



Padrões de Projeto (Design Patterns)


Padrões de Projeto

Padrão: é uma descrição de um problema e sua solução que pode ser aplicado em muitas situações, que sugere como aplicá-lo e discute as suas vantagens e desvantagens.

Padrões não expressam idéias de projeto novas.




Padrões tentam codificar conhecimento e princípios já conhecidos e que funcionam.



Padrões de Projeto

➔ Através de um padrão conhecido, os projetistas podem discutir princípios complexos ou ideias de projeto a partir do nome do padrão.



Padrões de Projeto

Pure Fabrication (Fabricação Pura) - Larman

Adapter (Adaptador) - GoF

Factory (Fábrica)

Abstract Factory (Fábrica Abstrata) - GoF


Strategy (Estratégia) - GoF

Singleton - GoF

Composite (Composto) - GoF

Facade (Fachada) - GoF

Observer (Observador) - GoF



1. Fabricação Pura (Pure Fabrication)

Problema: Qual objeto deve ter uma responsabilidade, quando você quer manter a alta coesão e o baixo acoplamento, mas as soluções oferecidas pelo padrão Especialista da Informação não são apropriadas?

Solução: Atribua um conjunto de responsabilidades altamente coesivas a uma classe artificial que não representa um conceito do domínio.

Fabricação Pura

Exemplo: Qual classe deverá ser a responsável pelo armazenamento de uma venda no banco de dados?

?
insere (objeto) atualiza (objeto) ...

- Pelo padrão Especialista da Informação, a classe Venda seria a candidata a armazenar uma compra no banco de dados.

➞ Baixa coesão, alto acoplamento e baixo reuso

Fabricação Pura

Exemplo: Qual classe deverá ser a responsável pelo armazenamento de uma venda no banco de dados?

ArmazenamentoPersistente
insere (objeto) atualiza (objeto) ...

Solução: nova classe responsável somente pelo armazenamento dos objetos persistentes.

Fabricação Pura

Exemplo:

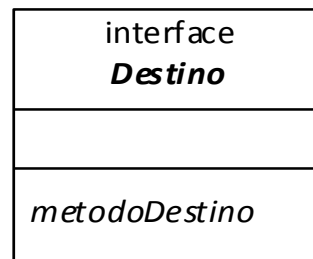
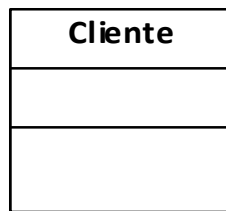
Com o padrão Fabricação Pura:

- A classe Venda permanece com alta coesão e baixo acoplamento.
- A classe ArmazenamentoPersistente é relativamente coesa: sua finalidade é armazenar e inserir objetos em um meio de armazenamento persistente.

2. Adaptador (Adapter)

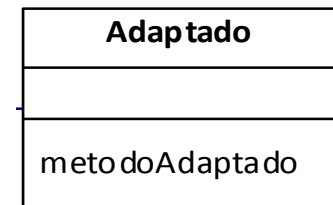
Aplicação: Você quer usar uma classe e a sua interface não combina com a que você precisa.

Problema: Como resolver interfaces incompatíveis?



← interface procurada

classe que possui o método cuja assinatura
não combina com a do método procurado



2. Adaptador (Adapter)

Aplicação: Você quer usar uma classe e a sua interface não combina com a que você precisa.

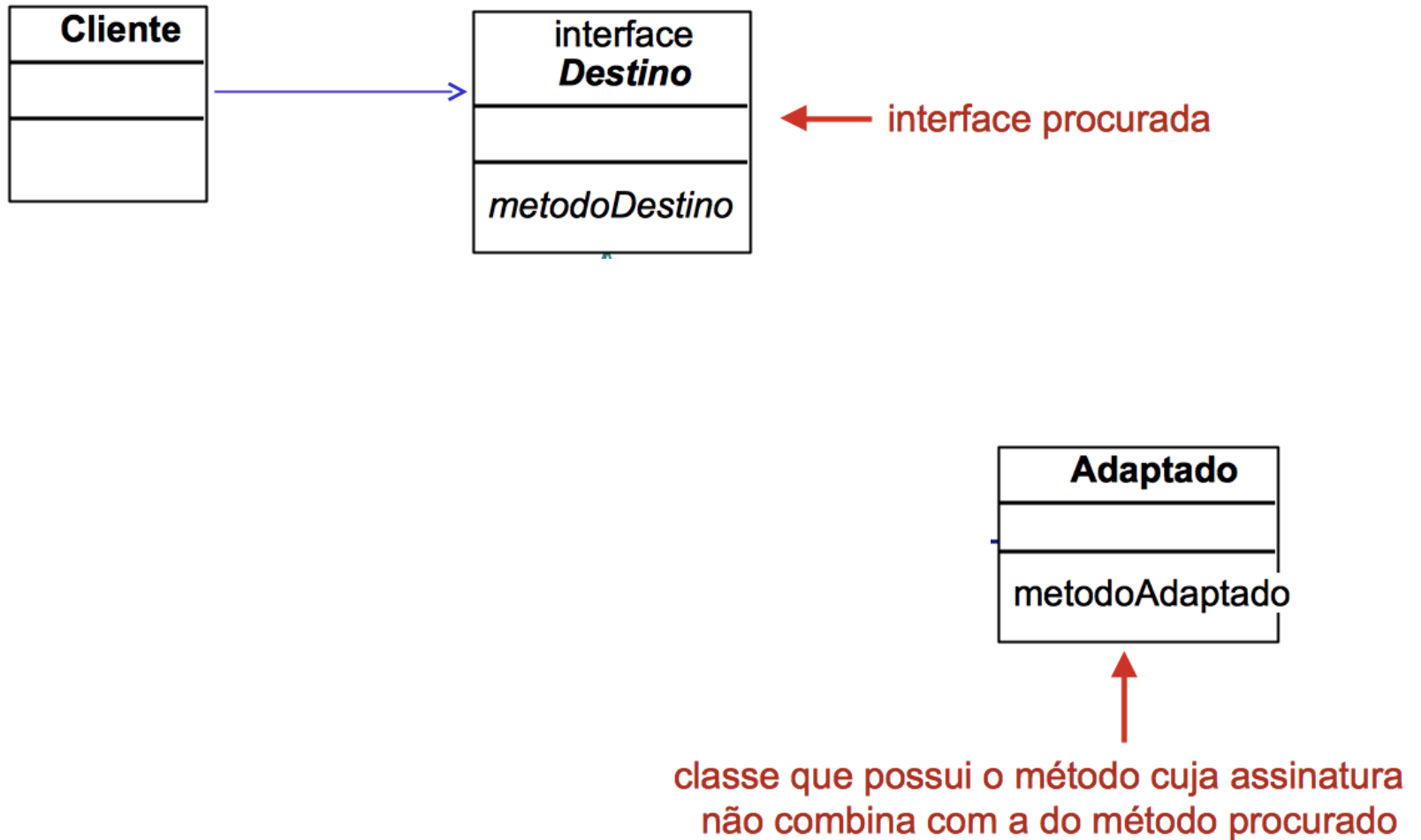
Problema: Como resolver interfaces incompatíveis?

Solução: Converta a interface original de um componente em outra interface, através de um objeto adaptador intermediário.

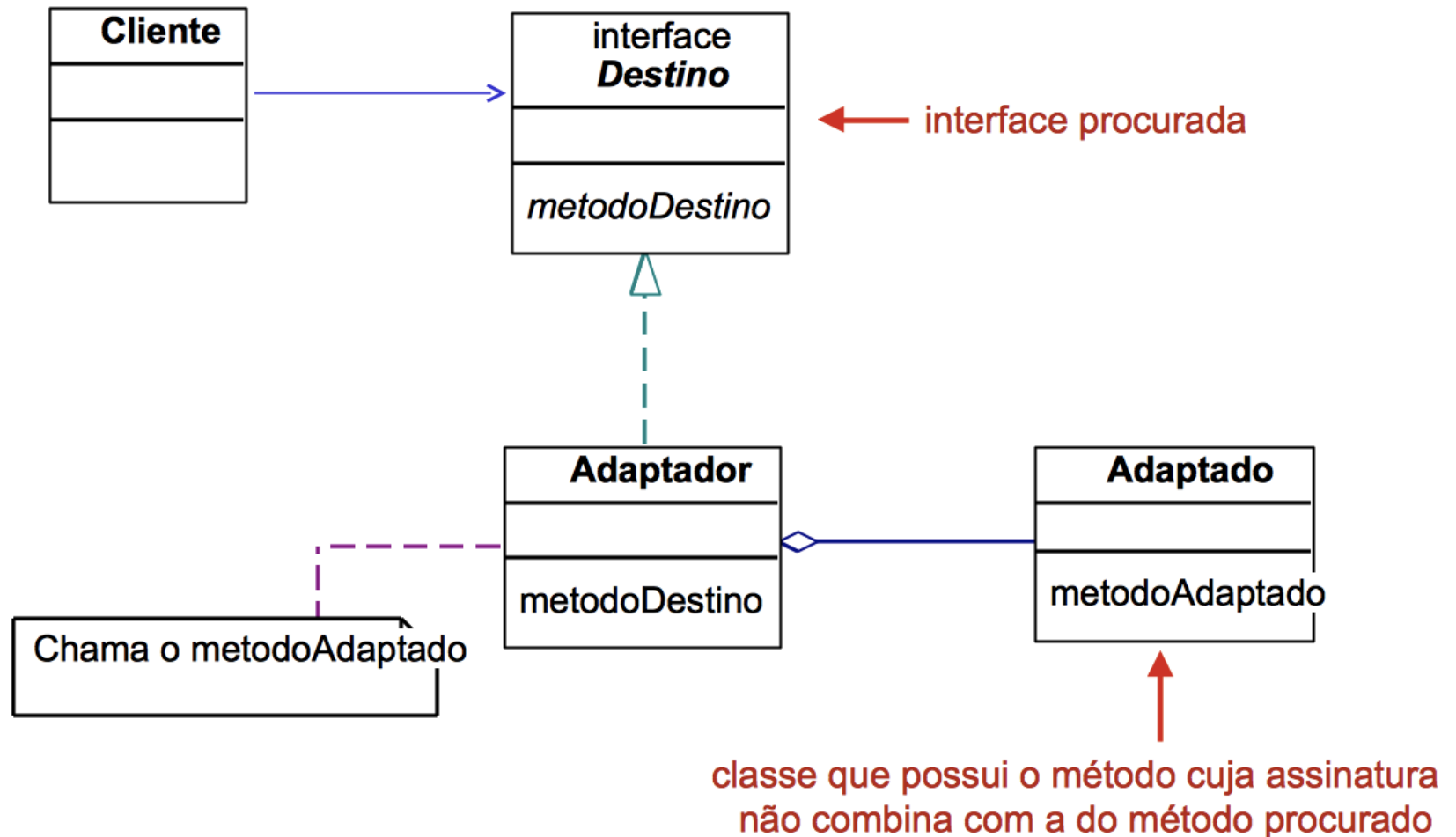
↳ A classe que tem a interface desejada é definida e então ela se comunica com a classe que tem uma interface diferente.

→ Também conhecido como Wrapper.

Adaptador



Adaptador



Adaptador

Exemplo 1:

- Programa que usa a classe List da biblioteca AWT será reescrito usando a classe JList da biblioteca Swing.
- Muitos métodos das classes da biblioteca AWT são iguais aos da biblioteca Swing, mas vários métodos da classe JList são diferentes da classe List.

Classe AWT List	Classe Swing JList
add (String)	-
remove (String)	-
String[] getSelectedItems ()	Object[] getSelectedValues ()

Adaptador

Exemplo 1: o nome fornecido é inserido na lista da esquerda e é possível mover um nome da lista da esquerda para a lista da direita e vice-versa.

```
public class JTwoList extends Frame
implements ActionListener {
    private Button Add, MoveRight, MoveLeft;
    private List leftList, rightList;
    private TextField txt;

    //adiciona um nome à list box da esquerda
    private void addName() {
        if (txt.getText().length() > 0) {
            leftList.add(txt.getText());
            txt.setText("");
        }
    }
}
```

Adaptador

Exemplo 1:

```
//adiciona um nome da list box esquerda para a list box direita
private void moveNameRight() {
    String sel[] = leftList.getSelectedItems();
    if (sel != null) {
        rightList.add(sel[0]);
        leftList.remove(sel[0]);
    }
}

// adiciona um nome da list box direita para a list box esquerda
public void moveNameLeft() {
    String sel[] = rightList.getSelectedItems();
    if (sel != null) {
        leftList.add(sel[0]);
        rightList.remove(sel[0]);
    }
}
```

Adaptador

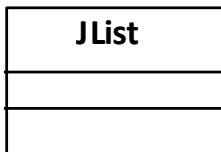
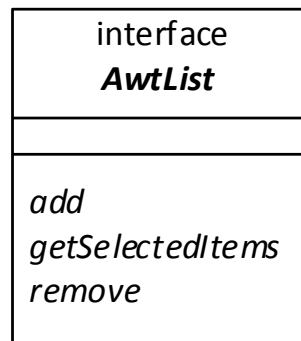
Exemplo 1:

- Para resolver o problema, é criada uma classe (adaptador) que emula a classe List e que precisa de somente 3 métodos (add, remove e getSelectedItems).

1. Define-se uma interface (AwtList).
2. Define-se uma classe (JAwtList) que implementa a interface (AwtList) e que contém um objeto da “nova” classe (JList)
3. A aplicação deve referenciar a nova classe (JAwtList) e não mais a “antiga” classe (List)

Adaptador

Exemplo 1: 1. Define-se uma interface.



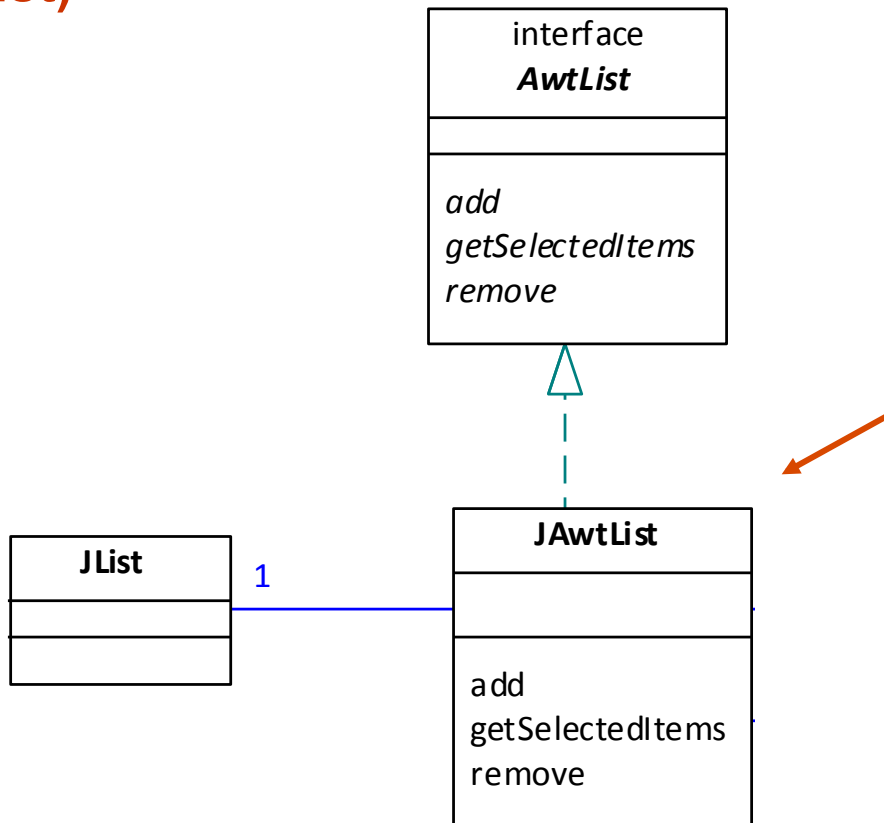
Adaptador

Exemplo 1: 1. Define-se uma interface.

```
public interface AwtList
{
    public void add(String s);
    public void remove(String s);
    public String[] getSelectedItems();
}
```

Adaptador

Exemplo 1: 2. Define-se uma classe (JAwList) que implementa a interface (AwList) e que contém um objeto da “nova” classe (JList)



Adaptador

Exemplo 1: 2. Define-se uma classe que implementa a interface (AwtList) e que contém um objeto da “nova” classe (Jlist)

```
public class JAwtList extends JScrollPane
implements ListSelectionListener, AwtList {
    private JList listWindow;
    private DefaultListModel listContents;

    public JAwtList(int rows) {
        listContents = new DefaultListModel ();
        listWindow = new JList(listContents);
        listWindow.setPrototypeCellValue("Abcdefg Hijkmnop");
        getViewport().add(listWindow);
    }
}
```

