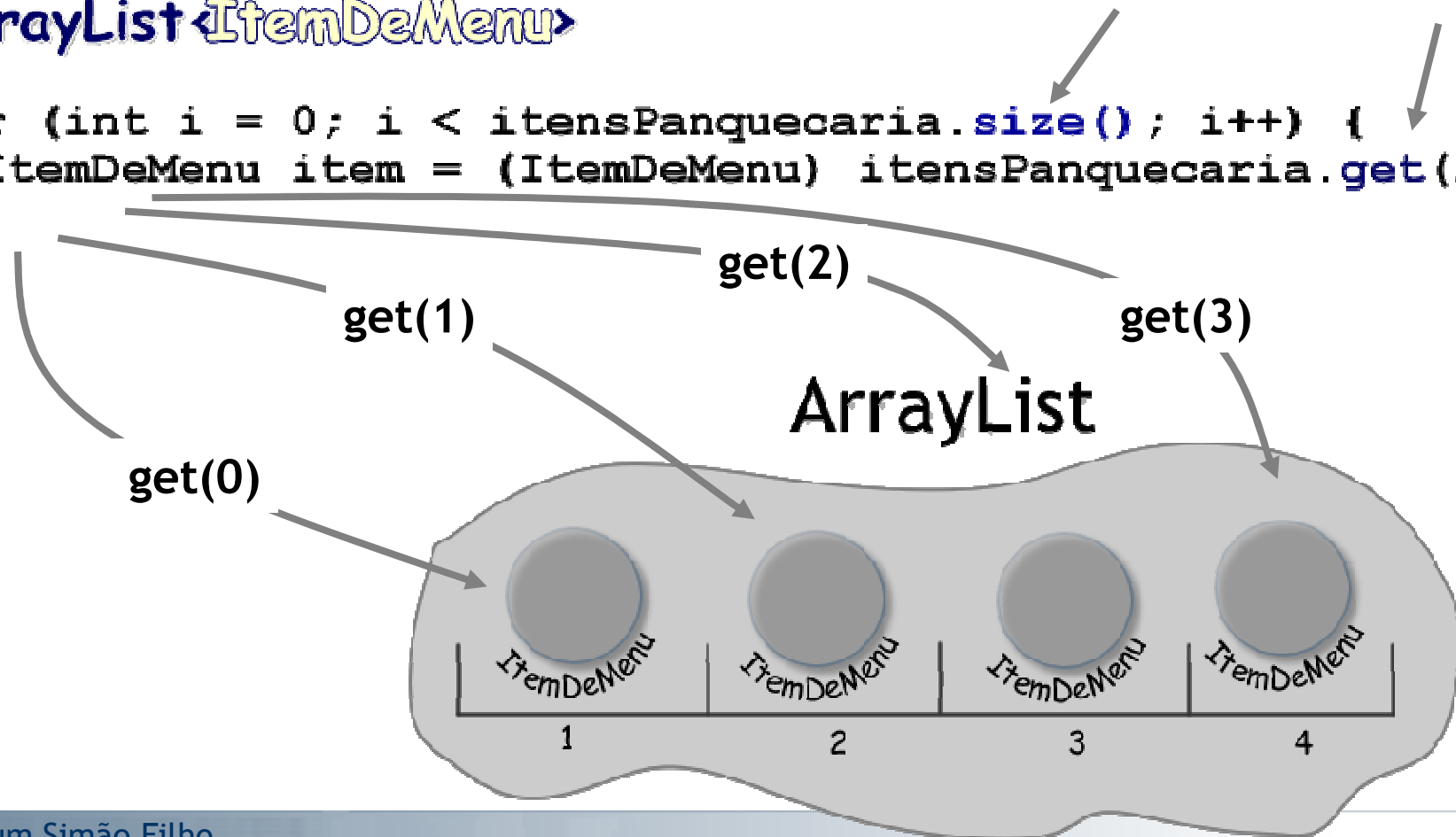
Waffles 3,59
Waffles com geleia de mocotó

A iteração



ArrayList<ItemDeMenu>

```
for (int i = 0; i < itensPanquecaria.size(); i++) {  
    ItemDeMenu item = (ItemDeMenu) itensPanquecaria.get(i);  
}
```



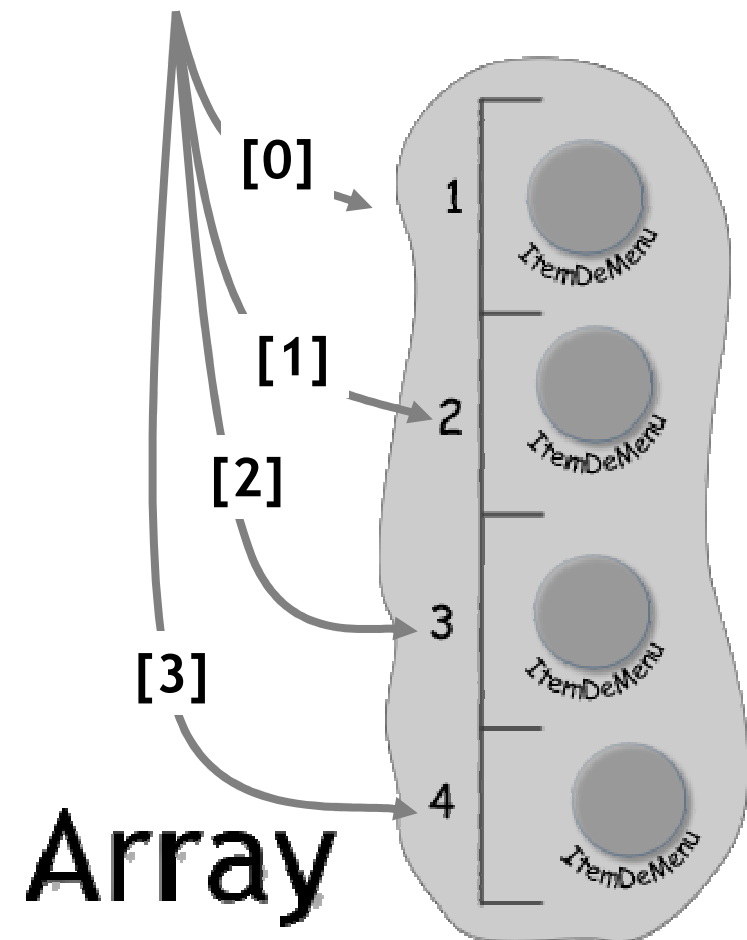
A 2.ª iteração

```
for (int i = 0; i < itensDoRestaurante.length; i++) {  
    ItemDeMenu item = itensDoRestaurante[i];  
}
```

| <i>Menu Restaurante</i> | |
|-----------------------------|------|
| Sanduche Vegetariano | 2,99 |
| Alface e Pão Integral | |
| Sopa do dia | 2,80 |
| Tigela de sopa com torradas | |
| Cachorro Quente | 1,50 |
| Salsicha com molho e queijo | |
| Legumes Cozidos | 2,50 |
| Mistura de legumes | |

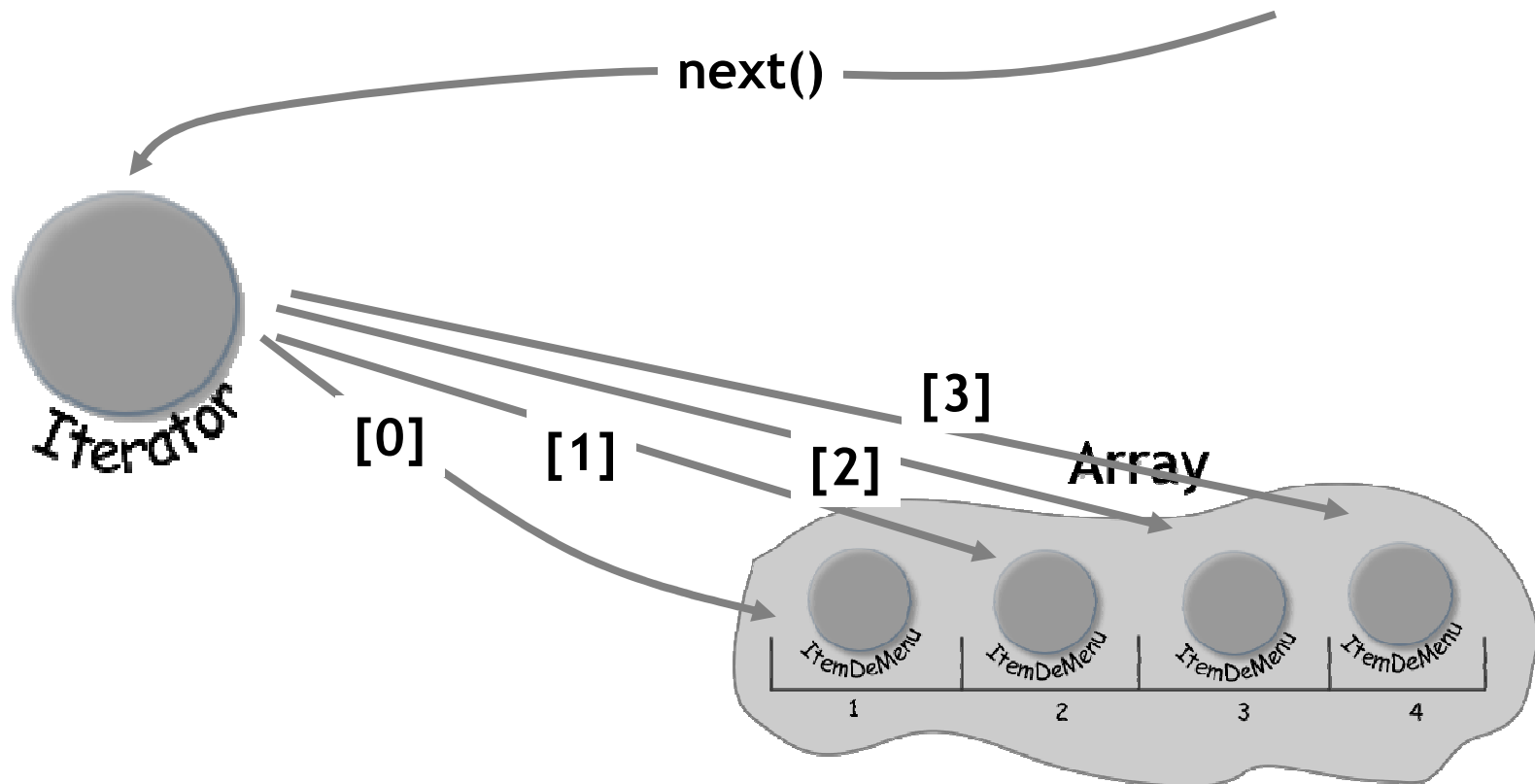


ItemDeMenu[]



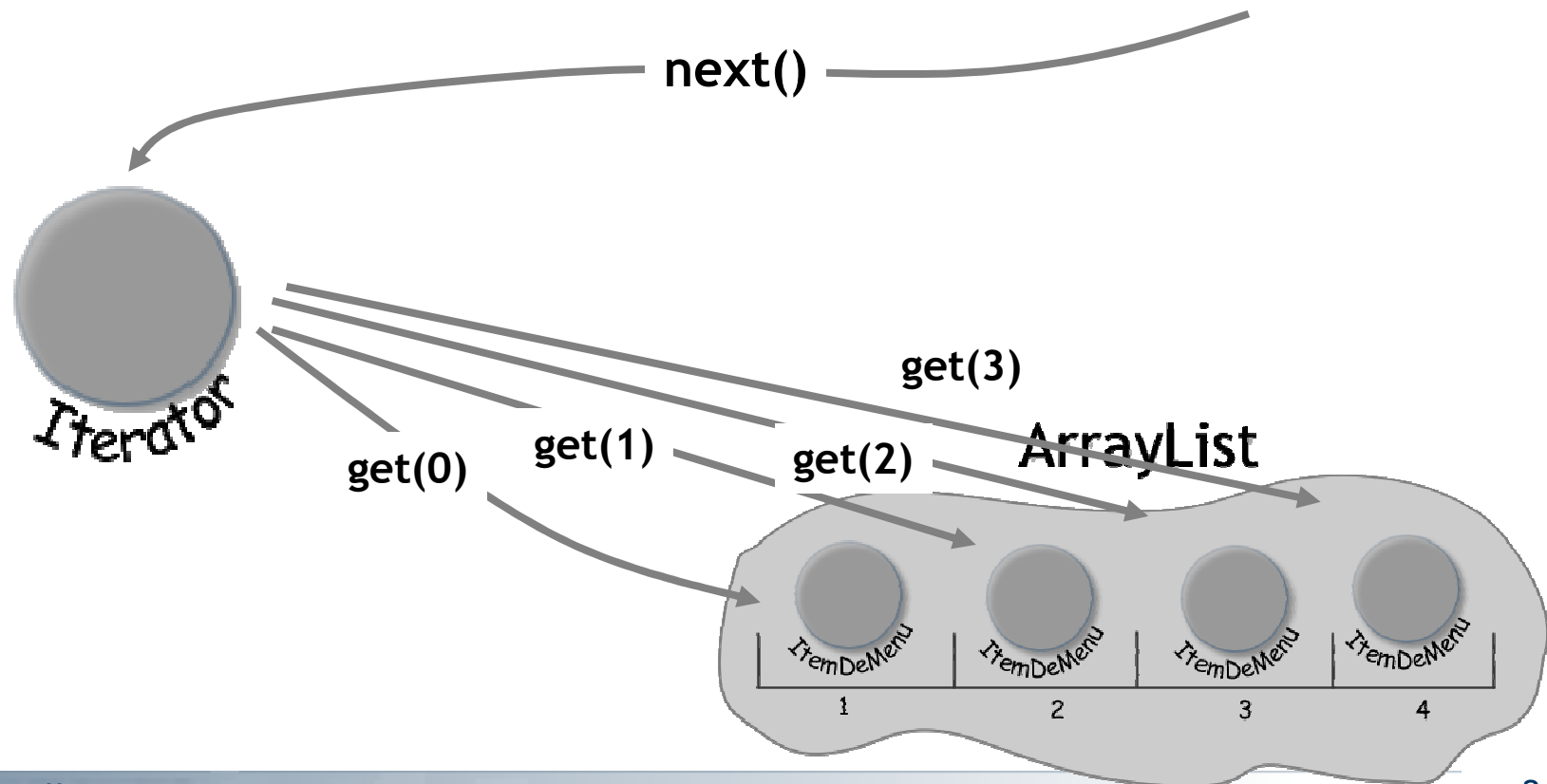
Encapsulando a iteração

```
Iterator iterator = menuRestaurante.criarIterator();  
  
while (iterator.hasNext()) {  
    ItemDeMenu item = (ItemDeMenu) iterator.next();  
}
```



Encapsulando a iteração

```
Iterator iterator = menuRestaurante.criarIterator();  
  
while (iterator.hasNext()) {  
    ItemDeMenu item = (ItemDeMenu) iterator.next();  
}
```



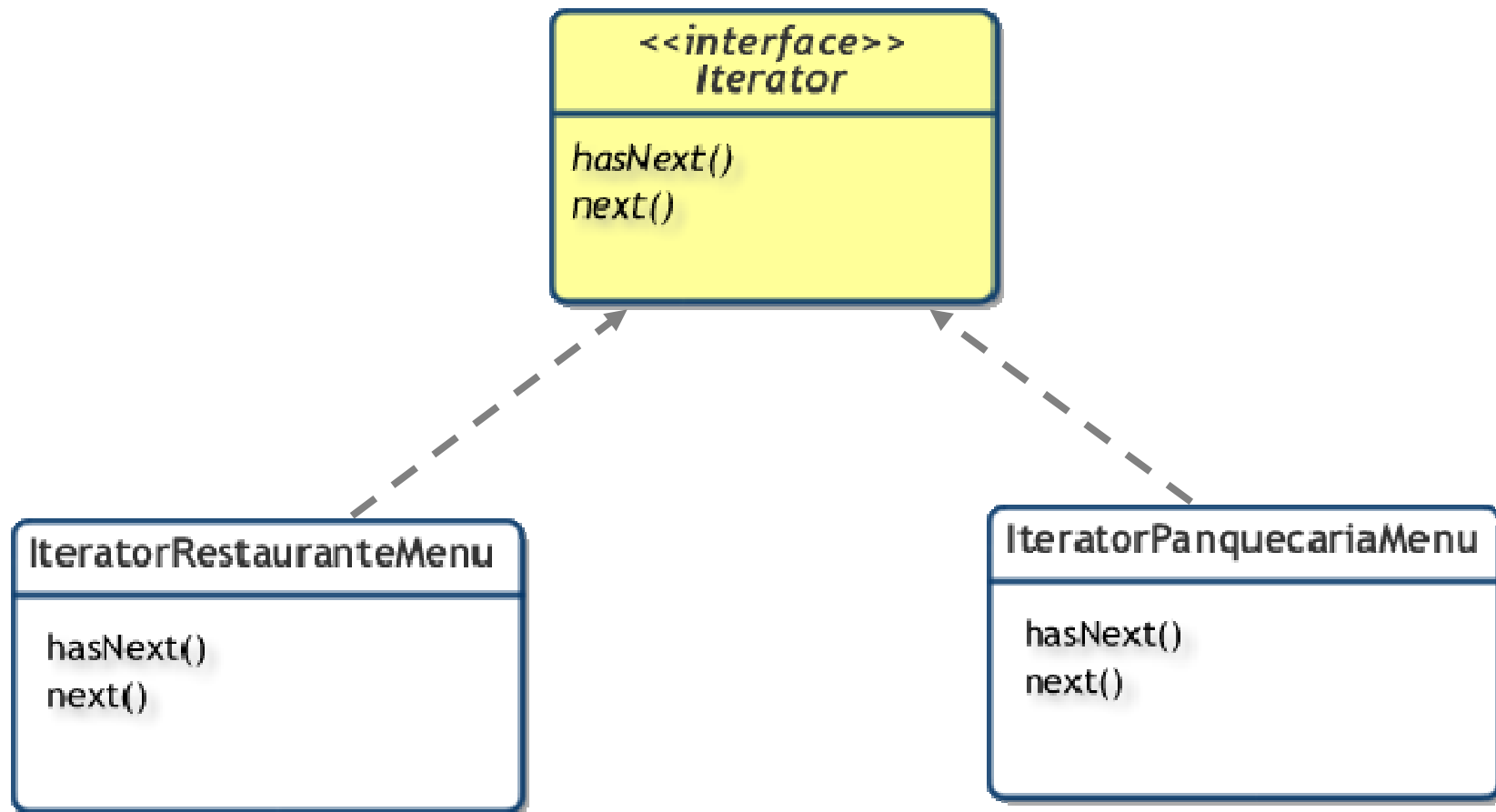
O Padrão Iterator

- Encapsular o que varia:
 - Encapsule a iteração.