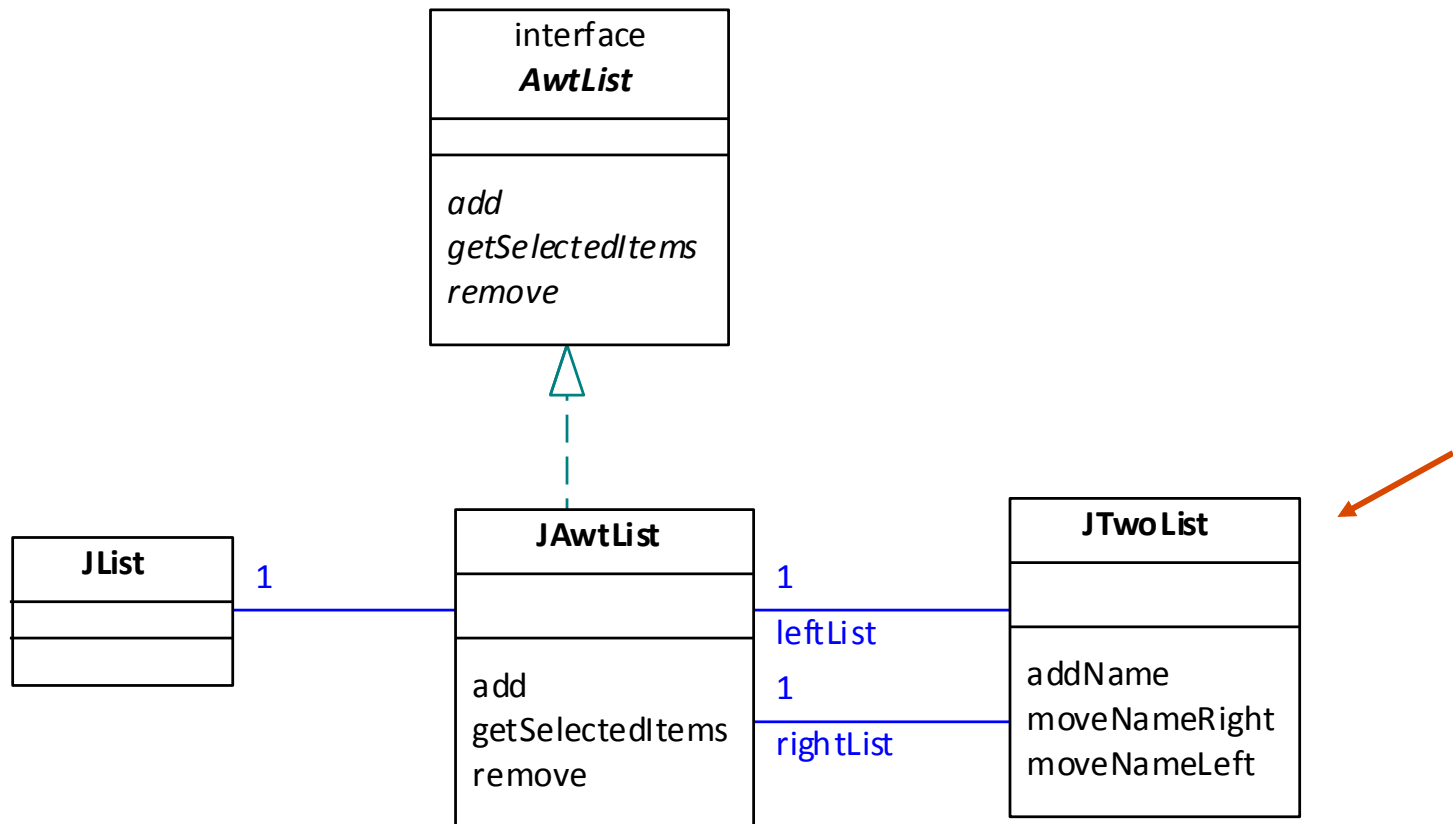
Adaptador

Exemplo 1:

```
//-----  
    public void add(String s) {  
        listContents.addElement(s);  
    }  
//-----  
    public void remove(String s) {  
        listContents.removeElement(s);  
    }  
//-----  
    public String[] getSelectedItems() {  
        Object[] obj = listWindow.getSelectedValues();  
        String[] s = new String[obj.length];  
        for (int i =0; i < obj.length; i++)  
            s[i] = obj[i].toString();  
        return s;  
    }  
}
```

Adaptador

Exemplo 1: 3. A aplicação deve referenciar a nova classe (JAwtList) e não mais a “antiga” classe (List)



Adaptador

Exemplo 1: 3. A aplicação deve referenciar a nova classe (JAwList) e não mais a “antiga” classe (List)

```
public class JTwoList extends Frame
implements ActionListener {
    private JButton Add, MoveRight, MoveLeft;
    private JAwList leftList, rightList;
    private TextField txt;

    //adiciona um nome à list box da esquerda
    private void addName() {
        if (txt.getText().length() > 0) {
            leftList.add(txt.getText());
            txt.setText("");
        }
    }
}
```

Adaptador

Exemplo 1:

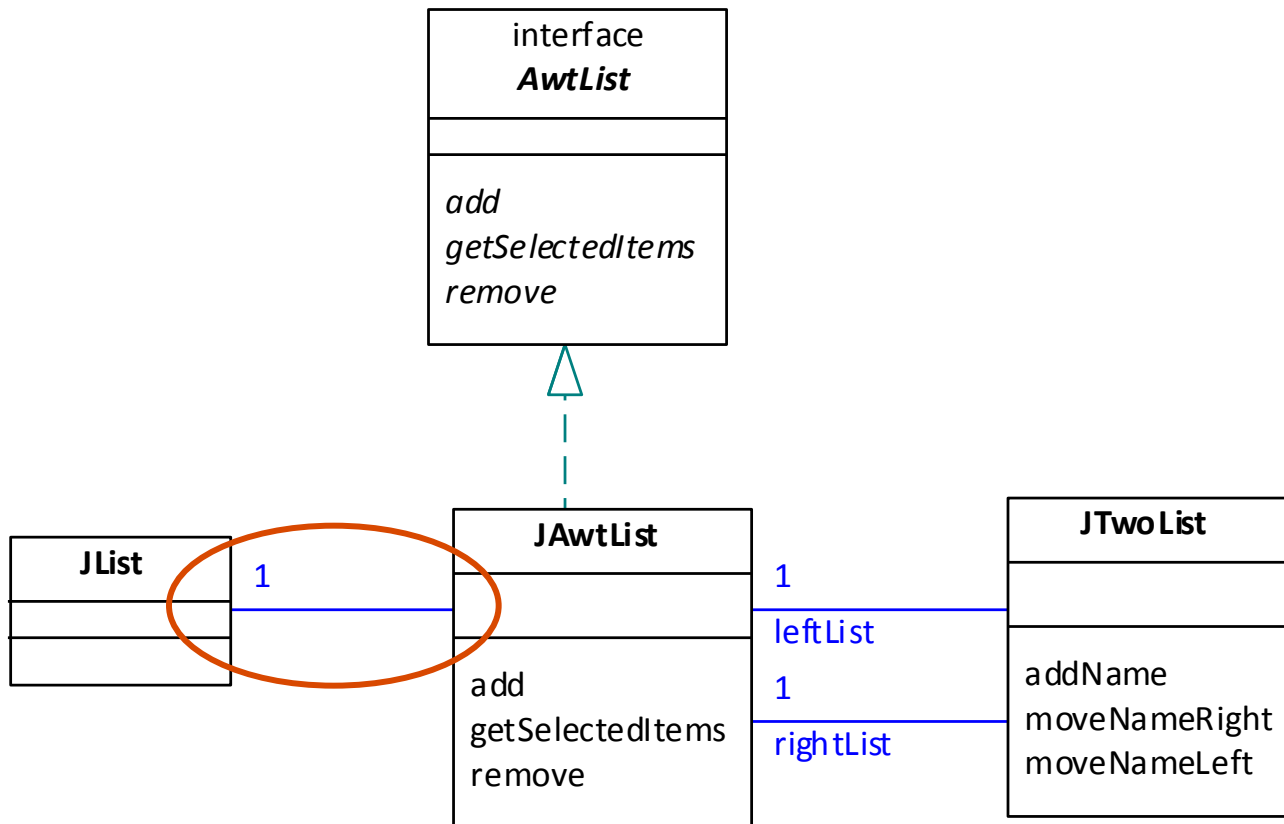
```
//adiciona um nome da list box esquerda para a list box direita
private void moveNameRight() {
    String sel[] = leftList.getSelectedItems();
    if (sel != null) {
        rightList.add(sel[0]);
        leftList.remove(sel[0]);
    }
}

// adiciona um nome da list box direita para a list box esquerda
public void moveNameLeft() {
    String sel[] = rightList.getSelectedItems();
    if (sel != null) {
        leftList.add(sel[0]);
        rightList.remove(sel[0]);
    }
}
```


Adaptador

Exemplo 1:

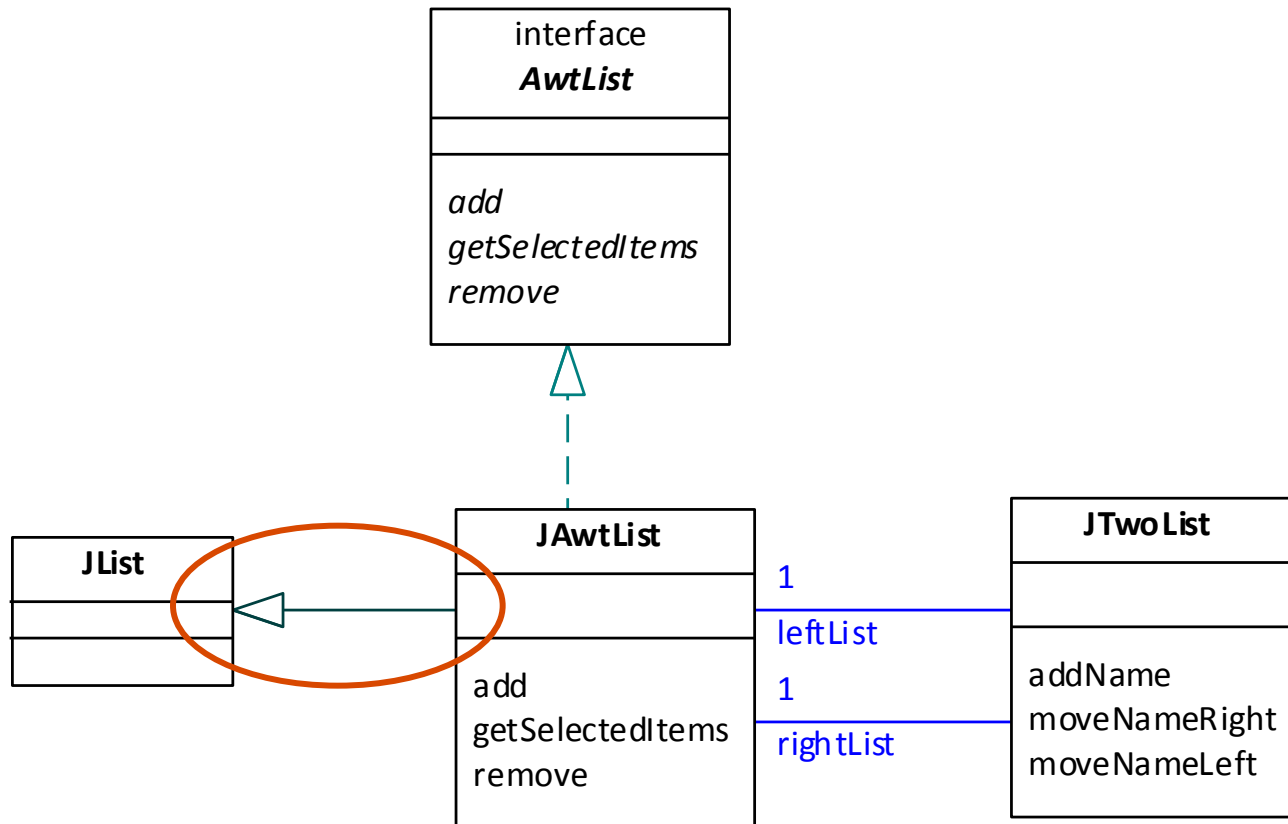
→ Implementação por composição



Adapter

Exemplo 1:

→ Implementação por herança



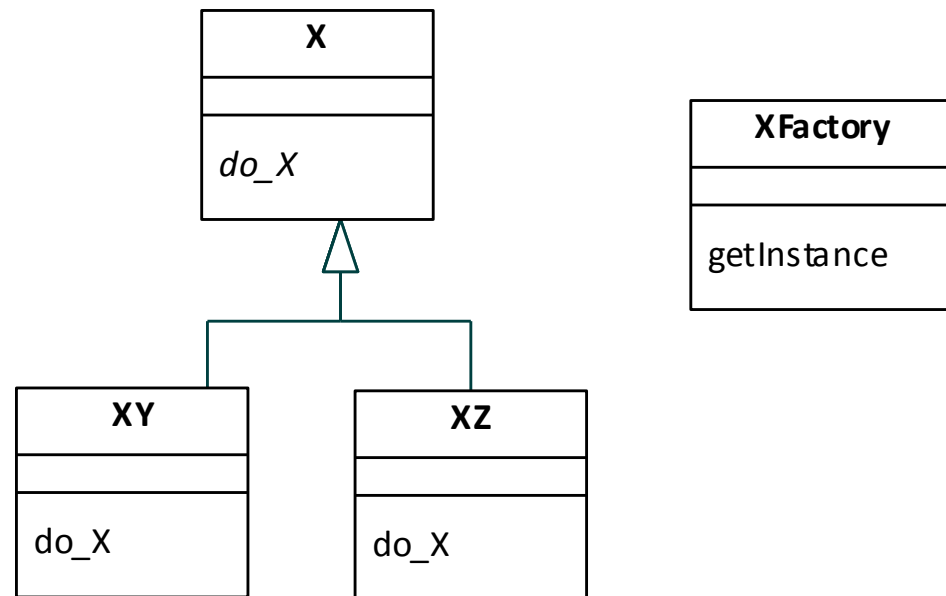
3. Factory (Fábrica)

Problema: Quem deve ser o responsável pela criação de objetos onde a escolha da classe depende de um dado fornecido?

Solução: Crie um objeto chamado Factory que retorna uma instância de classe entre várias classes possíveis, dependendo do dado fornecido.

- As classes das instâncias que o Factory retorna possuem métodos em comum, mas com implementações distintas.

Factory

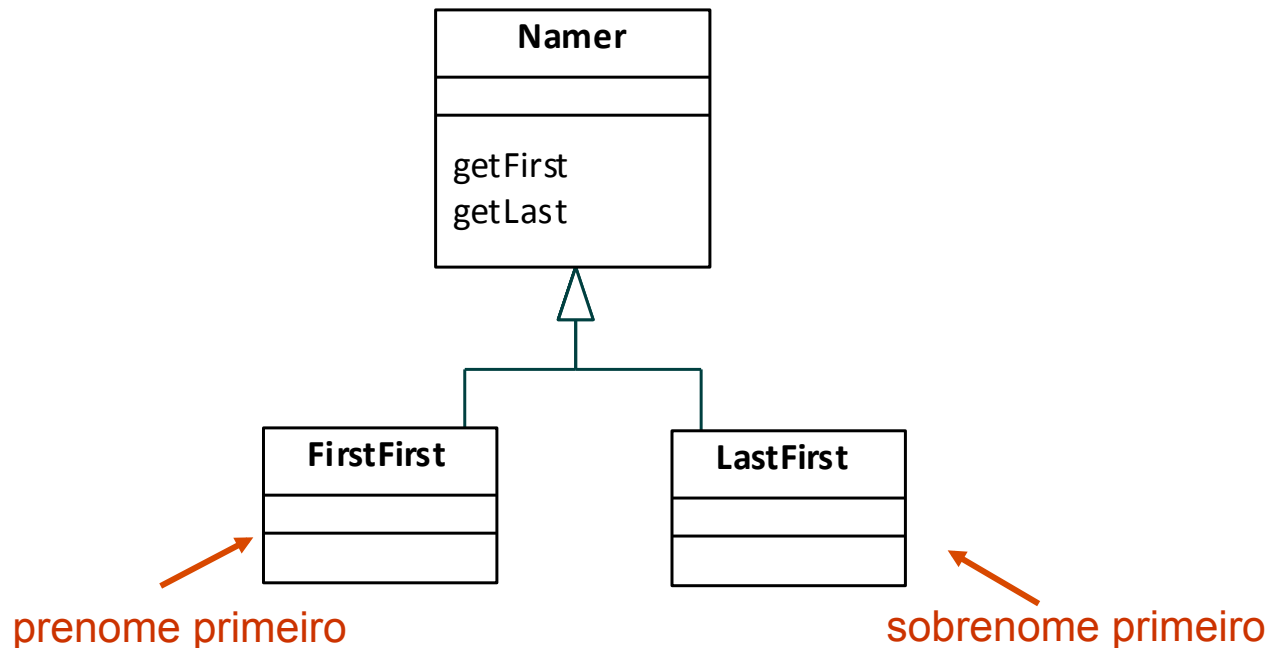


Factory

Exemplo 1:

- Você tem um *form* de entrada e deseja permitir que o usuário entre com seu nome como “nome sobrenome” ou “sobrenome, nome”.

A existência da vírgula identifica o segundo caso.



Factory

Exemplo 1:

```
public class Namer {  
    // classe base  
    protected String last;  
    protected String first;  
  
    public String getFirst() {  
        return first;           //return first name  
    }  
    public String getLast() {  
        return last;           //return last name  
    }  
}
```

Factory

Exemplo 1:

```
public class FirstFirst extends Namer {  
    // considera que tudo que vem antes do último espaço faz parte  
    // do primeiro nome.  
  
    public FirstFirst(String s) {  
        int i = s.lastIndexOf(" "); //find separate space  
        if (i>0) {  
            first = s.substring(0,i).trim();  
            last = s.substring(i+1).trim();  
        } else {  
            first = "";  
            last = s;    // if no space put all in last name  
        }  
    }  
}
```

Factory

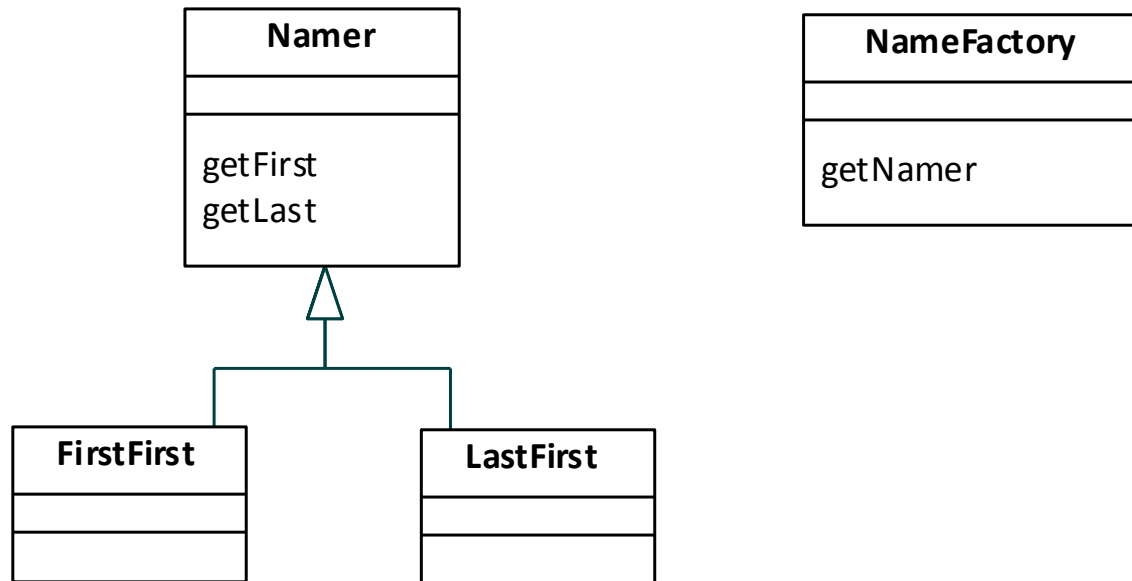
Exemplo 1:

```
public class LastFirst extends Namer {  
    // considera que a vírgula separa o sobrenome.  
  
    public LastFirst(String s) {  
        int i = s.indexOf(","); //find comma  
        if (i > 0) {  
            last = s.substring(0,i).trim();  
            first = s.substring(i+1).trim();  
        } else {  
            last = s;           //if no comma, put all in last name  
            first = "";  
        }  
    }  
}
```


Factory

Exemplo 1: Quem será o responsável em criar um instância da classe FirstFirst ou LastFirst?

Uma factory pode decidir a classe dependendo da existência da vírgula.



Factory

Exemplo 1:

```
public class NameFactory {  
    // A Factory decide a classe dependendo da existência da vírgula  
  
    public Namer getNamer(String entry) {  
        int i = entry.indexOf(",");  
        if (i > 0)  
            return new LastFirst(entry);  
        else  
            return new FirstFirst(entry);  
    }  
}
```

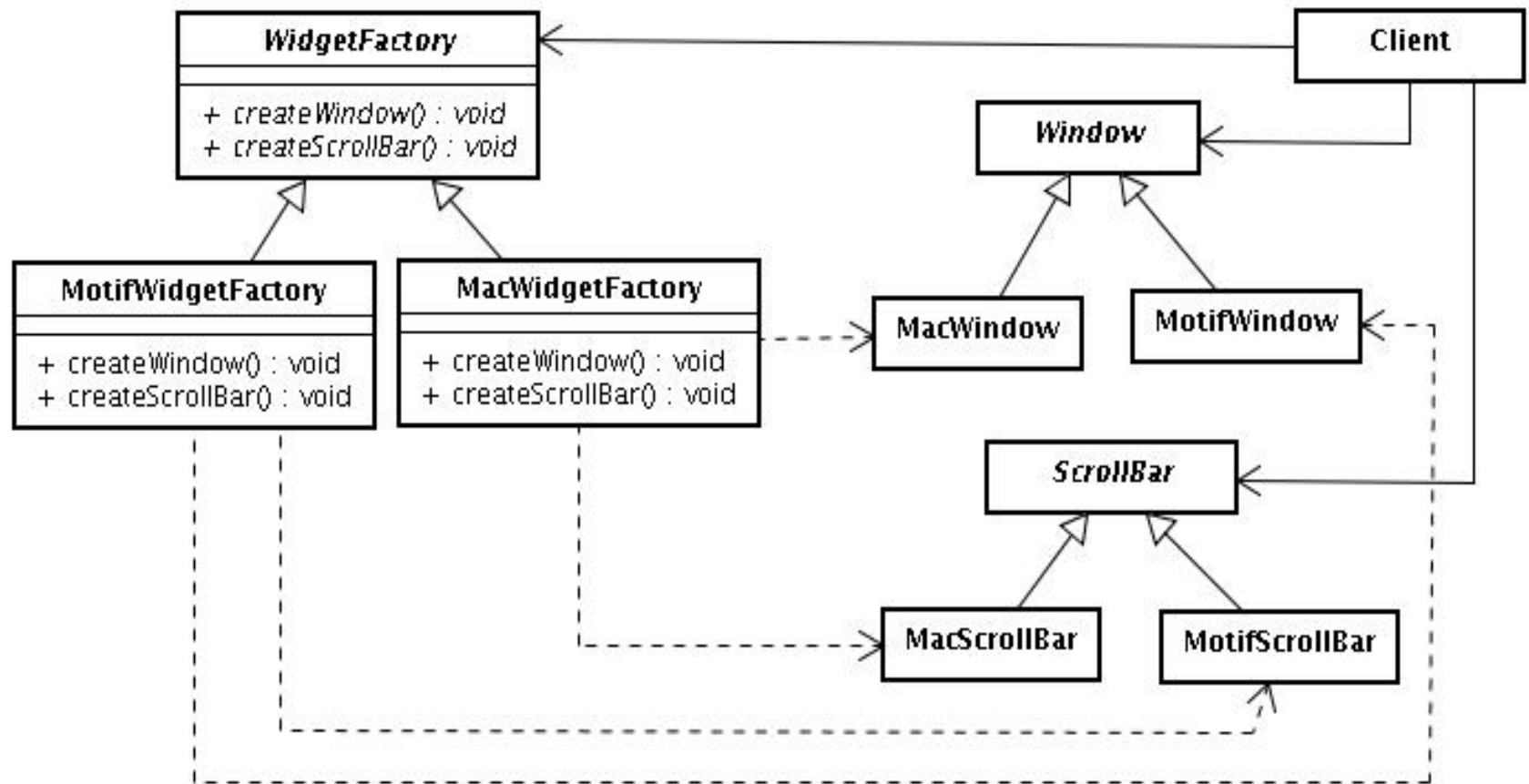
3. Fábrica Abstrata (Abstract Factory)

Problema: Como criar famílias de classes relacionadas que implementem uma interface comum sem especificar suas classes concretas?

Exemplo:

- Uma aplicação precisa dar suporte a várias “implementações” de interface gráfica com o usuário (look-and-feel user interface), como Windows, Motif e Mac.
- Solução: Você especifica que quer uma aplicação que seja executada com a interface do Mac e obtém uma fábrica de objetos GUI do tipo Mac. Quando for necessário requisitar objetos específicos, como botões, check boxes e janelas, a fábrica retornará instâncias Mac destes componentes.

Fábrica Abstrata (Abstract Factory)



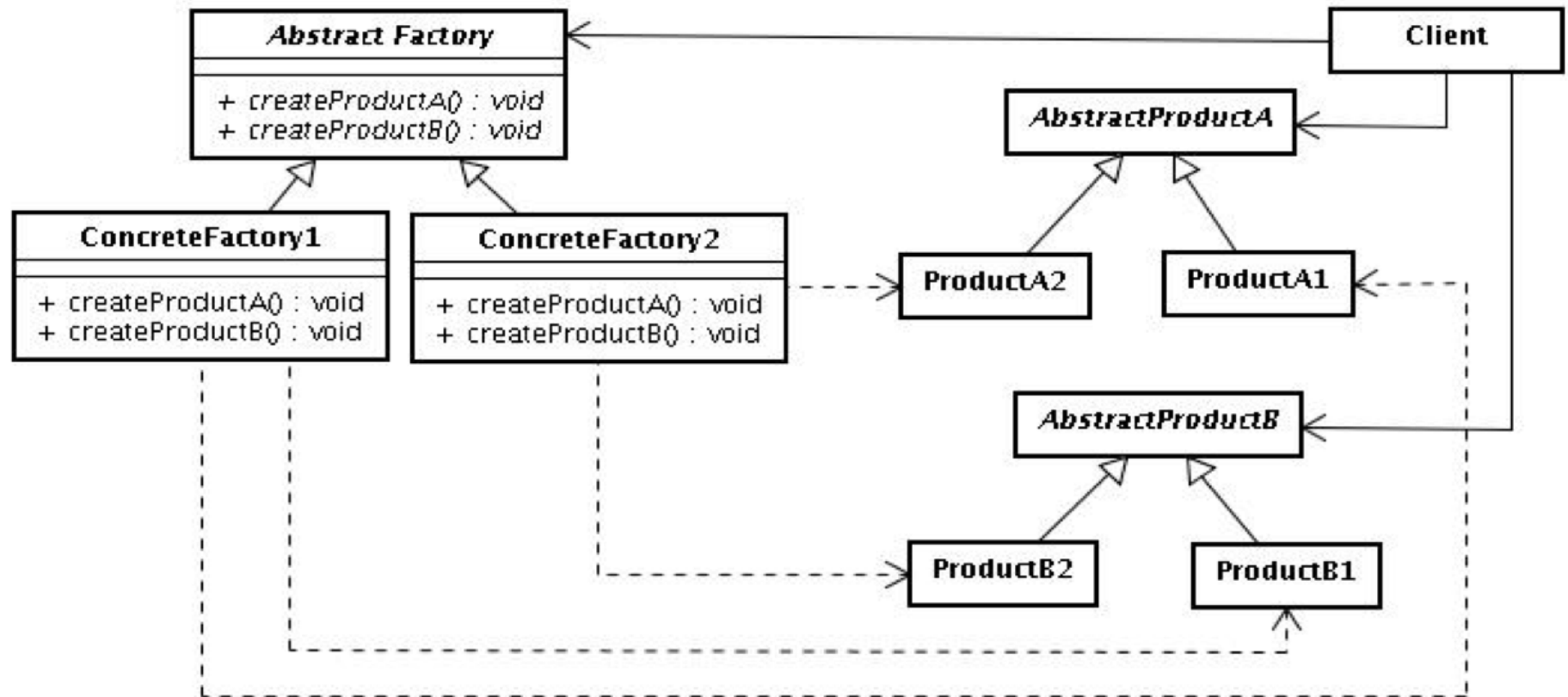
Fábrica Abstrata (Abstract Factory)

Problema: Como criar famílias de classes relacionadas que implementem uma interface comum sem especificar suas classes concretas?

Solução: Defina uma classe abstrata que forneça serviços comuns para as fábricas concretas que a estendem. Defina uma classe de fábrica concreta para cada família de itens a criar.

Opcionalmente, defina uma interface ao invés de uma classe abstrata.

Fábrica Abstrata (Abstract Factory)



Fábrica Abstrata (Abstract Factory)

Exemplo 2:

- Programa para planejar o layout de jardins. Os jardins podem ser jardins anuais, jardins de vegetais (hortas) ou jardins perenes. Independente do tipo de jardim, temos as mesmas perguntas:
 - Quais plantas são boas para ficarem no centro?
 - Quais plantas são boas para ficarem no canteiro?
 - Quais plantas são boas para ficarem na sombra?
- Precisamos de uma classe Garden (Jardim) que responda estas perguntas:

Fábrica Abstrata (Abstract Factory)

Exemplo 2:

```
public interface Garden {  
    public Plant getShade();  
    public Plant getCenter();  
    public Plant getBorder();  
}
```

- A interface Garden é a Fábrica Abstrata de Plantas.

Fábrica Abstrata (Abstract Factory)

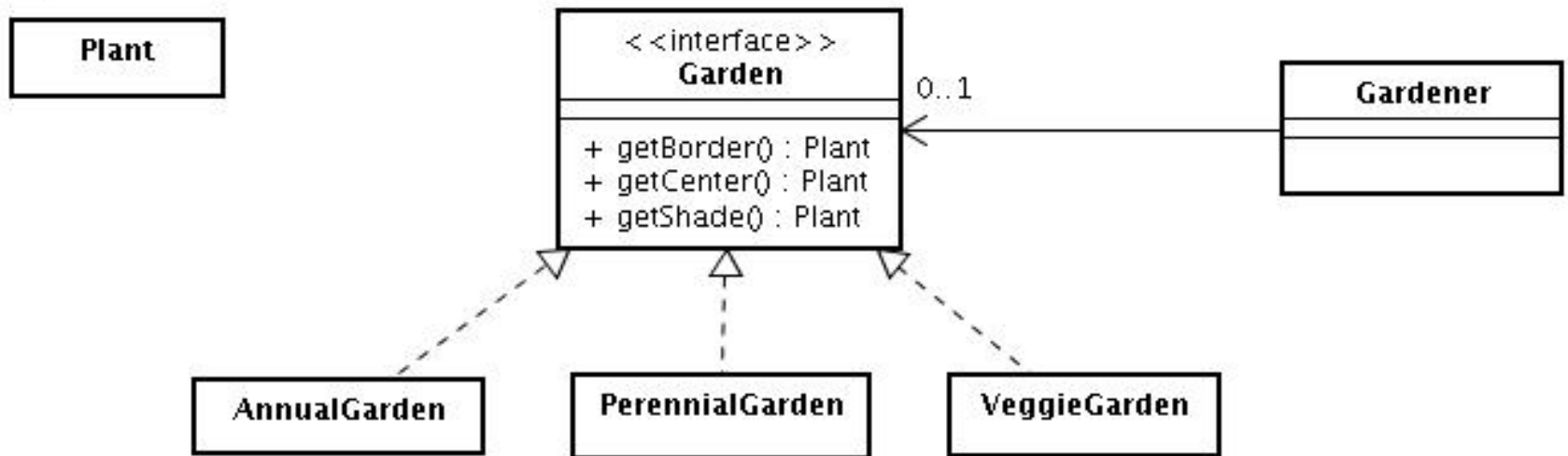
Exemplo 2:


```
public class VegieGarden implements Garden {  
  
    public Plant getShade() {  
        return new Plant("Broccoli");  
    }  
    public Plant getCenter() {  
        return new Plant("Corn");  
    }  
    public Plant getBorder() {  
        return new Plant("Peas");  
    }  
}
```

- A classe VegieGarden é uma Fábrica Concreta de Plantas.

Fábrica Abstrata (Abstract Factory)

Exemplo 2:





Fábrica Abstrata (Abstract Factory)

Vantagem:

- As classes concretas são isoladas. Os nomes reais das classes não precisam ser conhecidos no cliente.
- uma nova classe (factory) pode ser facilmente adicionada.
- as classes usadas na aplicação podem ser trocadas sem mudanças no código do cliente.