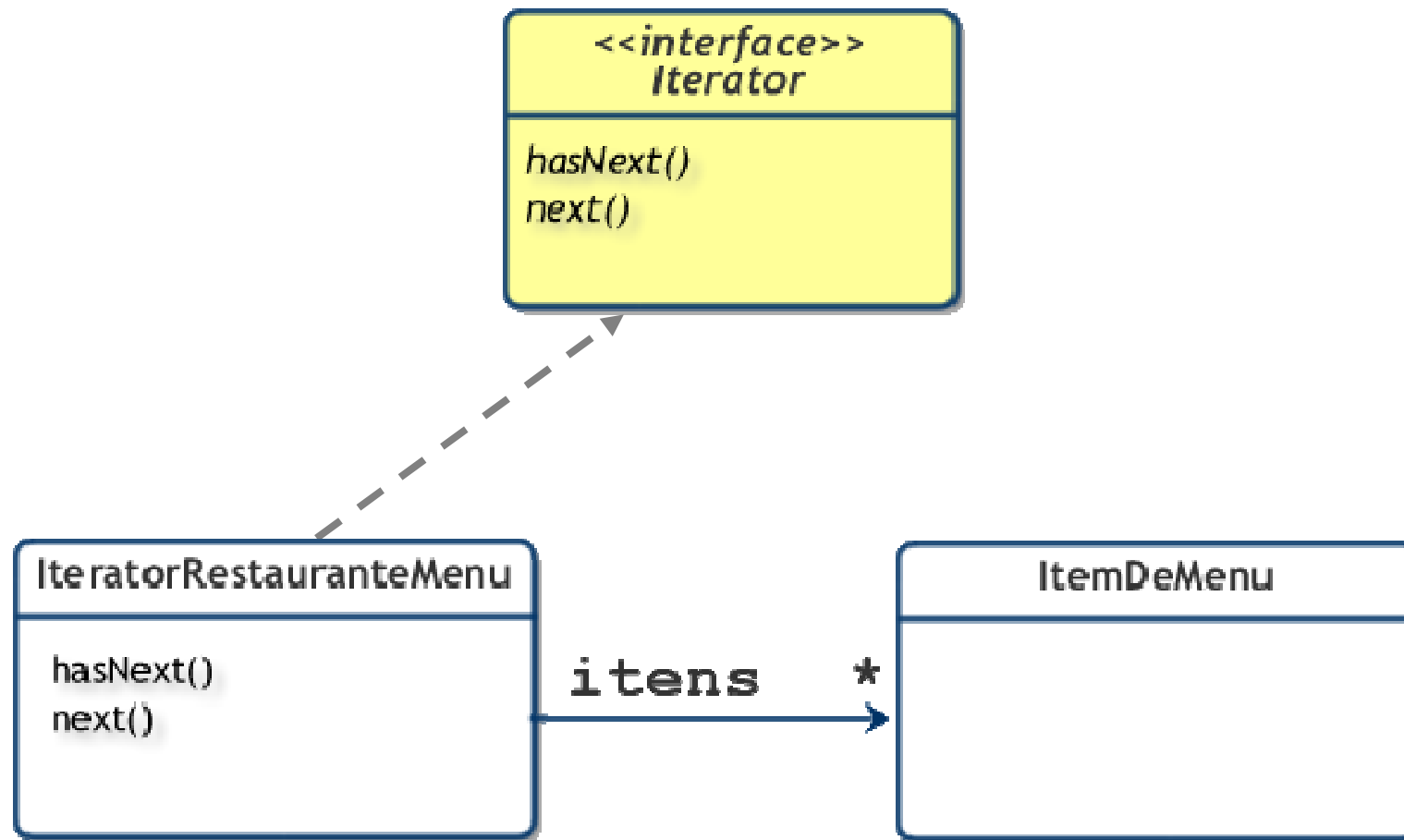
*<<interface>>
Iterator*

*hasNext()
next()*

Temos a interface Precisamos dos Objetos



IteratorRestauranteMenu.java



IteratorRestauranteMenu.java

```
public class IteratorRestauranteMenu implements Iterator {
    ItemDeMenu[] itens;
    int posicao = 0;
    public IteratorRestauranteMenu(ItemDeMenu[] itens) {
        this.itens = itens;
    }
    public boolean hasNext() {
        if (posicao >= itens.length || itens[posicao] == null) {
            return false;
        } else {
            return true;
        }
    }
    public Object next() {
        ItemDeMenu item = itens[posicao];
        posicao = posicao + 1;
        return item;
    }
}
```

No MenuDoRestaurante

■ Adicione o método criarIterator()

//Este método não é mais necessário

/*

```
public ItemDeMenu[] getItensDeMenu() {
```

```
    return itensDeMenu;
```

```
}
```

*/

```
public Iterator criarIterator() {
```

```
    return new IteratorRestauranteMenu(itensDeMenu);
```

```
}
```

Menu Restaurante

Sanduíche Vegetariano 2,99
Alface e Pão Integral

Sopa do dia 2,80
Tigela de sopa com torradas

Cachorro Quente 1,50
Salsicha com molho e queijo

Legumes Cozidos 2,50
Mistura de legumes

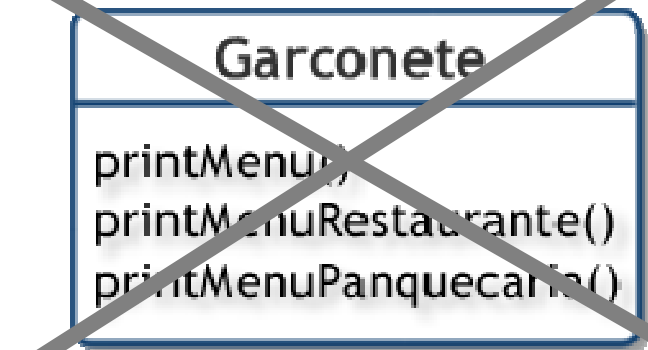
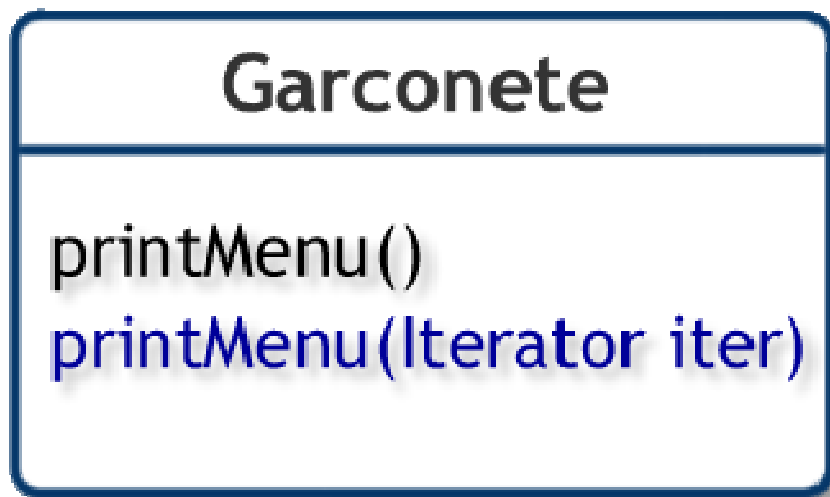
Exercício

- Fazer o mesmo processo para o MenuDaPanquecaria:
 - Criar o Iterator.
 - Adicionar o método criarIterator.

| MENU PANQUECARIA | |
|--------------------------------------|------|
| Desjejum de Panqueca | 2,99 |
| Panquecas com ovos mexidos | |
| Desjejum Tradicional | 2,99 |
| Panquecas com ovos fritos e salsicha | |
| Panquecas Doces | 3,49 |
| Panquecas com geléia de amora | |
| Waffles | 3,59 |
| Waffles com geléia de mocotó | |

Garçonete

- Ainda precisaríamos de 2 métodos para percorrer as 2 coleções?



A Garconete refatorada

```
public void printMenu(Iterator iterator) {
    while(iterator.hasNext()) {
        ItemDeMenu item = (ItemDeMenu) iterator.next();
        System.out.print(item.getNome() + " ");
        System.out.print(item.getPreco() + " ");
        System.out.println(item.getDescricao());
    }
}

public void printMenu() {
    Iterator panquecaIterator = menuDaPanquecaria.criarIterator();
    Iterator restauranteIterator = menuDoRestaurante.criarIterator();
    System.out.println("MENU\n --\nCafé da Manhã");
    printMenu(panquecaIterator);
    System.out.println("\nAlmoço");
    printMenu(restauranteIterator);
}
```

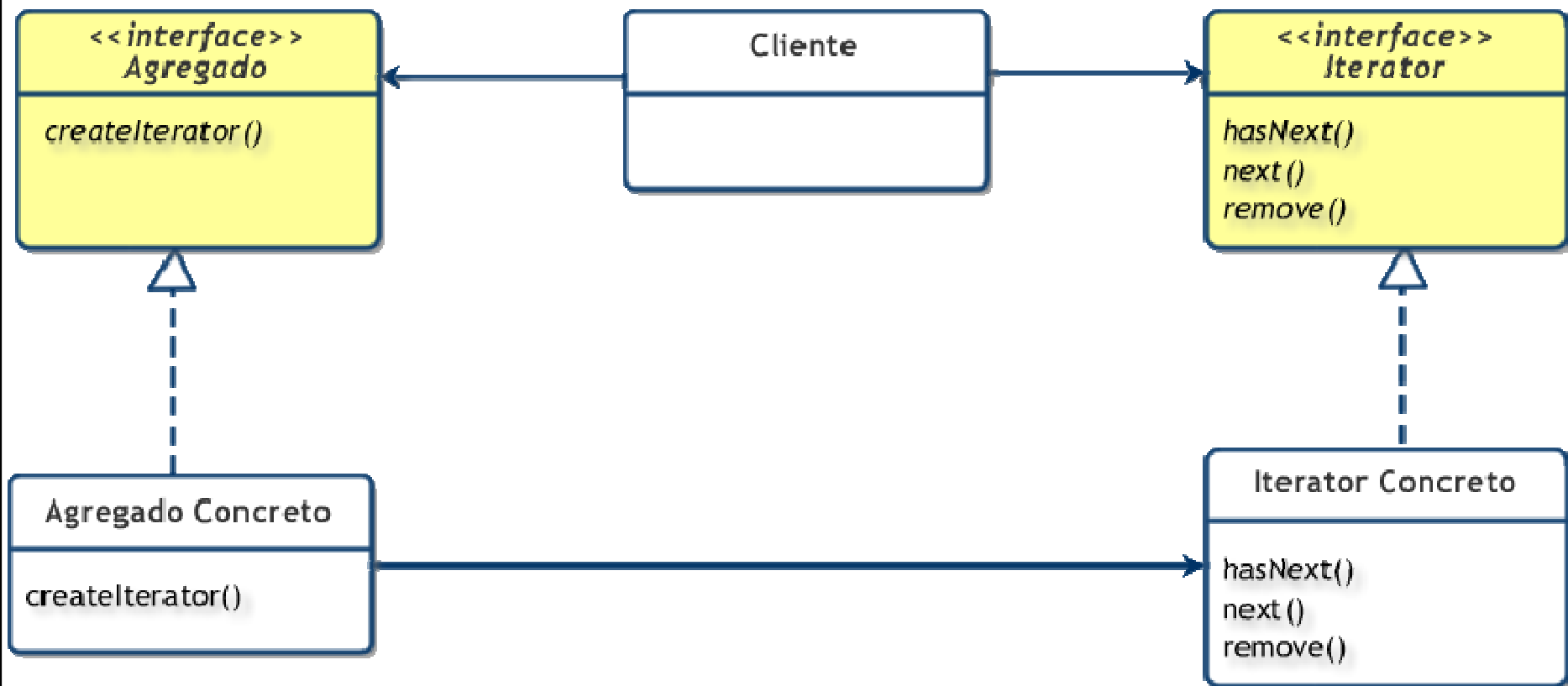
+ 1 Padrão ITERATOR

O **Padrão Iterator** fornece uma maneira de **acessar sequencialmente os elementos** de um objeto agregado **sem expor** a sua representação subjacente.

Aplicabilidade

- Acessar conteúdos de um objeto agregado sem expor a sua representação interna.
- Suportar múltiplos percursos de objetos agregados.
- Fornecer uma interface uniforme que percorra diferentes estruturas agregadas.

Diagrama de classes



Participantes

■ **Iterator**

- Fornece a interface que todos os iterators devem implementar e o conjunto de métodos para percorrer todos os elementos de uma coleção.

■ **Iterator Concreto**

- O Iterator concreto é responsável por gerenciar o posição atual da iteração.