4. Estratégia (Strategy)

Problema: Como projetar um programa que requer um serviço e existem várias maneiras de executar este serviço, e o serviço escolhido pode ser modificado durante a execução do programa?

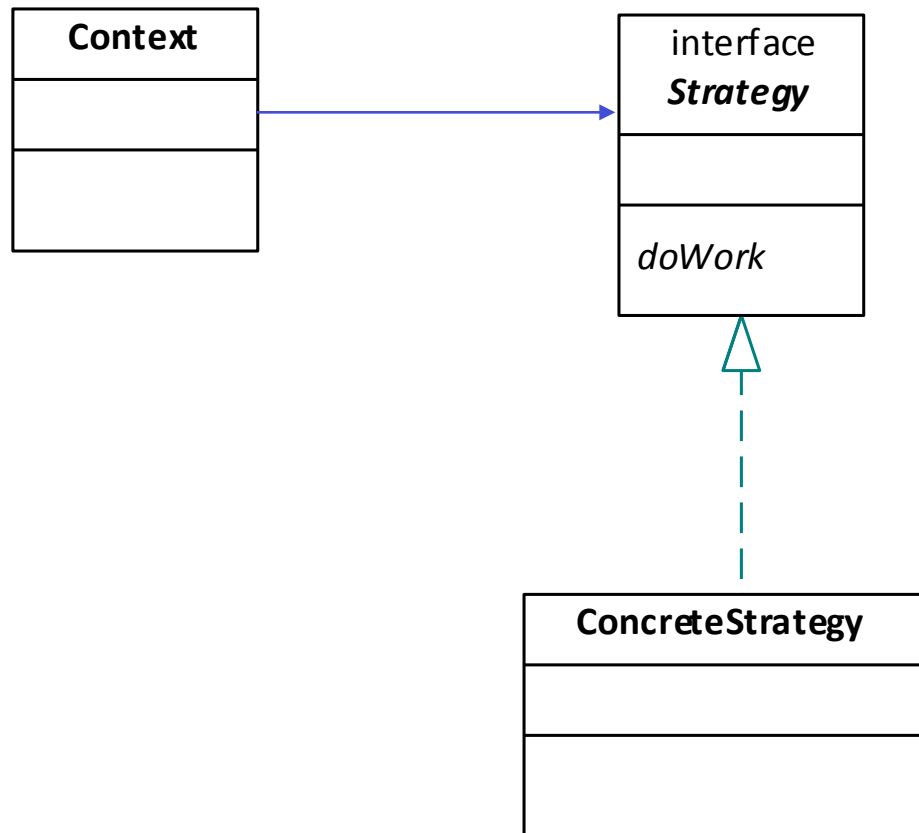
Solução: Defina cada algoritmo/estratégia em uma classe separada, com uma interface comum.

Uma classe “contexto” escolhe entre estes algoritmos baseados na escolha do usuário ou na eficiência computacional.

Exemplos:

- salvar arquivos em diferentes formatos; comprimir arquivos usando diferentes algoritmos; “plotar” o mesmo dado em diferentes formatos: line graph, bar chart, pie chart.

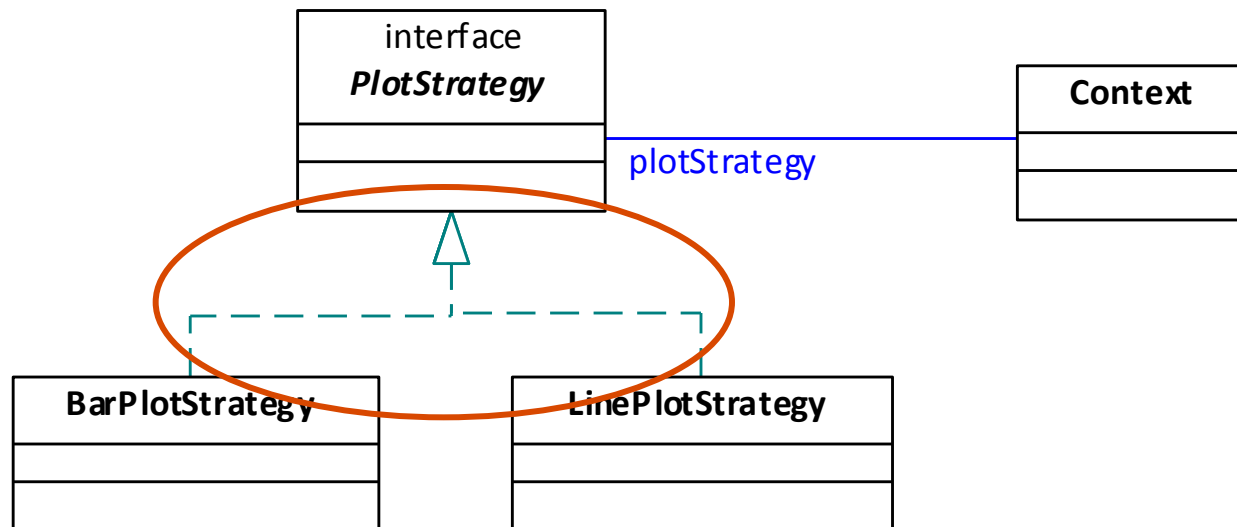
Estratégia



Estratégia

Exemplo 1: Programa gráfico que pode apresentar dados como um gráfico ou como um diagrama em barra.

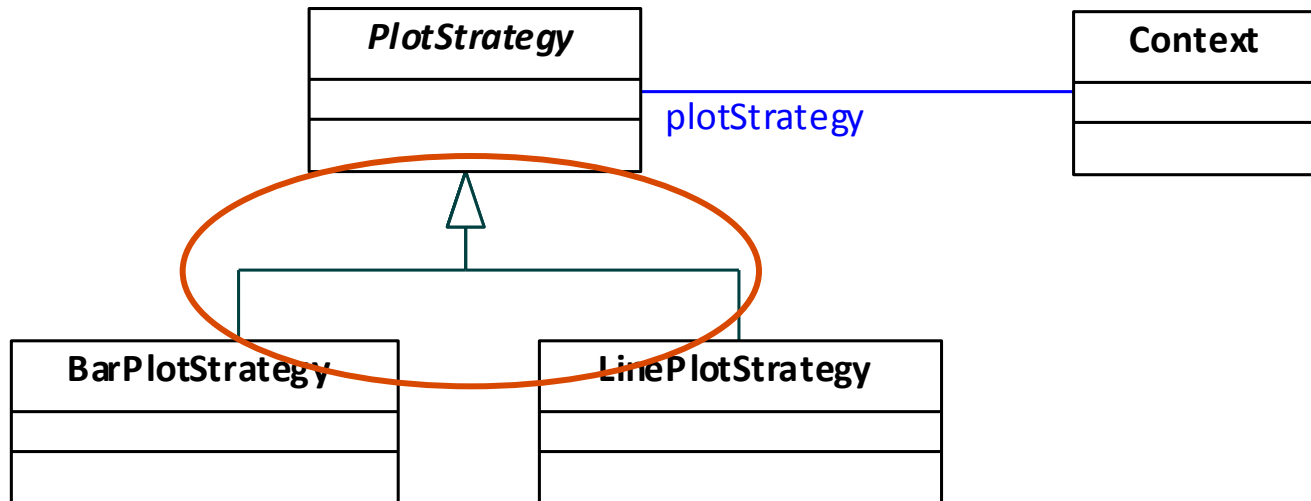
→ Solução com interface:



Estratégia

Exemplo 1:

→ Solução com herança:



Estratégia

Exemplo 1: Classe PlotStrategy

```
public abstract class PlotStrategy extends JFrame {  
    protected float[] x, y;  
    protected float minX, minY, maxX, maxY;  
    protected int width, height;  
    protected Color color;  
  
    public PlotStrategy(String title) {  
        super(title);  
        width = 300;  
        height = 200;  
        color = Color.black;  
        addWindowListener(new WindAp(this));  
    }  
  
    public abstract void plot(float xp[], float yp[]);  
    ...  
}
```

 todas as subclasses devem implementar este método

Estratégia

Exemplo 1: Subclasse LinePlotStrategy

```
public class LinePlotStrategy extends PlotStrategy {  
    private LinePlotPanel lp;  
  
    public LinePlotStrategy() {  
        super("Line plot");  
        lp = new LinePlotPanel();  
        getContentPane().add(lp);  
    }  
  
    public void plot(float[] xp, float[] yp) {  
        x = xp; y = yp;  
        findBounds();  
        setSize(width, height);  
        setVisible(true);  
        setBackground(Color.white);  
        lp.setBounds(minX, minY, maxX, maxY);  
        lp.plot(xp, yp, color);  
        repaint();  
    }  
}
```

Estratégia

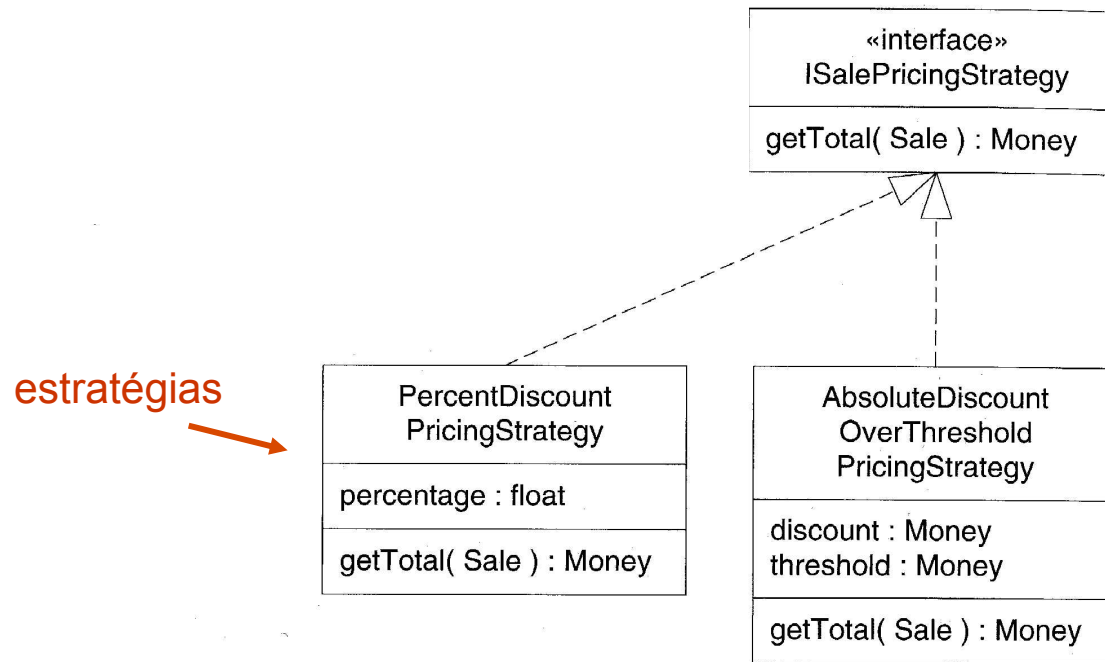
Exemplo 1: Classe Contexto que decide a estratégia a ser chamada.

```
public class Context {  
    // seleciona uma das estratégias para desenhar  
    private PlotStrategy plotStrategy; ← referêcia à superclasse  
    private float x[], y[];  
  
    public Context() {  
        setLinePlot();  
    }  
    public void setBarPlot() {  
        plotStrategy = new BarPlotStrategy();  
    }  
    public void setLinePlot() {  
        plotStrategy = new LinePlotStrategy();  
    }  
    public void plot() {  
        plotStrategy.plot(x, y);  
    }  
}
```

Estratégia

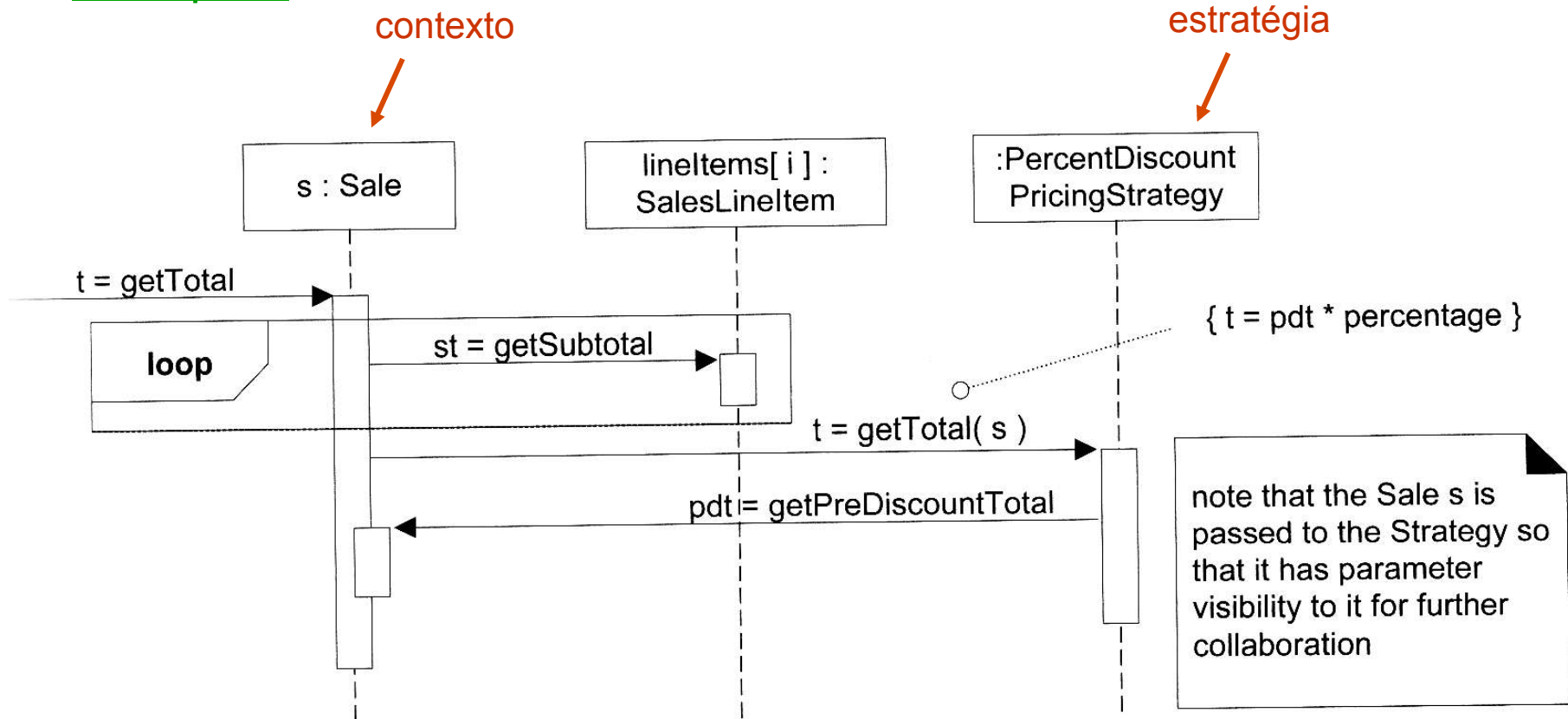
Exemplo 2 (livro do Larman):

O objeto estratégia é anexado a um objeto do contexto - o objeto sob o qual o algoritmo será aplicado.



Estratégia

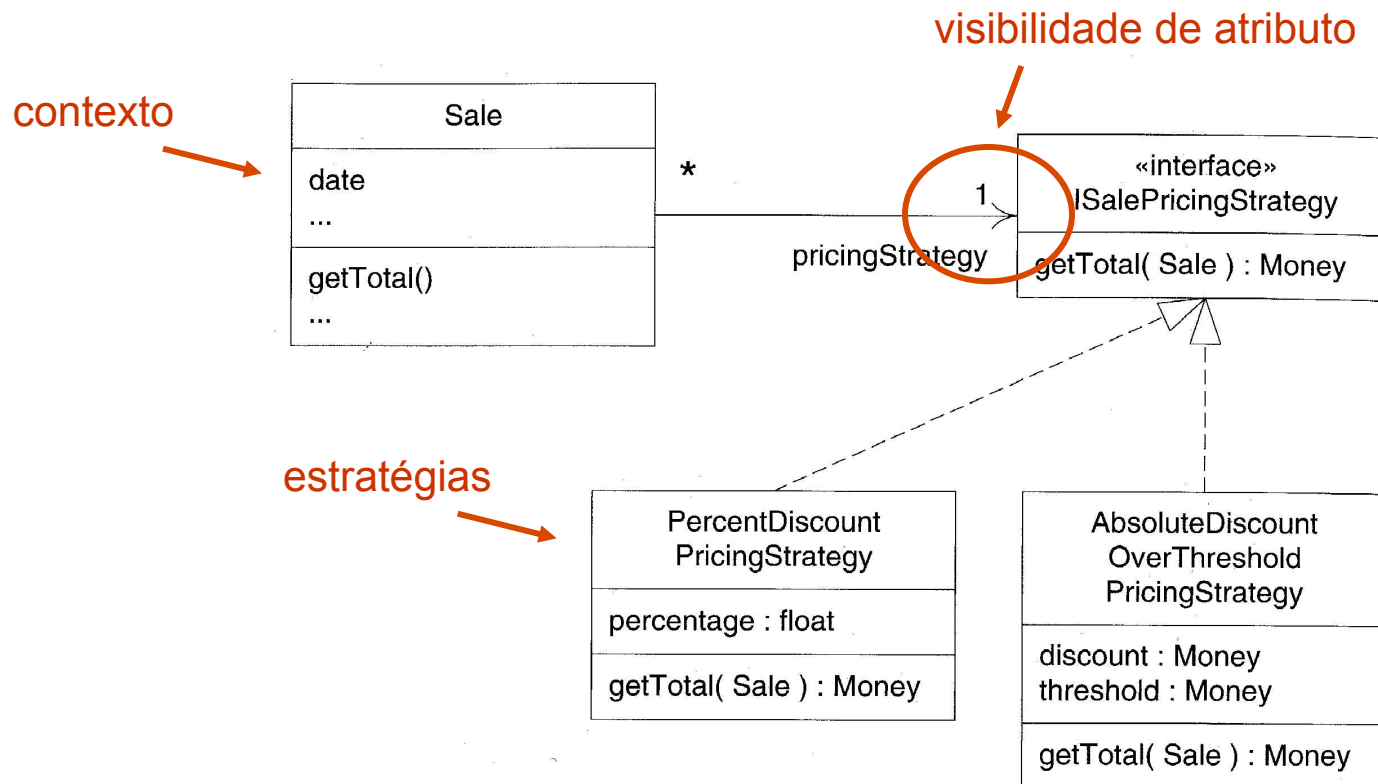
Exemplo 2:



➔ O objeto do contexto requer uma visibilidade de atributo.

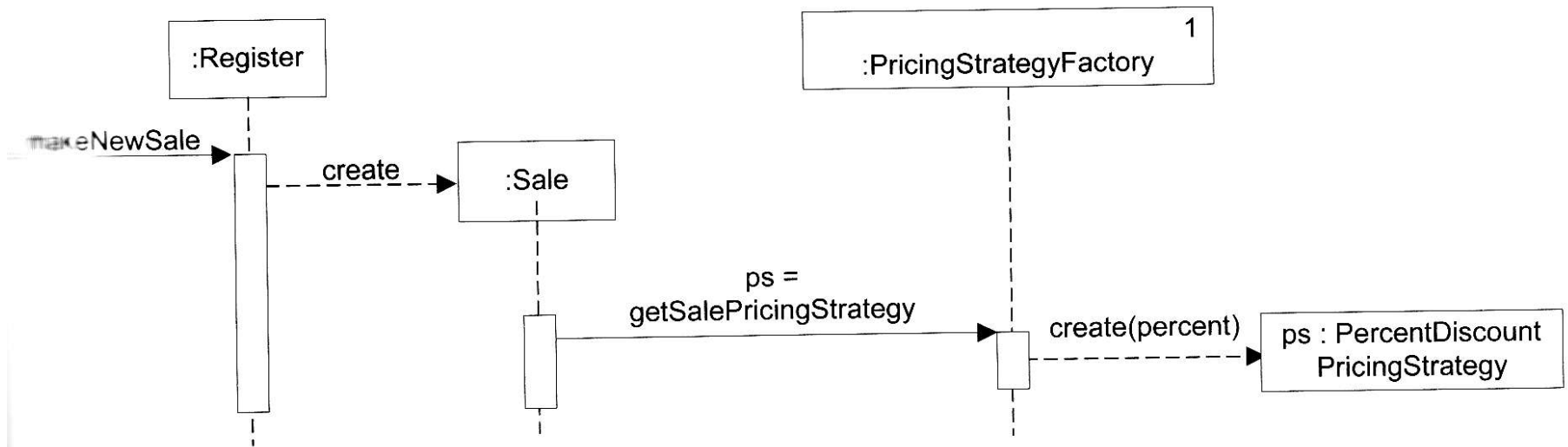
Estratégia

Exemplo 2:



Estratégia

Exemplo 2:



- Quando uma instância de Venda é criada, ela pede para a factory a sua estratégia de preço.

5. Singleton

Problema: Somente uma instância de uma classe é permitida (chamada de singleton) e ela deve ser acessível aos clientes de um ponto de acesso global e único.

Solução: Defina um método de classe que retorna o singleton e defina o construtor como um método privado.

Exemplos: uma única janela de gerenciamento, um único print spooler.

Singleton

Implementação:

- Defina um construtor privado.
- Defina um atributo estático dentro da classe (atributo de classe) ao qual será atribuído a única instância criada.
- Defina um método estático público que chamará o construtor e retornará uma nova instância somente se o atributo estático ainda não estiver instanciado. Caso contrário, retorna a instância atribuída ao atributo estático.

Singleton

Exemplo 1: Classe spooler

```
public class PrintSpooler {  
    private static PrintSpooler spooler;  
  
    private PrintSpooler() {  
  
    public static PrintSpooler getSpooler() {  
        if (spooler == null)  
            spooler = new PrintSpooler();  
        return spooler;  
    }  
  
    public void print(String s) {  
        System.out.println(s);  
    }  
}
```

atributo estático

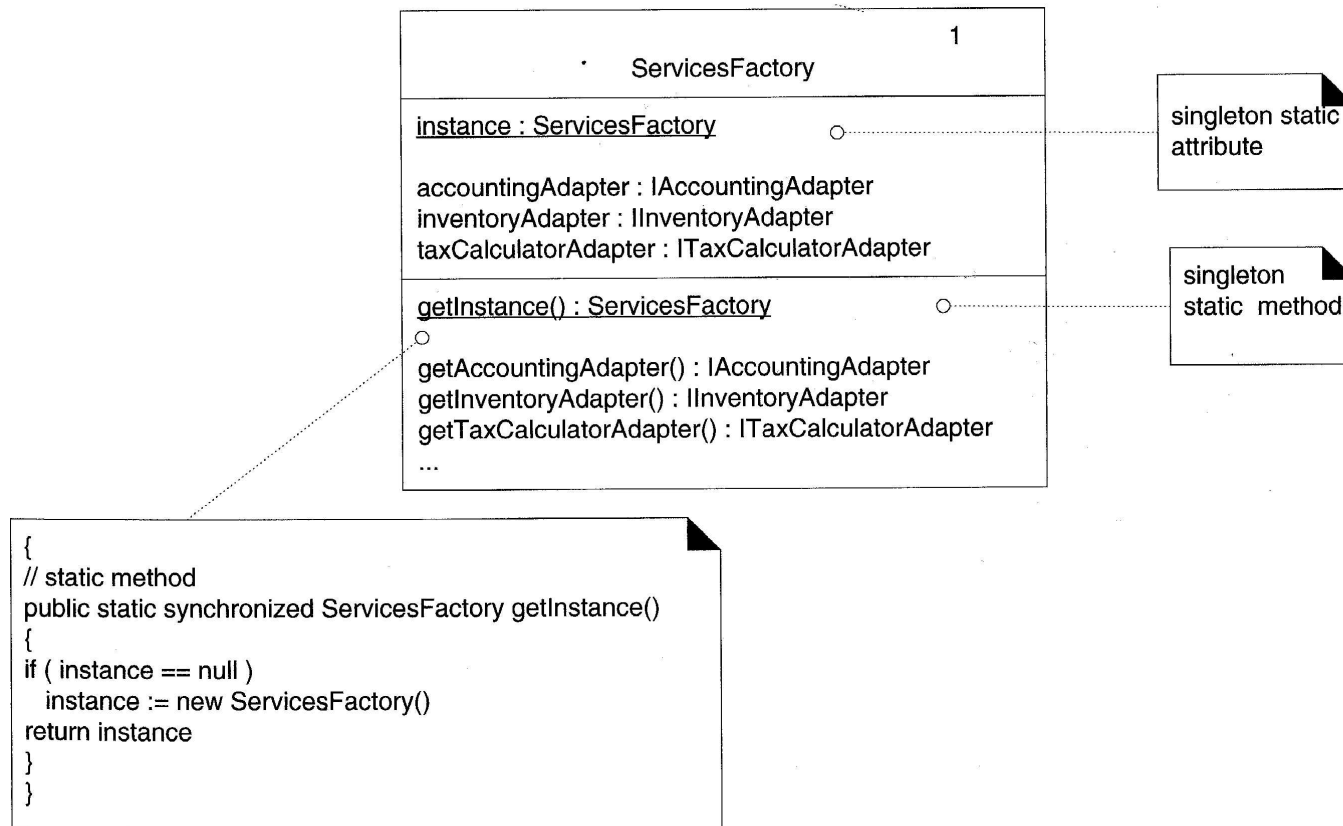
construtor privado

método estático público

Singleton

➡ Factories geralmente são acessadas com o padrão Singleton

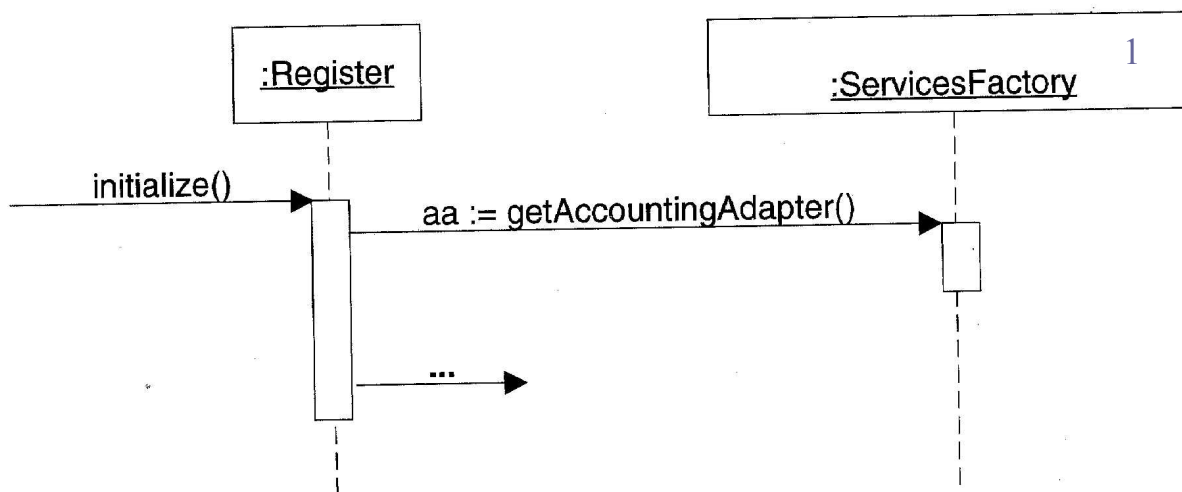
Exemplo 2 (livro do Larman):



Singleton

Singleton em um Diagrama de Interação

Exemplo:



- O “1” indica que a visibilidade a esta instância é obtida através do padrão Singleton.

Variação do Singleton

→ Se desejado, pode-se retornar null ou gerar uma exceção se a instância já havia sido criada.

Exemplo: Classe spooler com geração de exceção

```
public class Spooler {  
  
    static boolean instance_flag = false; //true se existe 1 instancia  
  
    public Spooler() throws RuntimeException  
    {  
        if (instance_flag)  
            throw new RuntimeException("Somente uma printer permitida");  
        else  
            instance_flag = true;  
    }  
}
```

6. Composite (Composto)

Problema: Como tratar um grupo ou composição de objetos da mesma maneira que um objeto individual?

Solução: Defina classes para os objetos compostos e individuais que implementem a mesma interface.

Aplicação:

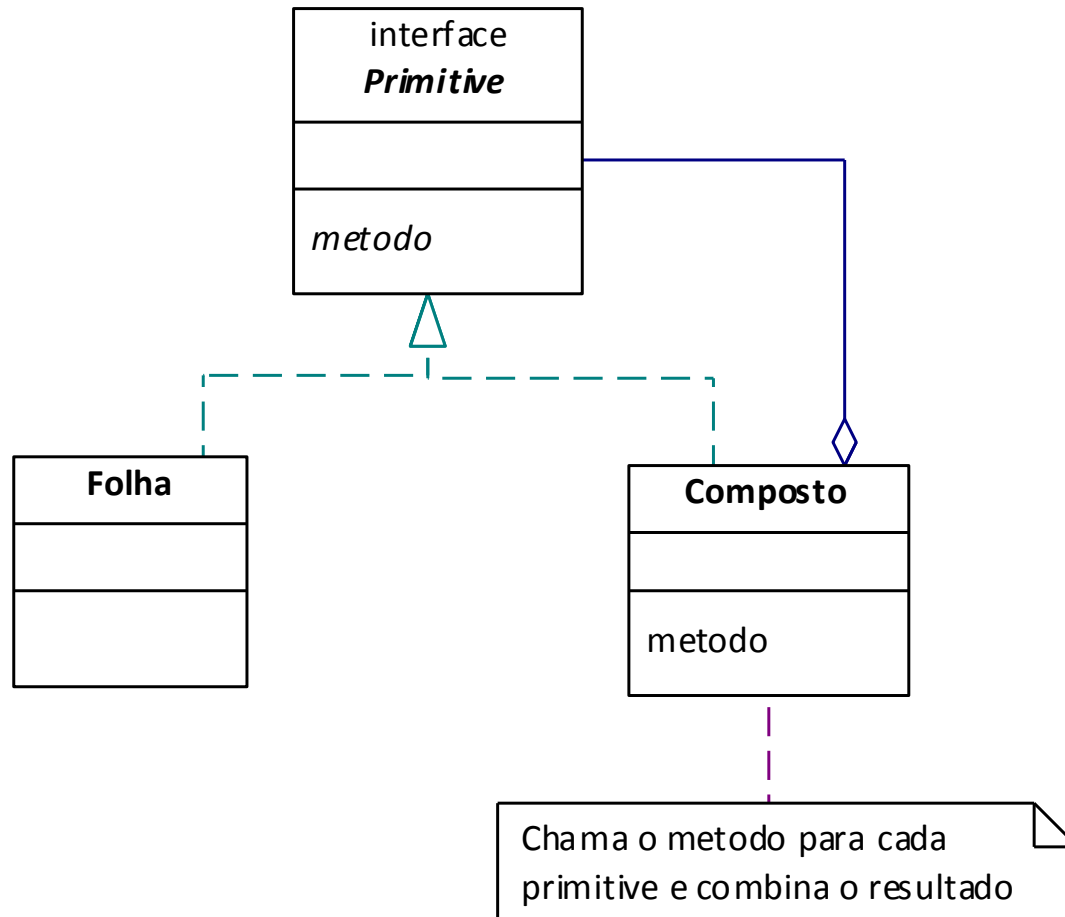
- Você quer representar hierarquias parte-todo de objetos.
- Você quer que os clientes ignorem a diferença entre composições de objetos e objetos individuais.