■ **Agregado**

- Uma interface comum para objetos agregados.
- Desacopla o cliente da implementação da sua coleção de objetos.

■ **Agregado Concreto**

- Possui uma coleção de objetos.
- Implementa o método que retorna um Iterator para esta coleção.

Colaborações

- Um ConcreteIterator mantém o controle do objeto corrente no agregado e pode computar o objeto sucessor (e/ou anterior) no percurso.

Consequências

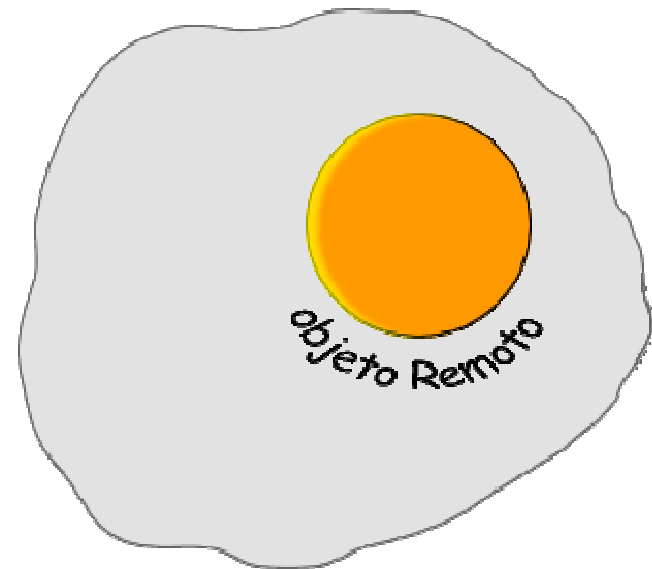
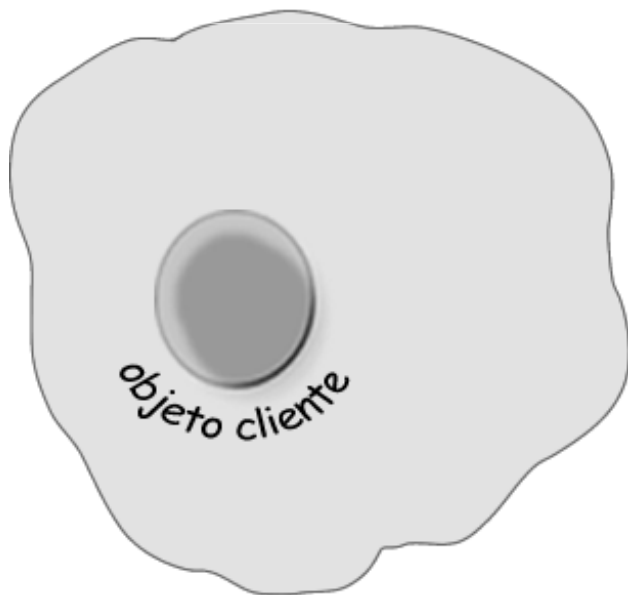
- Suporta variações no percurso do agregado.
 - Os agregados complexos podem ser percorridos de muitas maneiras.
- Iterators simplificam a interface do agregado.
 - A interface de percurso de Iterator elimina as necessidades de uma interface semelhante em Agregado.
- Mais do que um percurso pode estar em curso em um agregado.
 - Iterator mantém o controle de acompanhamento do estado do seu próprio percurso.



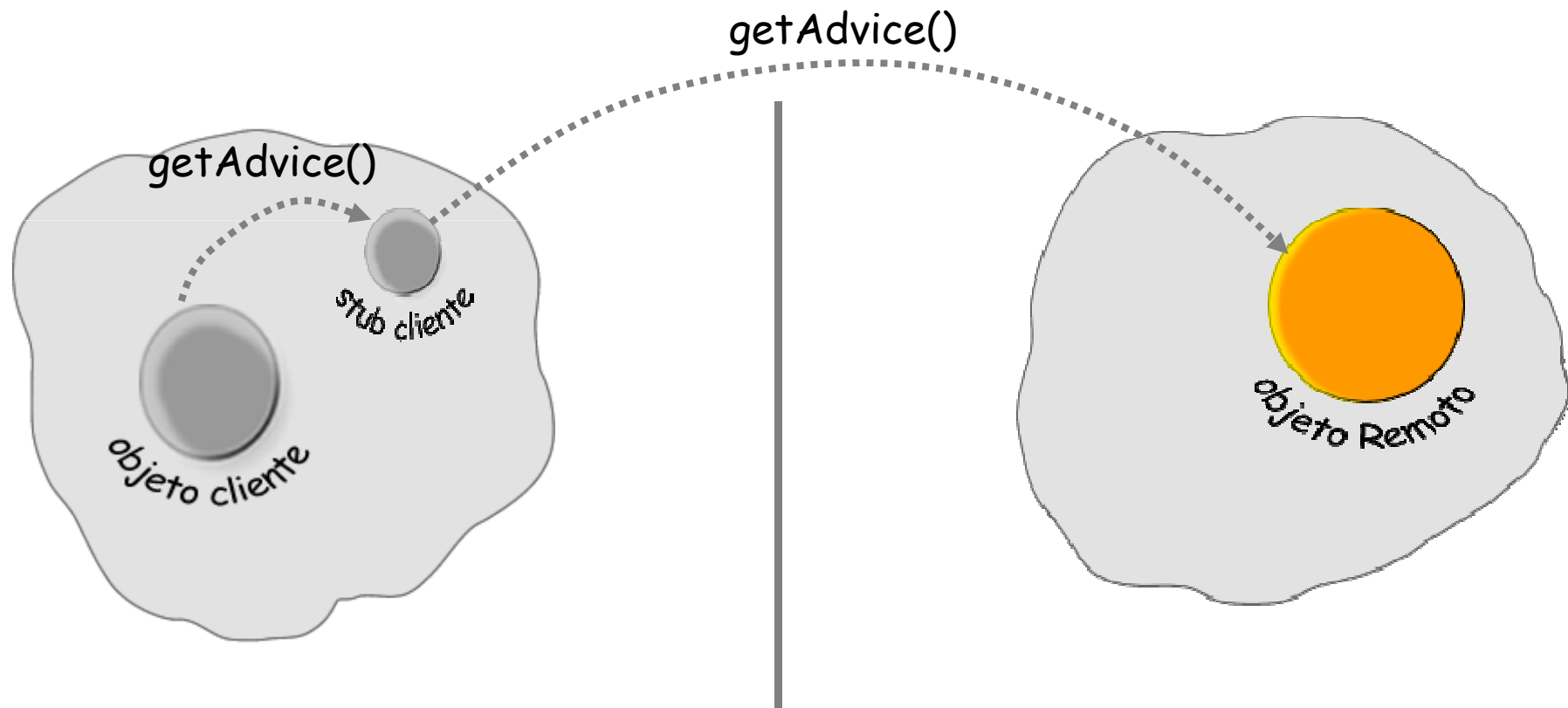
Padrão Proxy

Ator coadjuvante: O STUB

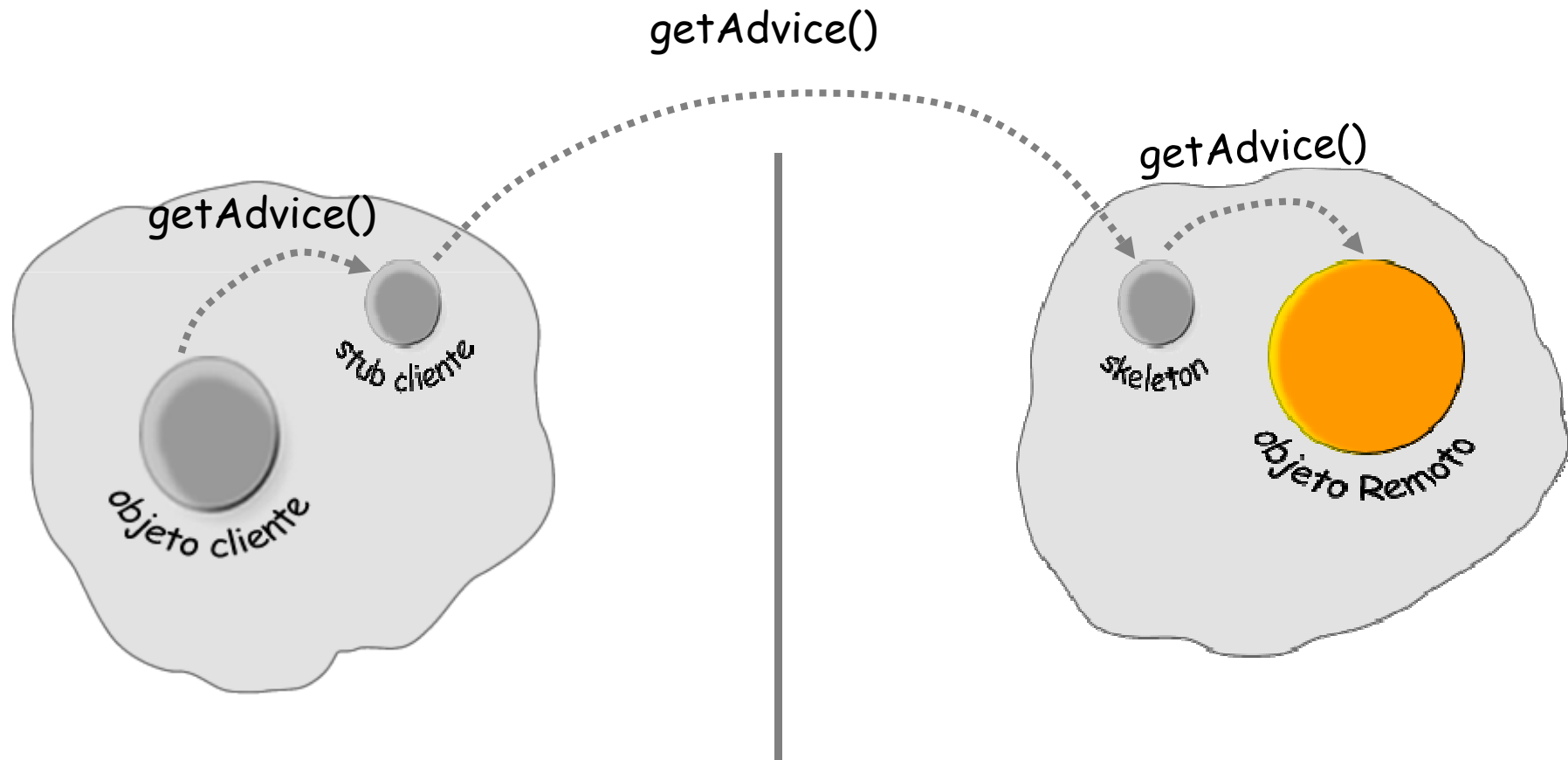
- Comunicação entre objetos remotos...
 - Precisamos de objetos auxiliares para cuidarem das tarefas de comunicação.