Composite

Folha

Composto
metodo

Composite





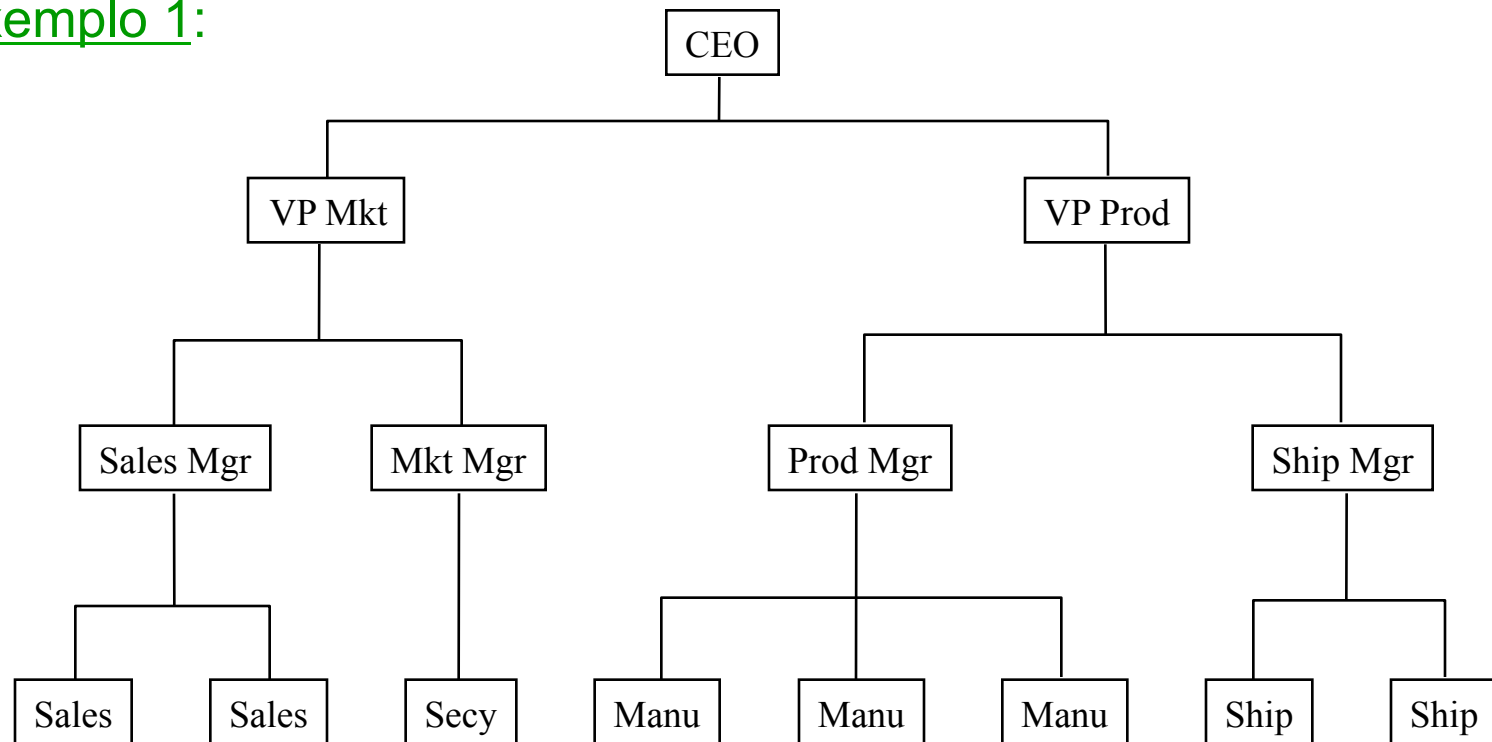
Composite

Exemplo 1: Considere uma empresa em que cada membro da empresa recebe um salário. O controle de custo é definido como o salário de um membro bem como os salários dos seus subordinados:

- O custo de um empregado individual é o seu salário.
- O custo de um empregado que é chefe de algum setor é o seu salário mais os salários dos empregados que ele controla.

Composite

Exemplo 1:

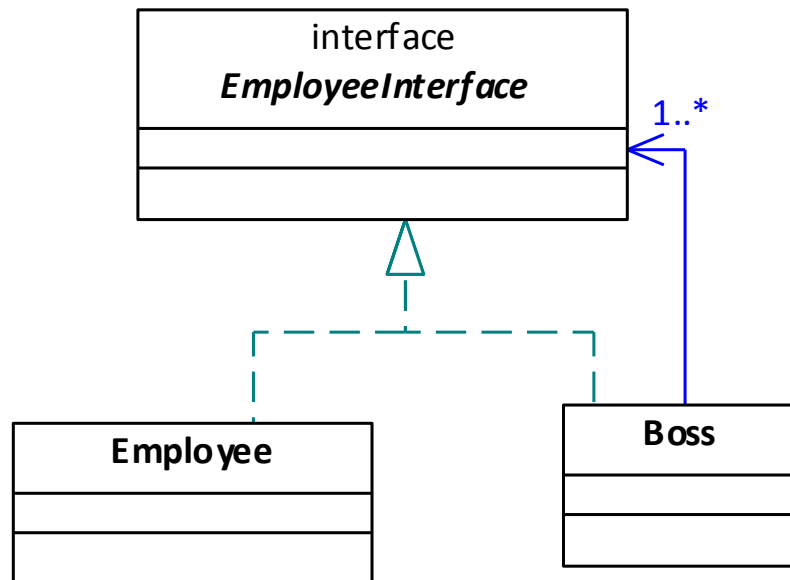


- Desejamos uma única interface que retorna o custo corretamente, independentemente do empregado ter subordinados ou não.

Composite

Exemplo 1: São definidas duas classes Empregado e Chefe.

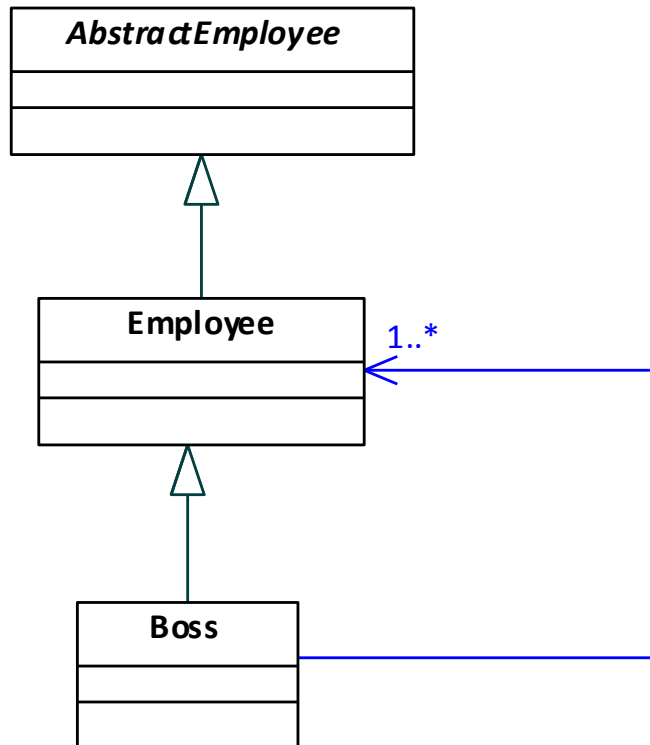
→ Solução com interface:



Composite

Exemplo 1: São definidas duas classes Empregado e Chefe.

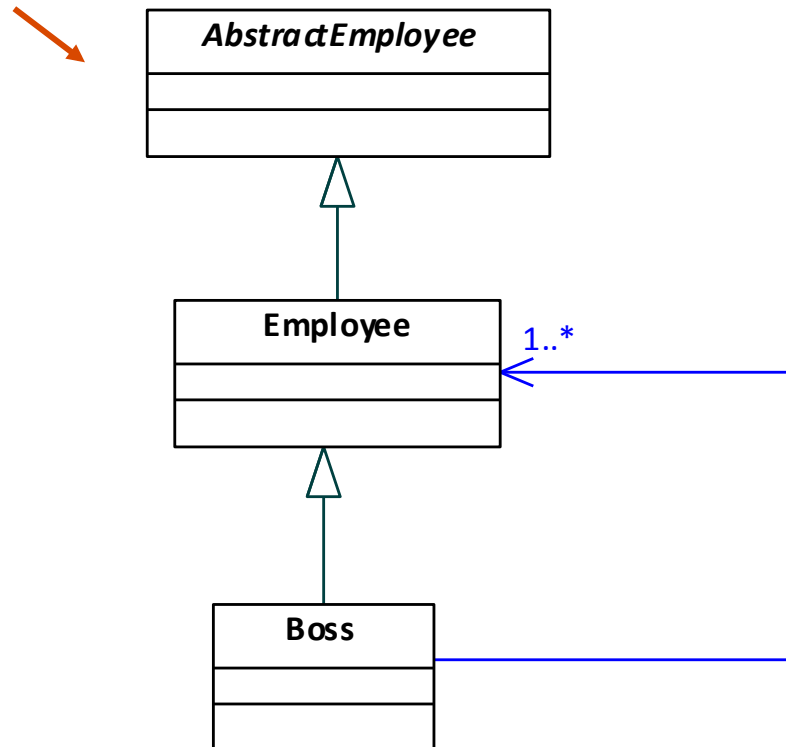
→ Solução com herança:



Composite

Exemplo 1: Classe abstrata

classe abstrata



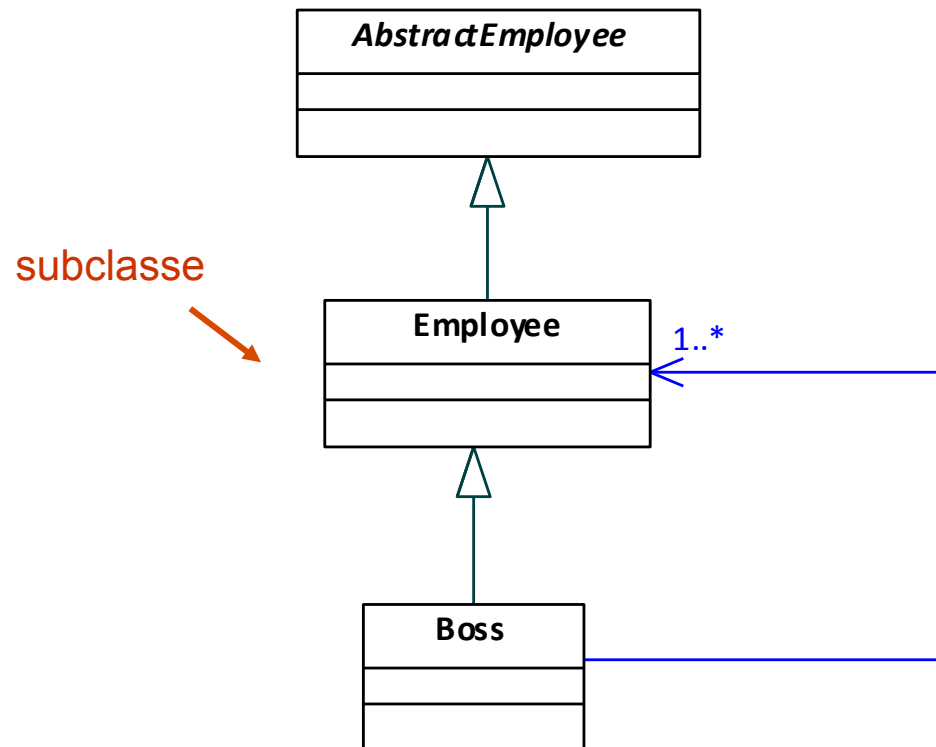
Composite

Exemplo 1: Classe abstrata

```
public abstract class AbstractEmployee {  
    protected String name;  
    protected long salary;  
    protected Employee parent = null;  
    protected boolean leaf = true;  
  
    public abstract long getSalary();  
    public abstract String getName();  
    public abstract boolean add(Employee e) throws NoSuchElementException;  
    public abstract void remove(Employee e) throws NoSuchElementException;  
    public abstract Enumeration subordinates();  
    public abstract Employee getChild(String s) throws NoSuchElementException;  
    public abstract long getSalaries(); ← retorna o custo  
    public boolean isLeaf() {  
        return leaf;  
    }  
}
```

Composite

Exemplo 1: Subclasse da classe abstrata



Composite

Exemplo 1: Classe Employee

```
public class Employee extends AbstractEmployee {  
    public Employee(String _name, long _salary) {  
        name = _name;  
        salary = _salary;  
        leaf = true;  
    }  
  
    public Employee(Employee _parent, String _name, long _salary) {  
        name = _name;  
        salary = _salary;  
        parent = _parent;  
        leaf = true;  
    }  
  
    public long getSalary() {  
        return salary;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

Composite

Exemplo 1: Classe Employee (continuação)

```
public boolean add(Employee e) throws NoSuchElementException {  
    throw new NoSuchElementException("No subordinates");  
}
```

```
public void remove(Employee e) throws NoSuchElementException {  
    throw new NoSuchElementException("No subordinates");  
}
```

```
public Enumeration subordinates () {  
    Vector v = new Vector();  
    return v.elements ();  
}
```

```
public Employee getChild(String s) throws NoSuchElementException {  
    throw new NoSuchElementException("No children");  
}
```

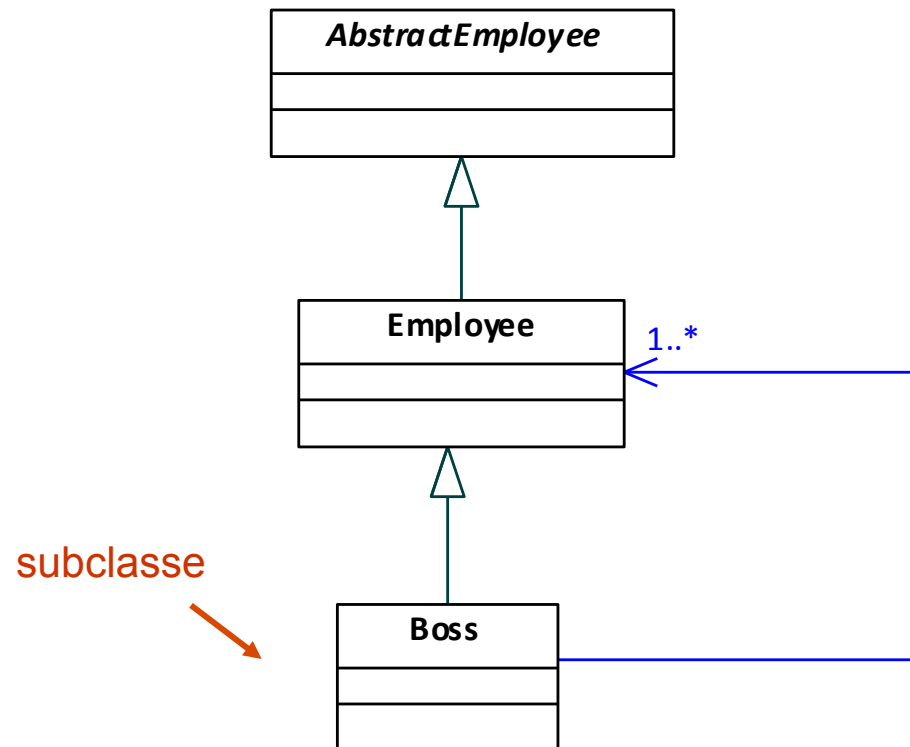
```
public long getSalaries() { ← retorna o custo  
    return salary;  
}
```

```
public Employee getParent() {  
    return parent;  
}
```

```
}
```


Composite

Exemplo 1: Subclasse da subclasse



Composite

Exemplo 1: Classe Boss

```
public class Boss extends Employee {  
    Vector employees;      ← referências aos subordinados  
  
    public Boss(String _name, long _salary) {  
        super(_name, _salary);  
        leaf = false;  
        employees = new Vector();  
    }  
  
    public Boss(Employee _parent, String _name, long _salary) {  
        super(_parent, _name, _salary);  
        leaf = false;  
        employees = new Vector();  
    }  
  
    public Boss(Employee emp) {  
        //promotes an employee position to a Boss  
        //and thus allows it to have employees  
        super(emp.getName (), emp.getSalary());  
        employees = new Vector();  
        leaf = false;  
    }  
}
```

Composite

Exemplo 1: Classe Boss (continuação)

```
public boolean add(Employee e) throws NoSuchElementException {
    employees.add(e);
    return true;
}

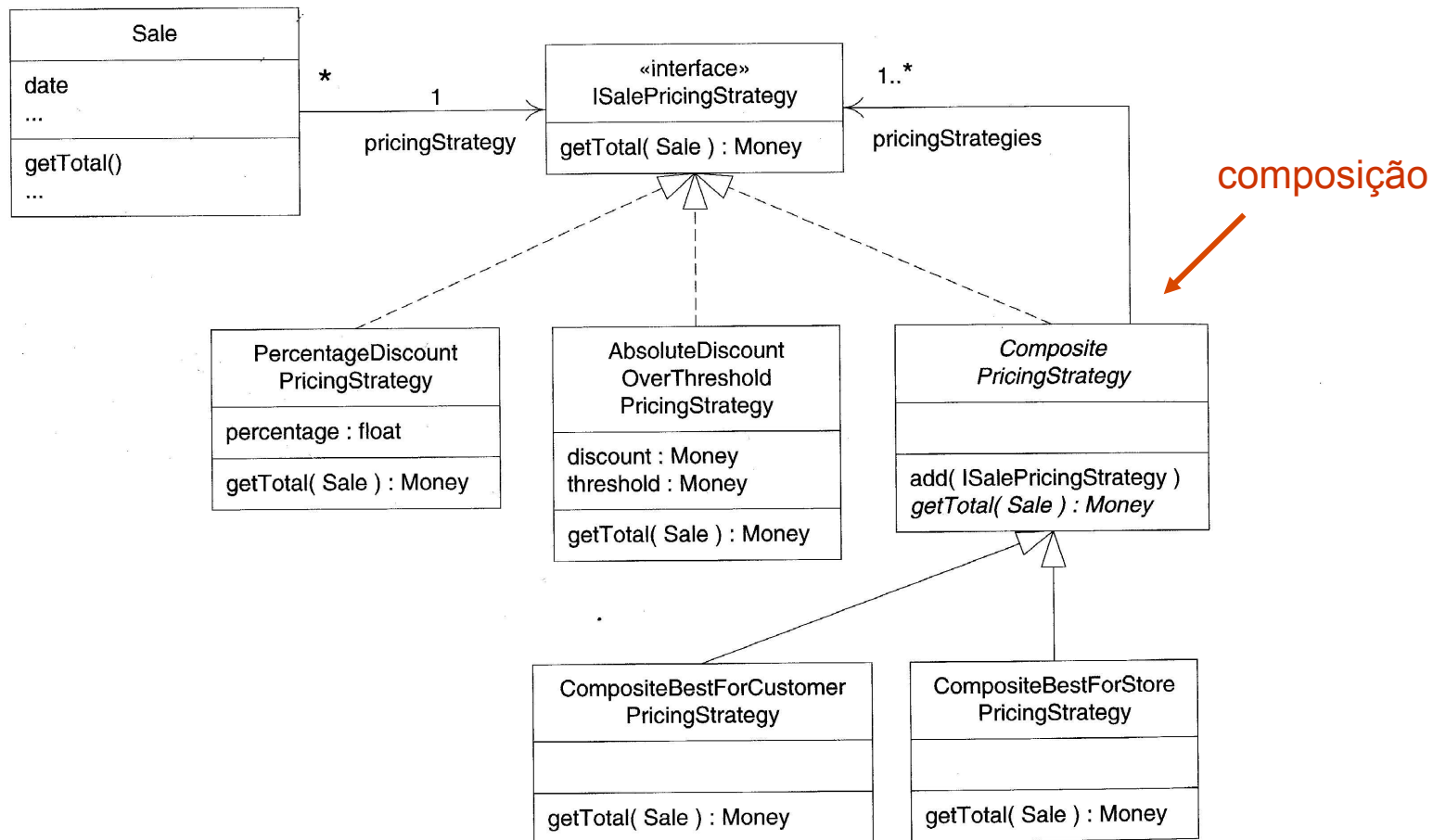
public void remove(Employee e) throws NoSuchElementException {
    employees.removeElement(e);
}

public Enumeration subordinates () {
    return employees.elements ();
}

public long getSalaries() { ← retorna o custo
    long sum = salary;
    for (int i = 0; i < employees.size(); i++) {
        sum += ((Employee)employees.elementAt(i)).getSalaries();
    }
    return sum;
}
}
```

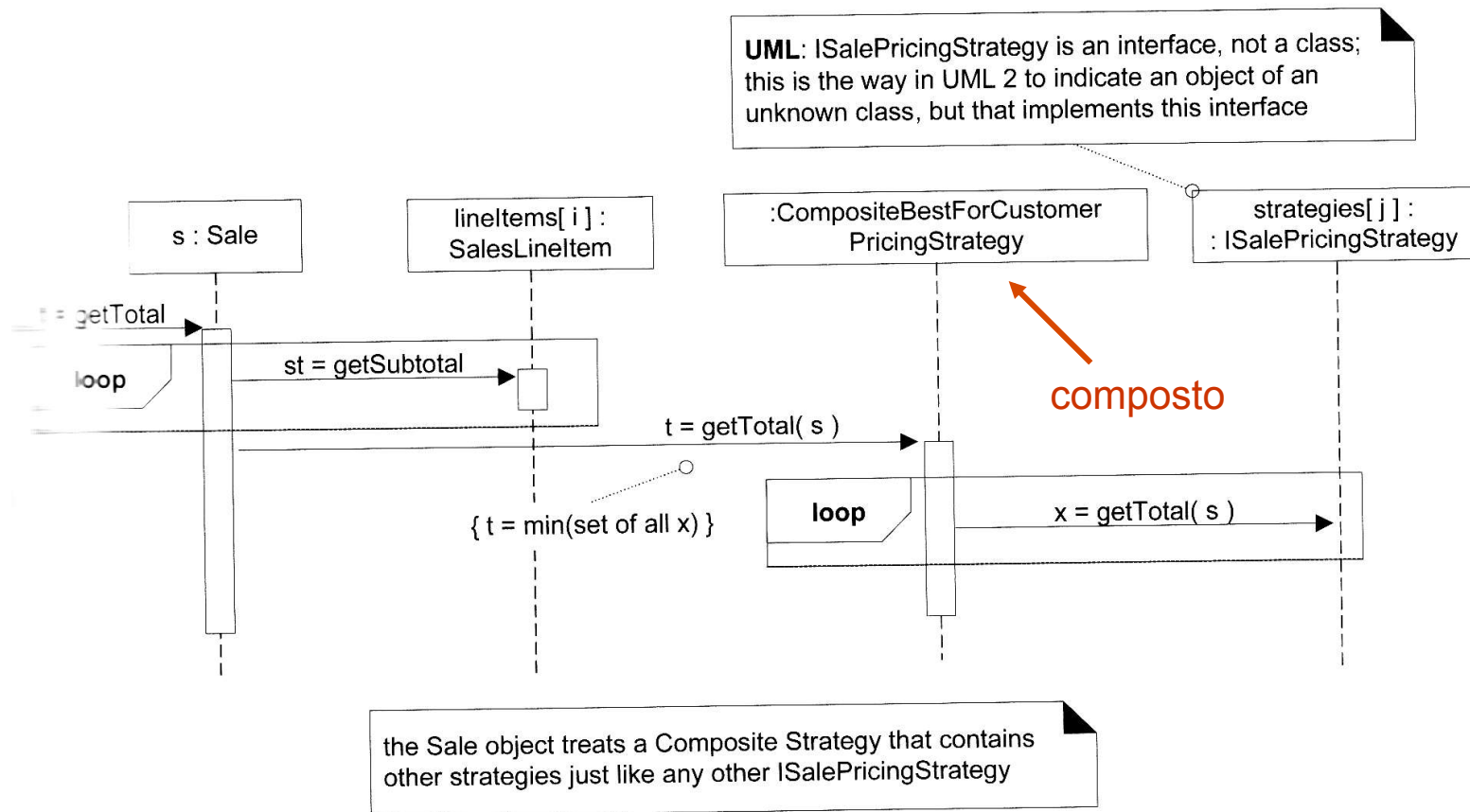
Composite

Exemplo 2 (livro do Larman):



Composite

Exemplo 2 (livro do Larman):




7. Fachada (Facade)

Problema: É necessária uma interface comum e unificada para um conjunto de implementações - como um subsistema.

Solução: Defina um ponto único de contato ao subsistema - um objeto fachada que encapsula o subsistema.

Este objeto fachada apresenta uma única interface e é responsável pela colaboração com os componentes do subsistema.

➔ Uma fachada é um objeto “front-end” que é o único ponto de entrada para os serviços de um subsistema; a implementação e outros componentes do subsistema podem ser privados e, por isso, não serem vistos pelos componentes externos.

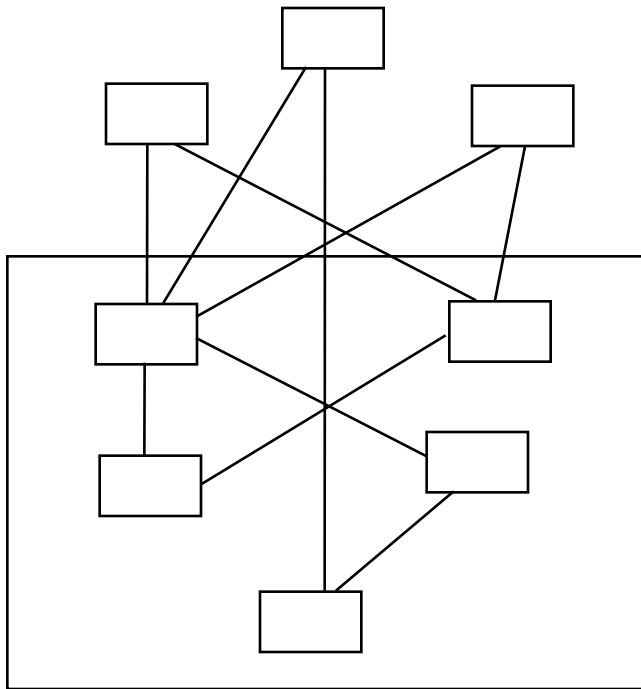


Fachada

Aplicação:

- Você quer fornecer uma interface simples para um subsistema complexo.
- Você quer dividir os seus subsistemas em camadas. Use uma fachada para definir um ponto de entrada para cada nível do subsistema.
- Existem muitas dependências entre clientes e as classes de implementação.

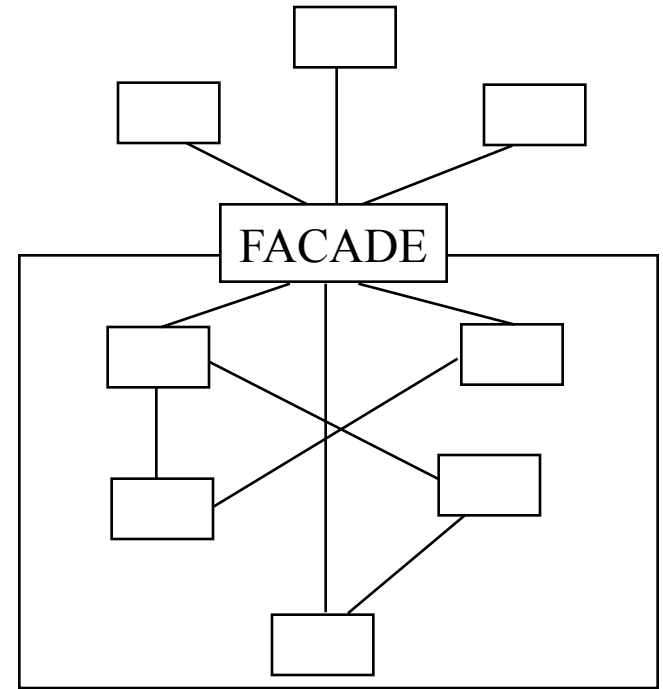
Fachada



classes cliente

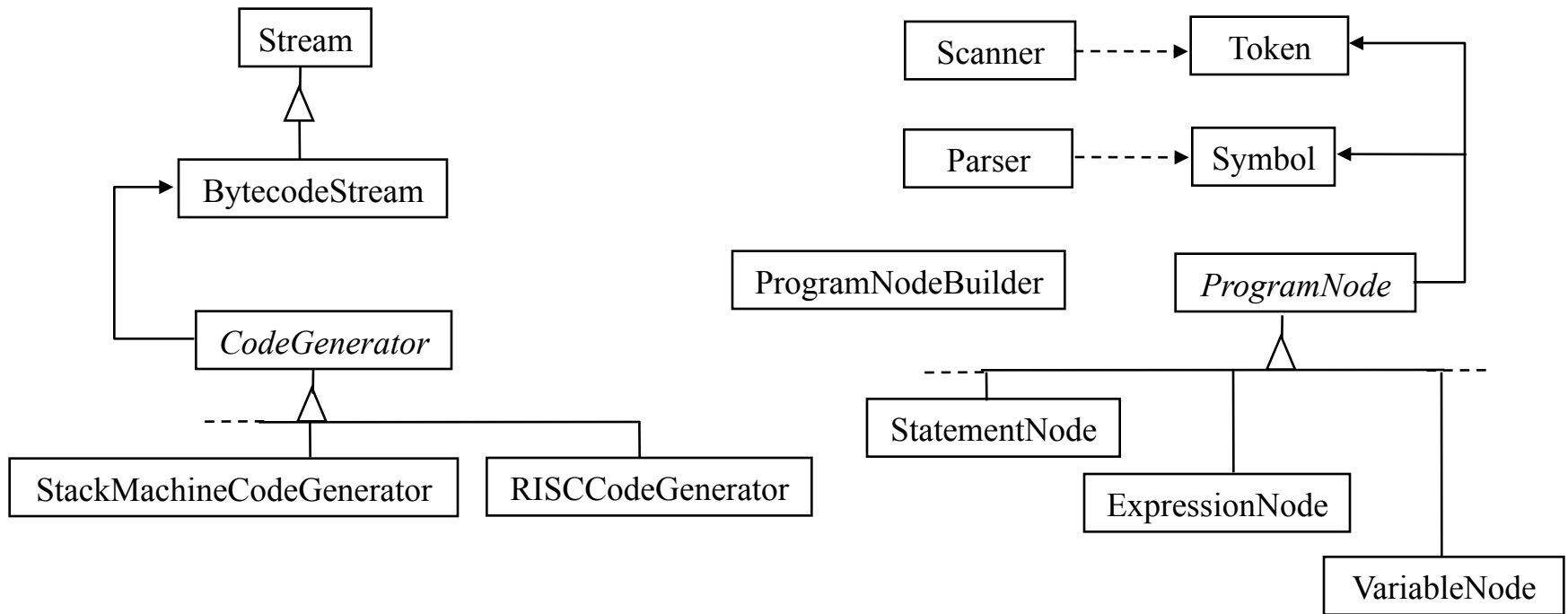


classes do subsistema



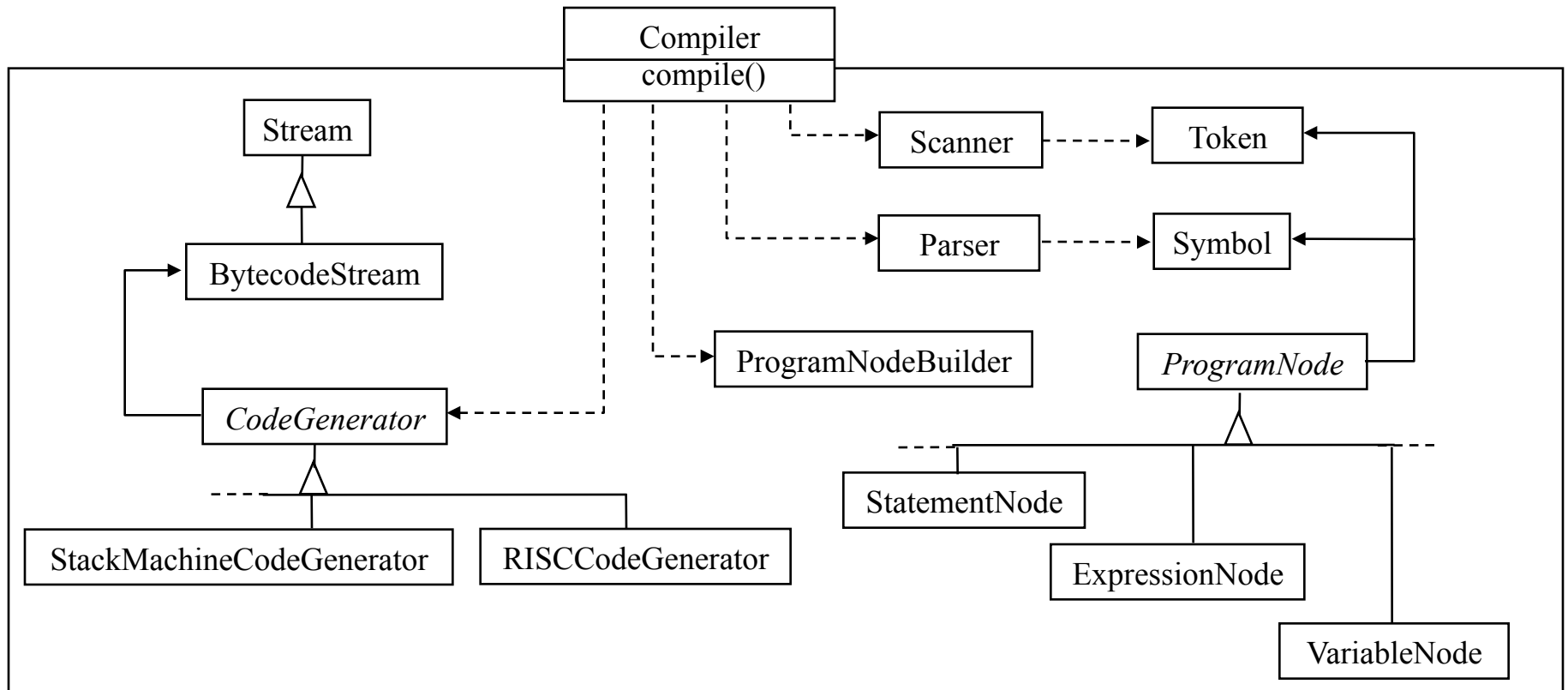
Fachada

Exemplo: Conjunto de classes Java que implementam um compilador: Scanner, Parser, ProgramNode, BytecodeStream, ProgramNodeBuilder.



Fachada

Exemplo: Para fornecer uma interface de mais alto nível que pode esconder estas classes do cliente, o compilador inclui uma classe **Compilador**, que age como uma fachada.



Fachada

Exemplo: Classe Compiler