A chamada do método: `getAdvice()`

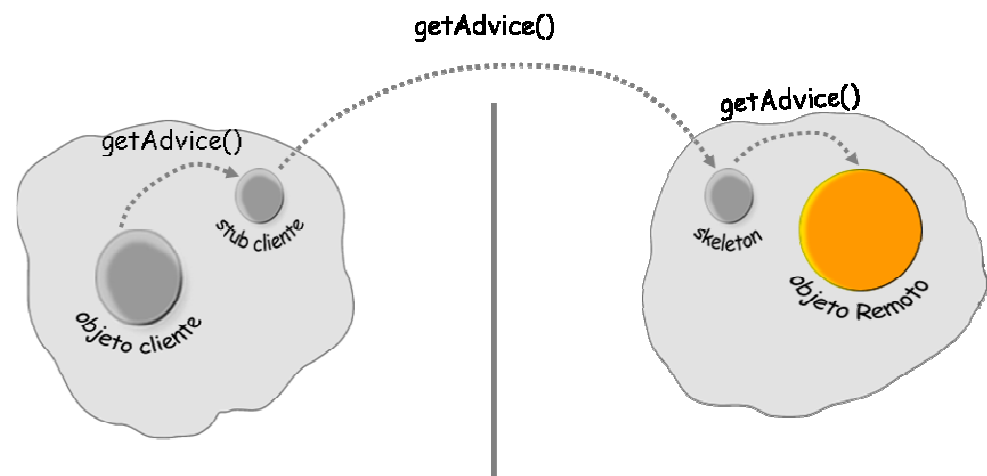


Algo está faltando!

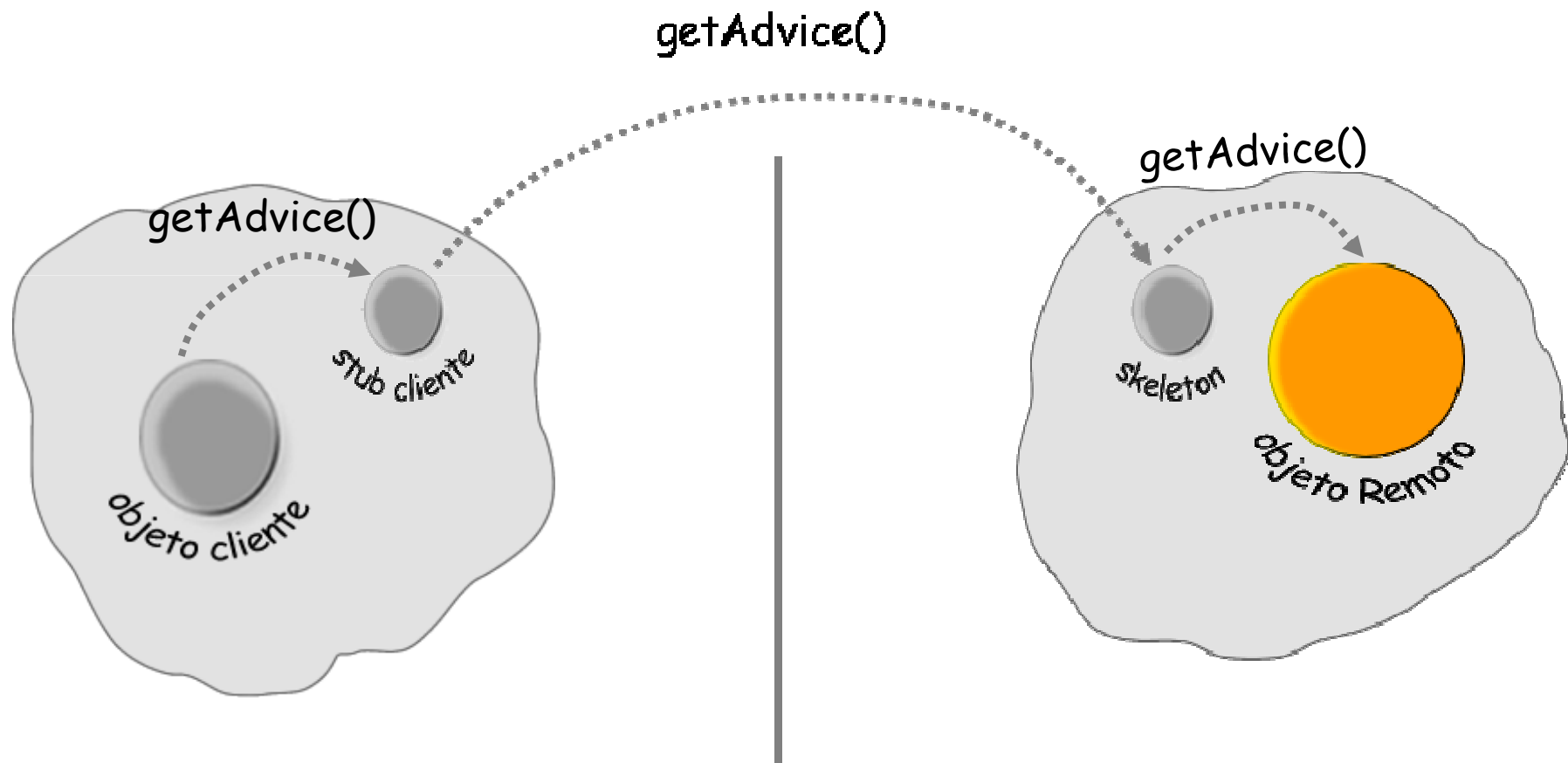


Algo está faltando!

- O cliente invoca o método do objeto remoto como se fosse a um objeto local.
- O objeto remoto recebe a invocação do método como se fosse pedido por um objeto que compartilha seu endereço de memória.



TRANSPARÊNCIA!!!



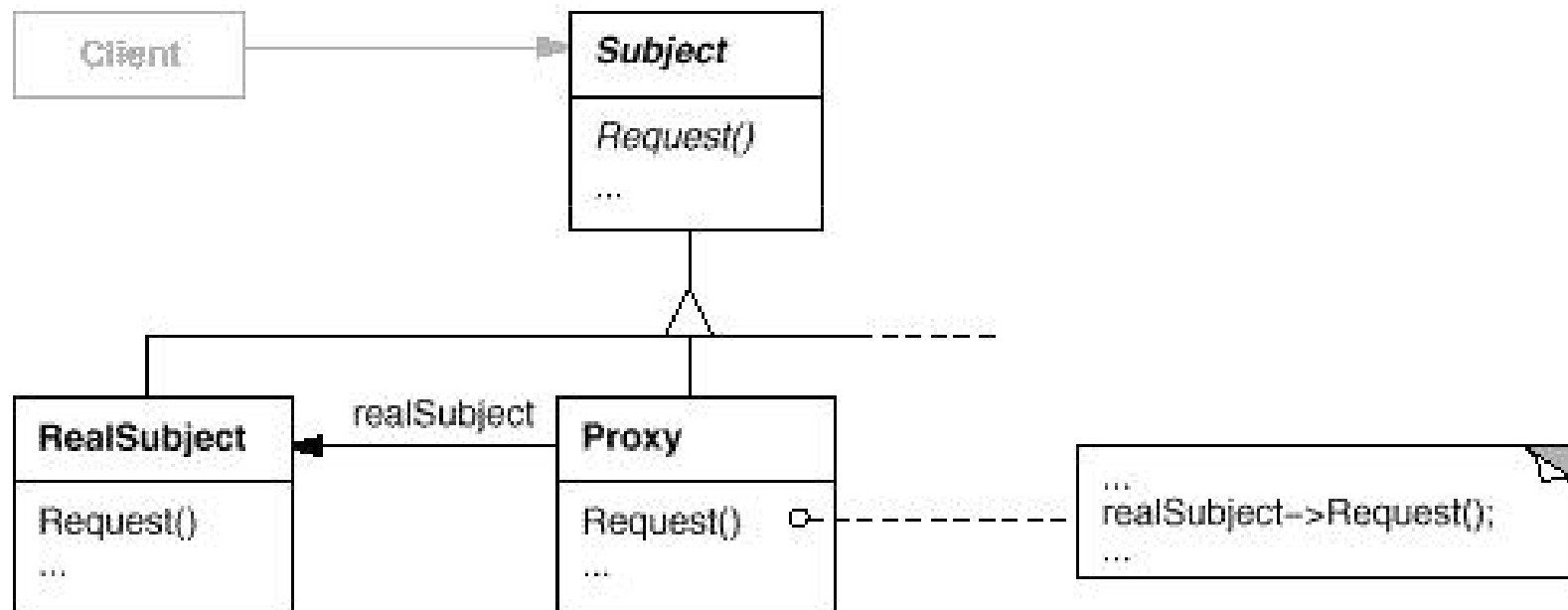
+ 1 Padrão Proxy

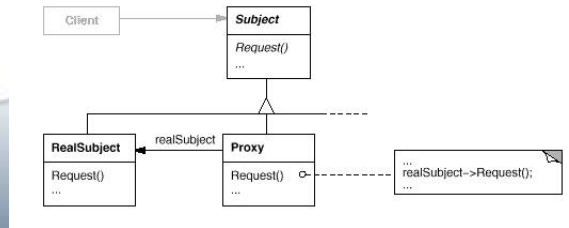
O **Padrão Proxy** fornece um substituto ou representante de outro objeto para controlar o acesso ao mesmo.

Aplicabilidade

- Use o padrão **Proxy** quando há necessidade de uma referência mais versátil, ou sofisticada, do que um simples apontador para um objeto. Por exemplo:
 - **Proxy Virtual**
 - Controla o acesso a um recurso cuja criação é onerosa.
 - **Proxy de Segurança**
 - Controla o acesso a um recurso com base em direitos de acesso.

Diagrama de Classes

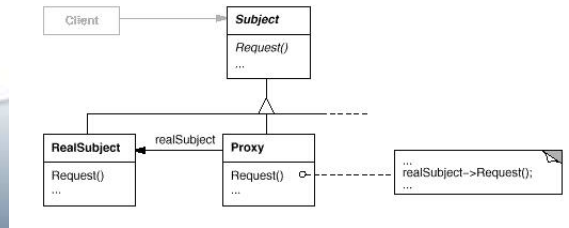




Participantes

■ Proxy

- Mantém uma referência que permite ao proxy acessar o objeto real (RealSubject).
- Fornece uma interface idêntica a de Subject, de modo que o proxy possa substituir o objeto real (RealSubject).
- Controla o acesso ao objeto real e pode ser responsável pela sua criação e exclusão.



Participantes

■ Subject

- Define uma interface comum para RealSubject e Proxy, de maneira que um proxy possa ser usado em qualquer lugar em que um RealSubject é esperado.

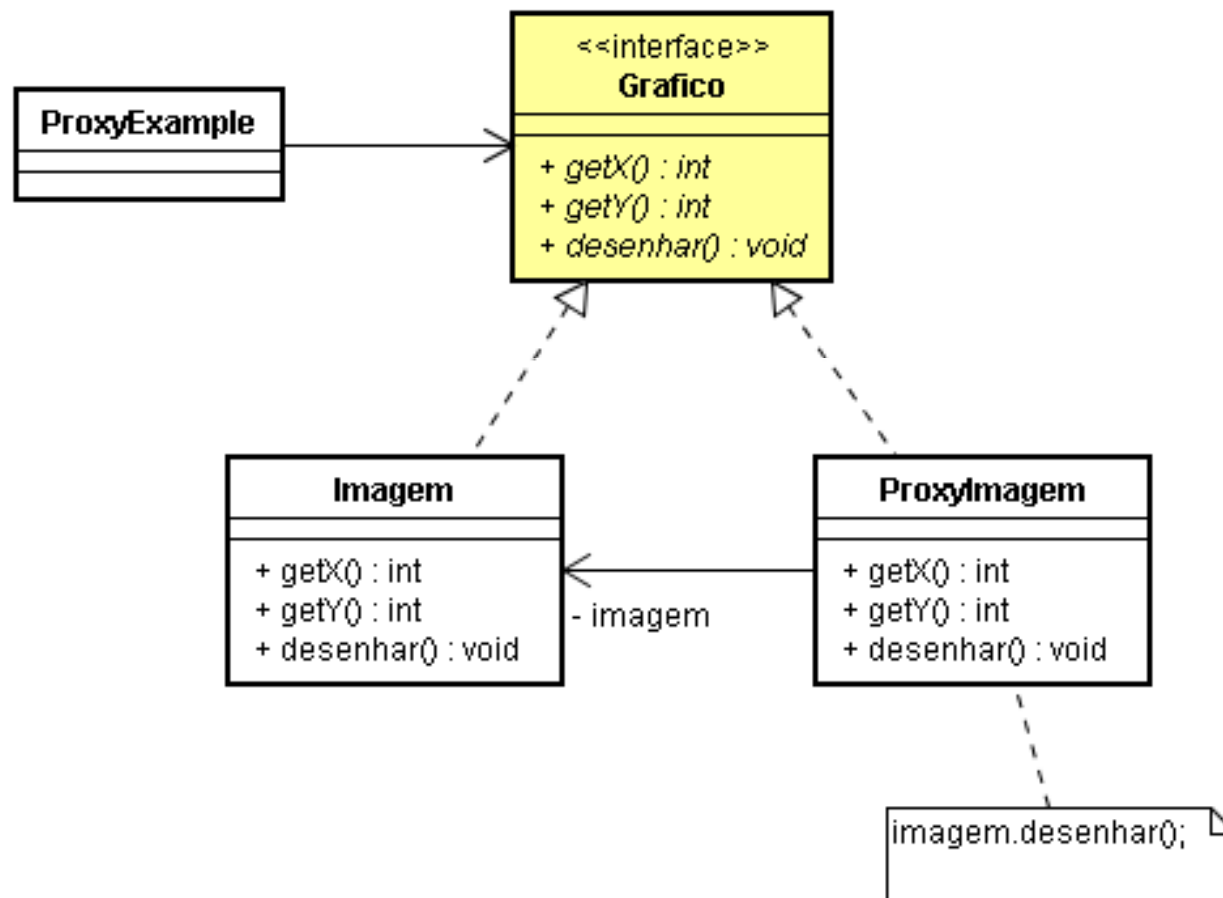
■ RealSubject

- Define o objeto real que o proxy representa.

Colaborações

- Dependendo de seu tipo, Proxy repassa solicitações para RealSubject quando apropriado.

Outro exemplo: Carregando imagens pesadas



Interface Grafico.java (Subject)

```
/**
 * Interface comum entre o objeto real e o Proxy
 */
public interface Grafico
{
    public int getX();
    public int getY();
    public void desenhar();
}
```

Imagem.java (RealSubject)

```
class Imagem implements Grafico {  
    private int x;  
    private int y;  
    public Imagem(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public int getX() { return x; }  
    public int getY() { return y; }  
    public void desenhar() {  
        System.out.println(  
            "desenhando imagem pesada...");  
    }  
}
```

ProxyImagem.java (Proxy)

```
class ProxyImagem implements Grafico {
    private int x;
    private int y;
    private Grafico imagem;
    public ProxyImagem(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() { return x; }
    public int getY() { return y; }
    public void desenhar() {
        System.out.println("desenhando imagem do proxy...");
        if (imagem == null) {
            //Apenas cria o objeto real quando chamar este método
            imagem = ImagemDAO.getImagemByXeY(this.x, this.y);
        }
        //Delega para o objeto real
        imagem.desenhar();
    }
}
```

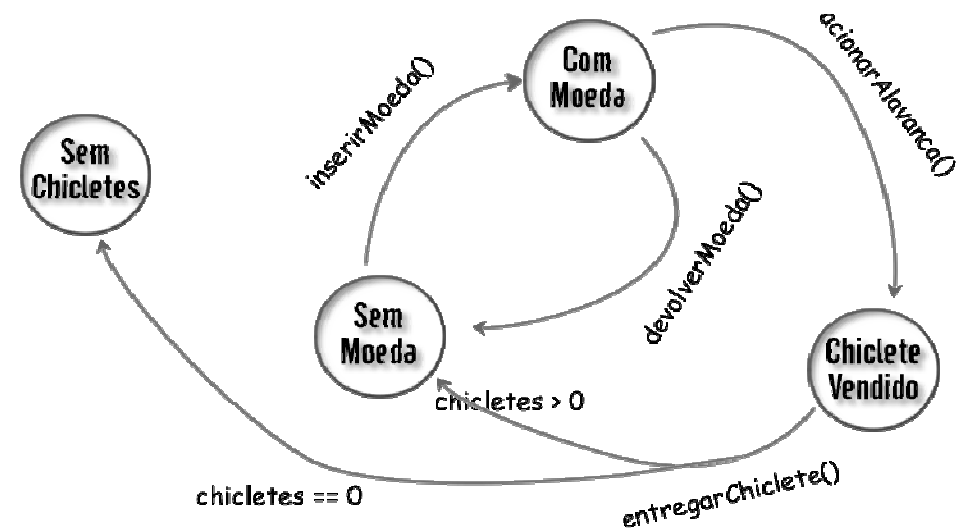
Consequências

- O uso do proxy (remoto) pode ocultar o fato de que um objeto reside num espaço de endereçamento diferente.
- Um proxy virtual pode executar otimizações, tais como criação de um objeto sob demanda.
- Alguns proxies permitem tarefas adicionais de organização quando o objeto é acessado.
- O uso do proxy pode reduzir significativamente o custo computacional da cópia de objetos pesados.



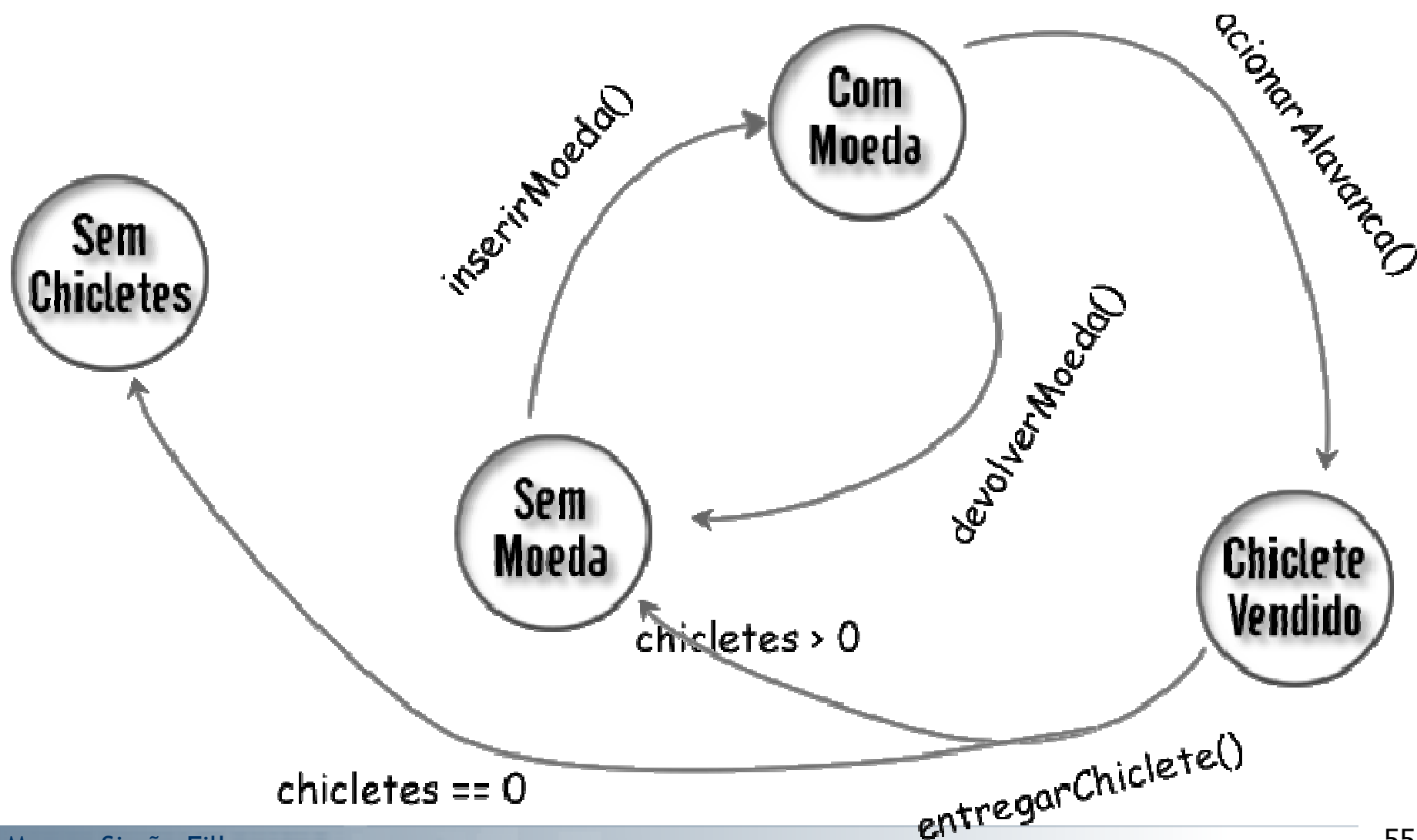
Padrão State

Padrão State





Máquina e seus estados



Transformando os estados em algo programável

- Determine quais são os estados.

**Sem
Moeda**

**Com
Moeda**

**Chiclete
Vendido**

**Sem
Chicletes**

Transformando os estados em algo programável

- Em uma classe

- Crie uma variável de instância para referenciar cada um destes estados.
- Atribua valores para cada uma:

```
final static int SEM_CHICLETE = 0;
```

```
final static int SEM_MOEDA = 1;
```

```
final static int COM_MOEDA = 2;
```

```
final static int CHICLETE_VENDIDO = 3;
```



Transformando os estados em algo programável

- Determine quais ações podem ocorrer no sistema

inserirMoeda()

devolverMoeda()

acionarAlavanca()

entregarChiclete()

Estes métodos são a interface da máquina

**Sem
Moeda**

**Com
Moeda**

**Chiclete
Vendido**

**Sem
Chicletes**

Transformando os estados em algo programável

- A classe se comportará como uma máquina de estados.
- Cada método utiliza instruções condicionais para determinar qual comportamento deve ser realizado em cada estado.

Sem
Moeda

Com
Moeda

Chiclete
Vendido

Sem
Chicletes

Transformando os estados em algo programável

- Exemplo para a ação de inserir moeda

```
public void inserirMoeda() {  
    if (estado == COM_MOEDA) {  
        System.out.println("Você não pode inserir outra moeda");  
    } else if (estado == SEM_MOEDA) {  
        estado = COM_MOEDA;  
        System.out.println("Você inseriu uma moeda");  
    } else if (estado == SEM_CHICLETE) {  
        System.out.println(  
            "Não insira moedas. A máquina está sem chicletes.");  
    } else if (estado == CHICLETE_VENDIDO) {  
        System.out.println("Aguarde a entrega do seu chiclete.");  
    }  
}
```

Sem
Moeda

Com
Moeda

Chiclete
Vendido

Sem
Chicletes

A classe

inserirMoeda()

devolverMoeda()

acionarAlavanca()

entregarChiclete()

| MaquinaDeChiclete |
|--|
| <u>~ SEM_CHICLETE : int = 0</u> |
| <u>~ SEM_MOEDA : int = 1</u> |
| <u>~ COM_MOEDA : int = 2</u> |
| <u>~ CHICLETE_VENDIDO : int = 3</u> |
| ~ estado : int = SEM_CHICLETE |
| ~ quantidadeDeChicletes : int = 0 |
| + MaquinaDeChiclete(quantidade : int) |
| + inserirMoeda() : void |
| + devolverMoeda() : void |
| + acionarAlavanca() : void |
| + entregarChiclete() : void |
| + reabastecer(numChicletes : int) : void |
| + toString() : String |

Sem
Moeda

Com
Moeda

Chiclete
Vendido

Sem
Chicletes

Problemas

- Quando for necessário modificações.
- Adição de novo estado.

| MaquinaDeChiclete |
|--|
| <u>~ SEM_CHICLETE : int = 0</u> <u>~ SEM_MOEDA : int = 1</u> <u>~ COM_MOEDA : int = 2</u> <u>~ CHICLETE_VENDIDO : int = 3</u> ~ estado : int = SEM_CHICLETE ~ quantidadeDeChicletes : int = 0 |
| + MaquinaDeChiclete(quantidade : int) + inserirMoeda() : void + devolverMoeda() : void + acionarAlavanca() : void + entregarChiclete() : void + reabastecer(numChicletes : int) : void + toString() : String |

Solução

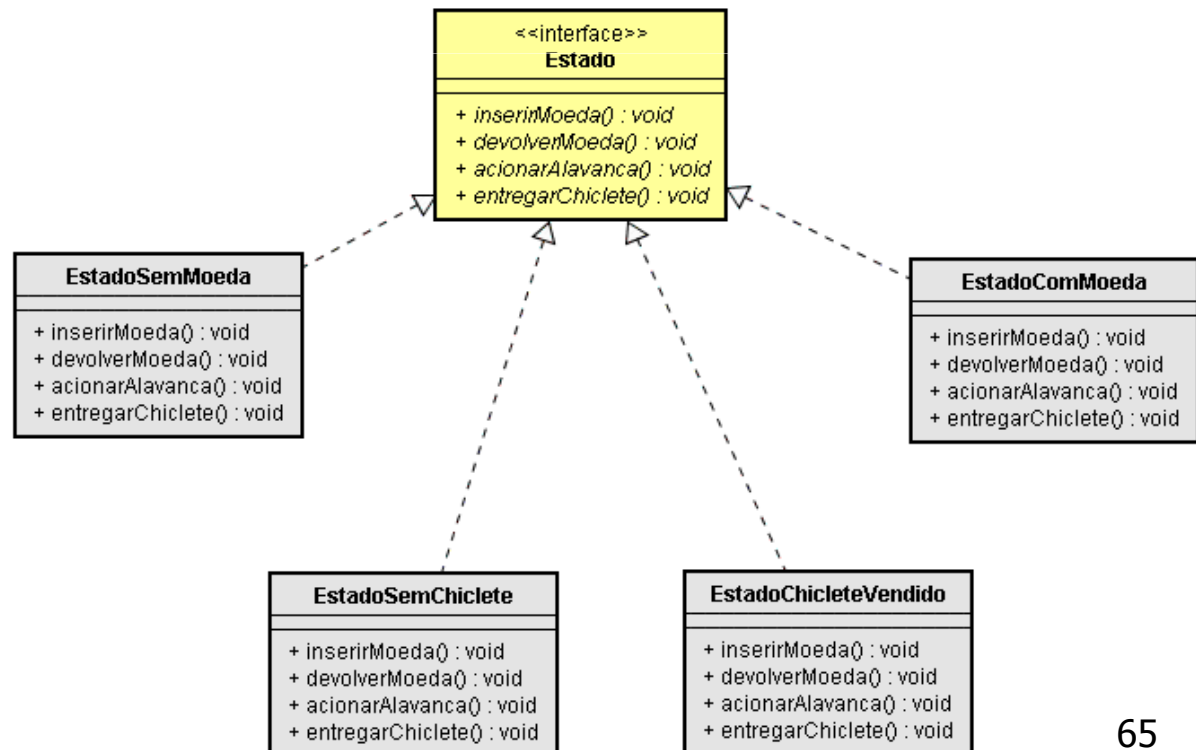
- Encapsular o que varia.
- Colocar o comportamento de cada estado dentro da sua própria classe.
- A Máquina vai delegar as tarefas para seus estados.

Solução

- Definição de uma interface para o Estado.
 - Deve conter um método para cada ação da máquina.
- Implementar a classe Estado para cada estado da máquina.
- Remover o código condicional.
- Delegar.

O que deve ser implementado

```
final static int SEM_CHICLETE = 0;  
final static int SEM_MOEDA = 1;  
final static int COM_MOEDA = 2;  
final static int CHICLETE_VENDIDO = 3;
```



EstadoSemMoeda.java

```
public class EstadoSemMoeda implements Estado
{
    MaquinaDeChiclete maquinaDeChiclete;

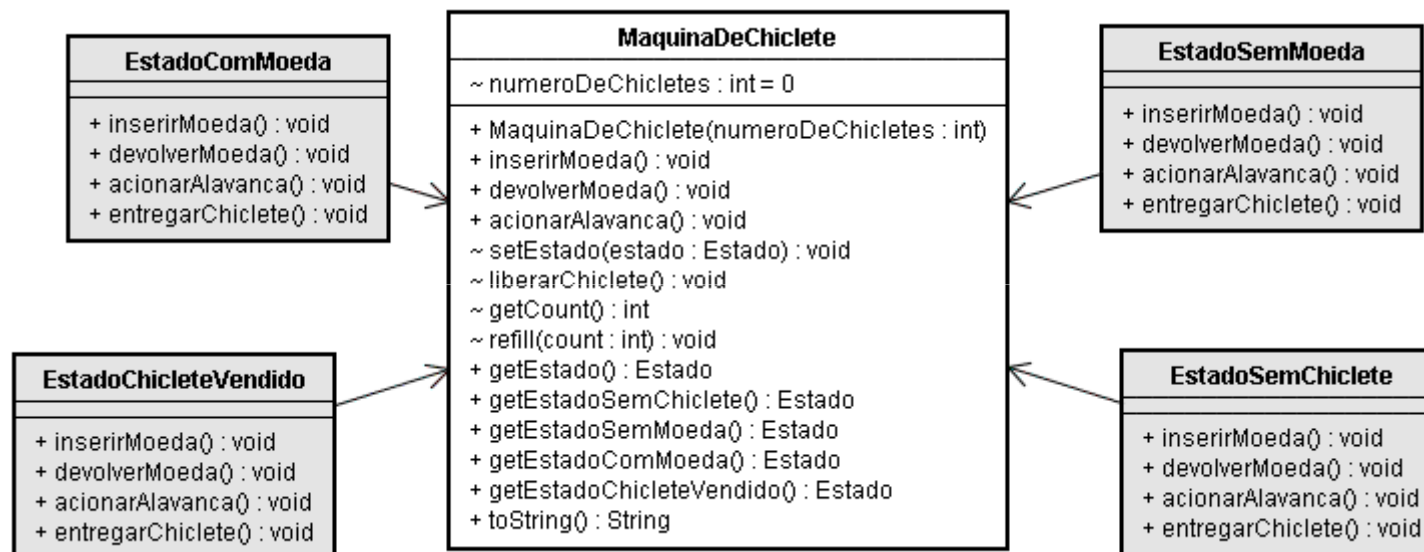
    public EstadoSemMoeda(MaquinaDeChiclete maquinaDeChiclete)
    {
        this.maquinaDeChiclete = maquinaDeChiclete;
    }

    public void inserirMoeda()
    {
        System.out.println("Você inseriu uma moeda");
        maquinaDeChiclete.setEstado(
            maquinaDeChiclete.getEstadoComMoeda());
    }
}
```

EstadoSemMoeda.java

```
public void devolverMoeda() {  
    System.out.println("Você não inseriu moedas");  
}  
  
public void acionarAlavanca() {  
    System.out.println("Você acionou a alavanca mas esqueceu da  
moeda");  
}  
  
public void entregarChiclete() {  
    System.out.println("Você precisa pagar primeiro");  
}  
  
public String toString() {  
    return "esperando que alguém insira uma moeda";  
}  
}
```

A nova máquina



O que está sendo feito?

- Separação do comportamento de cada estado em sua própria classe.
- Remoção dos condicionais.
- Proteção de cada estado contra modificações.
- Máquina de Chiclete aberta para receber novos estados.

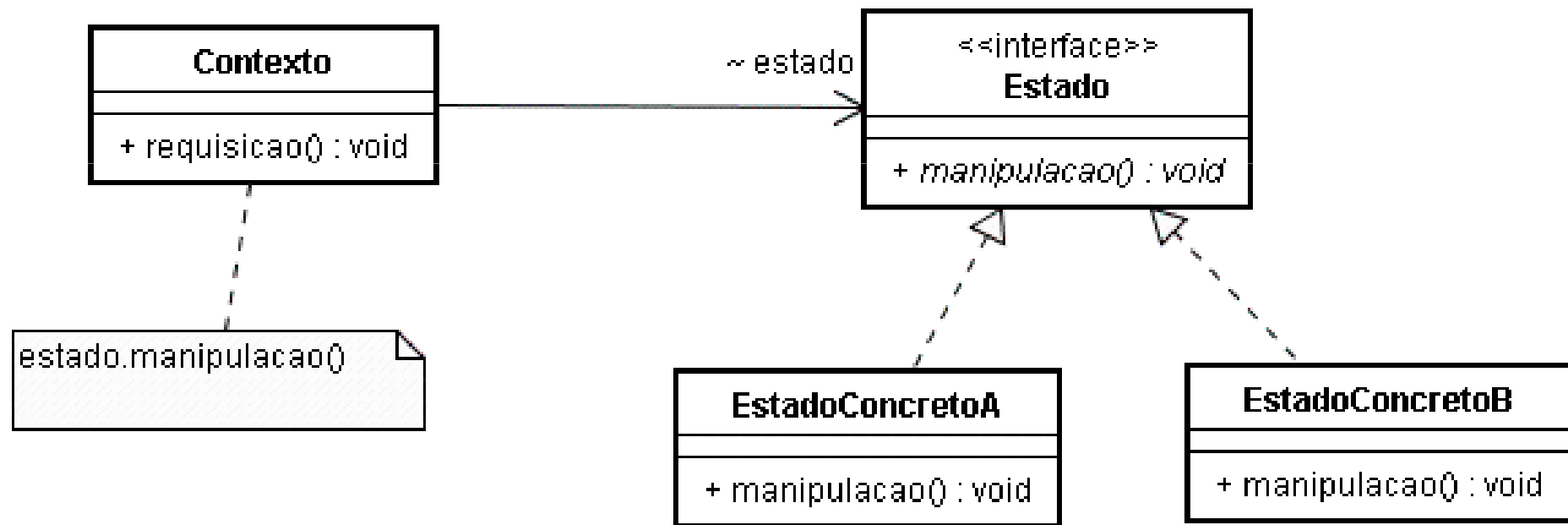
+1 Padrão STATE

O **Padrão State** permite que um objeto altere seu comportamento quando o seu estado interno muda. O objeto parecerá ter mudado de classe.

Aplicabilidade

- O comportamento de um objeto depende de seu estado e ele pode mudar seu comportamento em tempo de execução, dependendo desse estado.
- Operações têm comandos condicionais grandes, de várias alternativas, que dependem do estado.
 - Esse estado é comumente representado por uma ou mais constantes enumeradas.
 - Frequentemente, várias operações conterão essa mesma estrutura condicional.
 - O padrão State coloca cada ramo do comando condicional em uma classe separada.

Diagrama de classes



Participantes

■ Contexto

- Define uma interface de interesse para os clientes.
- Mantém uma referência para uma subclasse da interface Estado que define o estado atual.

■ Estado

- Define uma interface para encapsulamento associado com um determinado estado do Contexto.

■ EstadoConcreto

- Cada subclasse implementa um comportamento associado com um estado do Contexto.

Colaborações

- O Contexto delega solicitações específicas de estados para o objeto corrente EstadoConcreto.
- Um Contexto pode passar a si próprio como um argumento para o objeto Estado que trata a solicitação.
 - Isso permite ao objeto Estado acessar o Contexto, se necessário.

Colaborações

- Contexto é a interface primária para os clientes.
 - Os clientes podem configurar um Contexto com objetos Estado. Uma vez que o Contexto está configurado, seus clientes não têm que lidar com os objetos Estado diretamente.
- Tanto Contexto quanto as subclasses EstadoConcreto podem decidir qual estado sucede outro, e sob quais circunstâncias.

MaquinaDeChiclete.java

```
public class MaquinaDeChiclete {
    Estado estadoComMoeda;
    Estado estadoSemMoeda;
    Estado estadoSemChiclete;
    Estado estadoChicleteVendido;
    int numChicletes = 0;
    Estado estado = estadoSemChiclete;

    public MaquinaDeChiclete(int numChicletes) {
        estadoSemMoeda = new EstadoSemMoeda(this);
        estadoComMoeda = new EstadoComMoeda(this);
        estadoSemChiclete = new EstadoSemChiclete(this);
        estadoChicleteVendido = new EstadoChicleteVendido(this);
        this.numChicletes = numChicletes;
        if (numChicletes > 0)
            estado = estadoSemMoeda;
    }
}
```

MaquinaDeChiclete.java

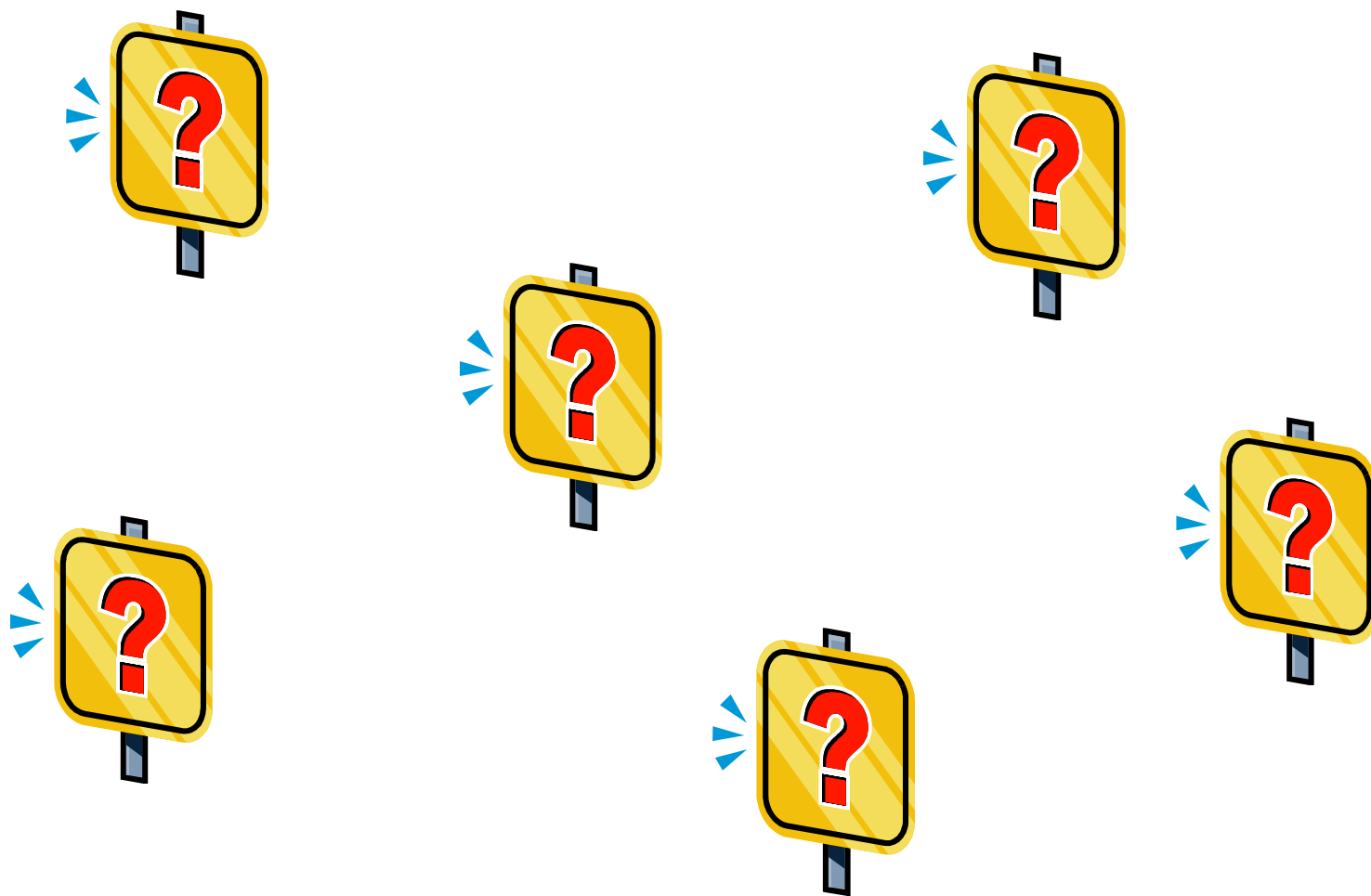
```
public void inserirMoeda() {
    estado.inserirMoeda();
}
public void devolverMoeda() {
    estado.devolverMoeda();
}
public void acionarAlavanca() {
    estado.acionarAlavanca();
    estado.entregarChiclete();
}
public void entregarChiclete() {
    if (numChicletes != 0)
        numChicletes = numChicletes - 1;
}
void setEstado (Estado estado) {
    this.estado = estado;
}
void getEstadoComMoeda() { return estadoComMoeda; } ...
}
```

Consequências

- Confina comportamento específico de estados.
 - State coloca todo comportamento associado com um estado particular em um objeto.
- Particiona o comportamento para estados diferentes.
 - O comportamento para diversos estados é distribuído entre várias subclasses de State.

Consequências

- Torna explícitas as transições de estado.
 - Quando um objeto define seu estado corrente unicamente em termos de valores de dados internos, suas transições de estado não têm representação explícita.
- Objetos State podem ser compartilhados.
 - Se os objetos State não possuem variáveis de instância, pois o estado que eles representam está codificado inteiramente em seu tipo, então contextos podem compartilhar um objeto State.





Obrigado!!!

Agradecimentos:

Prof. Eduardo Mendes

Prof. Régis Simão

Faculdade 7 de Setembro