```
public class Compiler {  
  
    public Compiler(){} //construtor  
  
    public void compile (InputStream input, BytecodeStream output){  
        Scanner scanner = new Scanner(input);  
        ProgramNodeBuilder builder = new ProgramNodeBuilder();  
        Parser parser = new Parser();  
        parser.parse (scanner, builder);  
        RISCCodeGenerator generator = new RISCCodeGenerator(output);  
        ProgramNode parseTree = builder.getRootNode();  
        parseTree.traverse(generator);  
    }  
}
```

8. Observador

Problema: Diferentes tipos de objetos observadores (ouvintes) estão interessados nas mudanças de um objeto observado (emissor) e querem reagir de uma maneira única quando o observado gera um evento. Mas o observado quer manter um baixo acoplamento com os observadores.

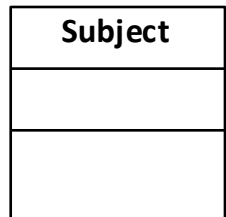
Solução: Defina uma interface “observador”. Os observadores devem implementar esta interface. O observado pode registrar dinamicamente os observadores que estão interessados em um evento e notificá-los quando um evento ocorre.

Observador = Observer = Subscriber = Listener

Observado = Subject = Publisher = Observable

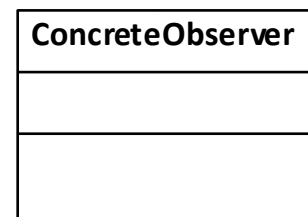
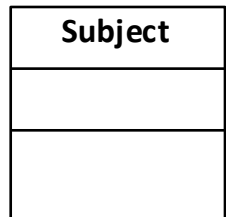
Observador

- Observado (Subject): objeto com os dados.



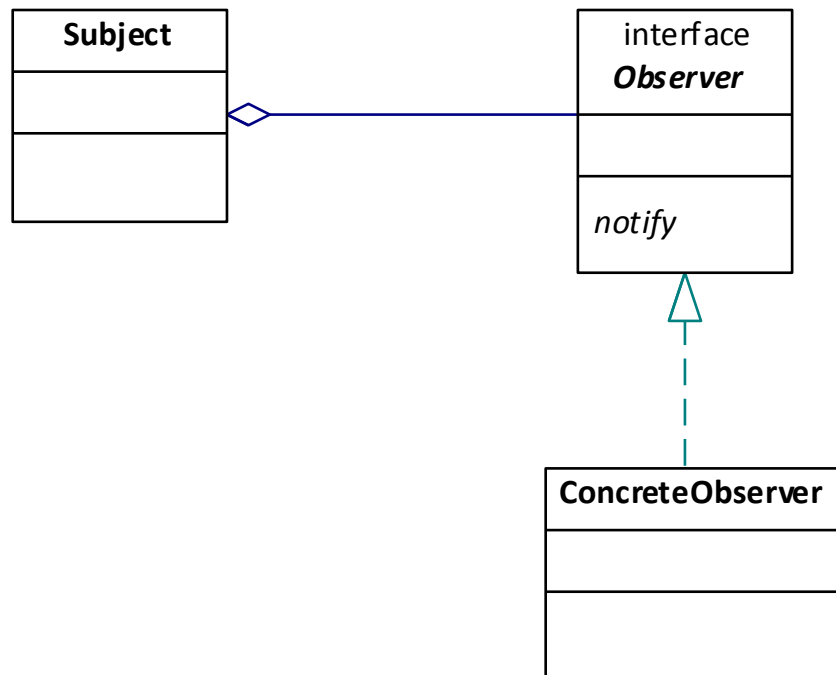
Observador

- Observado (Subject): objeto com os dados.
- Observador (Observer): objeto interessado nas mudanças sobre os dados.



Observador

- Observador (Observer): objeto interessado nas mudanças sobre os dados.
- Cada Observer registra seu interesse no dado, chamando um método público do Subject.
- Cada Observer tem uma interface conhecida que o Subject chama quando o dado muda.





Observador

```
public interface Subject {  
    // notifica o Subject que você está interessado em mudanças  
    public void registerInterest(Observer obs);  
}
```

```
public interface Observer {  
    // notifica os observadores que uma mudança ocorreu  
    public void sendNotify(String s);  
}
```




Observador

Aplicação:

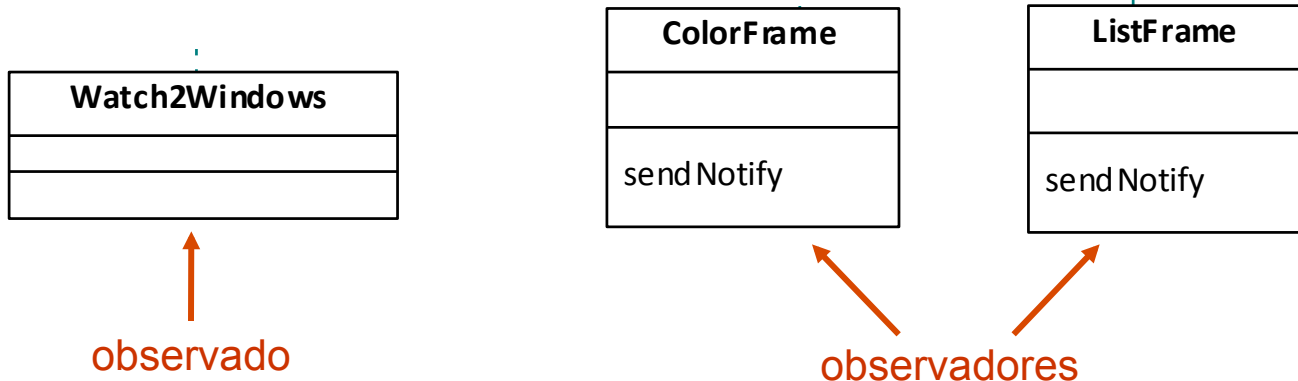
- Quando um objeto deve ser capaz de notificar outros objetos sem precisar conhecer quem são estes objetos.

Exemplo:

- os preços de ações de mercado devem ser representados como um grafo ou como uma lista, e sempre que eles mudam, essas representações também devem mudar.

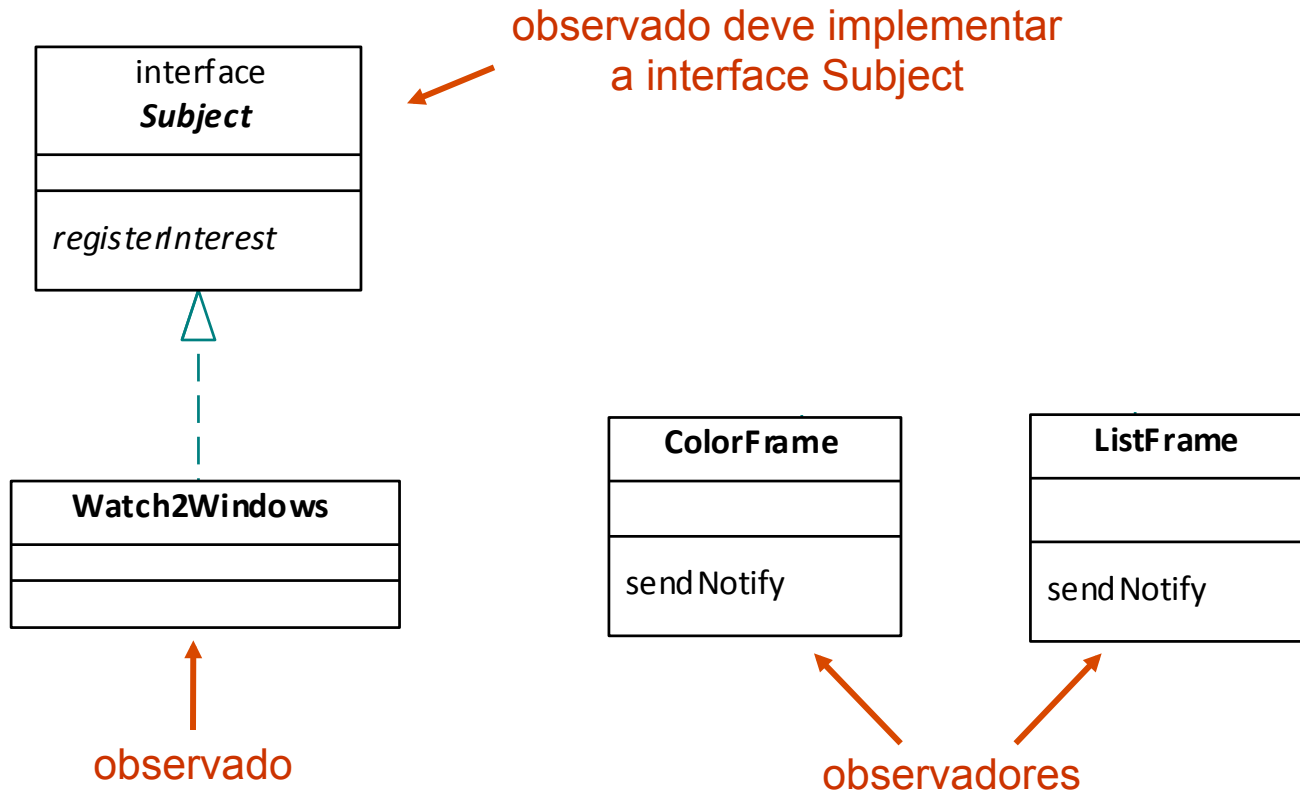
Observador

Exemplo 1: Uma janela contém 3 botões: Red, Green, Blue. Quando um botão é selecionado, duas outras janelas precisam ser notificadas: uma para mudar a sua cor e a outra para colocar o nome da cor selecionada em uma lista.



Observador

Exemplo 1: Uma janela contém 3 botões: Red, Green, Blue. Quando um botão é selecionado, duas outras janelas precisam ser notificadas: uma para mudar a sua cor e a outra para colocar o nome da cor selecionada em uma lista.



Observador

Exemplo 1: Classe Watch2Windows

```
public class Watch2Windows extends JFrame
implements ActionListener, ItemListener, Subject {
    private JButton Close;
    private JRadioButton red, green, blue;
    private Vector observers;

    public Watch2Windows() {
        super("Change 2 other frames");
        observers = new Vector();           //lista dos observadores
        JPanel p = new JPanel(true);
        p.setLayout(new BorderLayout());
        getContentPane().add("Center", p);
        ...
    }
```

Observador

Exemplo 1: Classe Watch2Windows (continuação)

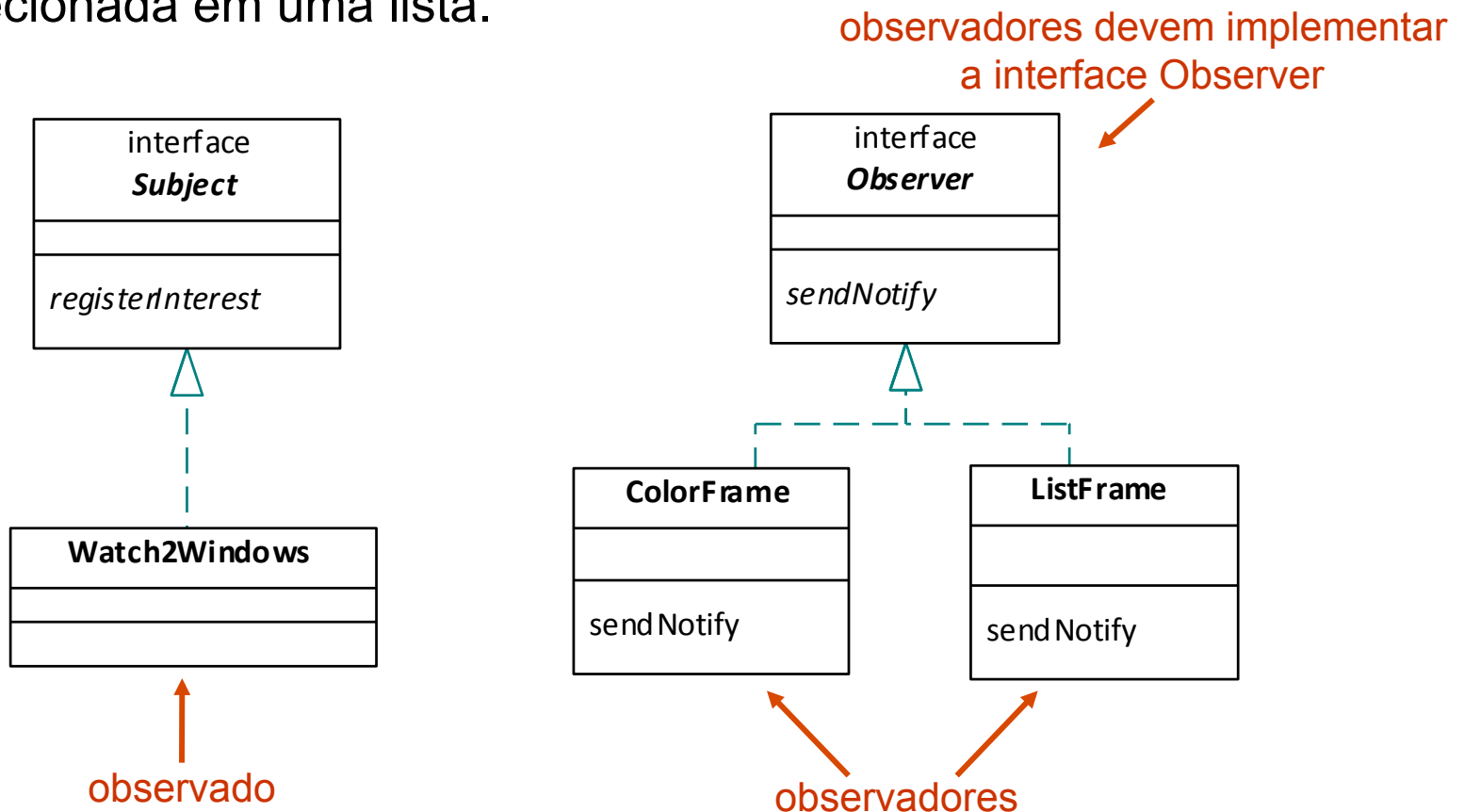
```
public void itemStateChanged(ItemEvent e) {
    if (e.getStateChange() == ItemEvent.SELECTED)
        notifyObservers((JRadioButton)e.getSource());
}

private void notifyObservers(JRadioButton rad) {
    String color = rad.getText();
    for (int i=0; i< observers.size(); i++) {
        ((Observer) (observers.elementAt(i))).sendNotify(color);
    }
}

public void registerInterest(Observer obs) {
    observers.addElement(obs);
}
}
```

Observador

Exemplo 1: Uma janela contém 3 botões: Red, Green, Blue. Quando um botão é selecionado, duas outras janelas precisam ser notificadas: uma para mudar a sua cor e a outra para colocar o nome da cor selecionada em uma lista.



Observador

Exemplo 1: Classe ListFrame

```
public class ListFrame extends JFrame implements Observer {  
    private JList list;  
    private JPanel p;  
    private JScrollPane lsp;  
    private JListData listData;  
  
    public ListFrame(Subject s) {  
        super("Color List");  
        p = new JPanel(true);  
        getContentPane().add("Center", p);  
        p.setLayout(new BorderLayout());  
        s.registerInterest(this);  
        ...  
    }  
  
    public void sendNotify String s) {  
        listData.addElement(s);  
    }  
}
```

Observador

Exemplo 1: Classe ColorFrame

```
public class ColorFrame extends JFrame implements Observer {  
    private Color color;  
    private String color_name="black";  
    private Font font;  
    private JPanel p = new JPanel(true);  
  
    public ColorFrame(Subject s) {  
        super("Colors");  
        getContentPane().add("Center", p);  
        s.registerInterest(this);  
        setBounds(100, 100, 100, 100);  
        font = new Font("Sans Serif", Font.BOLD, 14);  
        setVisible(true);  
    }  
}
```


Observador

Exemplo 1: Classe ColorFrame (continuação)

```
public void sendNotify (String s) {  
    color_name = s;  
    if (s.equalsIgnoreCase ("RED"))  
        color = Color.red;  
    if (s.equalsIgnoreCase ("BLUE"))  
        color =Color.blue;  
    if (s.equalsIgnoreCase ("GREEN"))  
        color = Color.green;  
    //p.repaint();  
    setBackground(color);  
}  
}
```

Observador

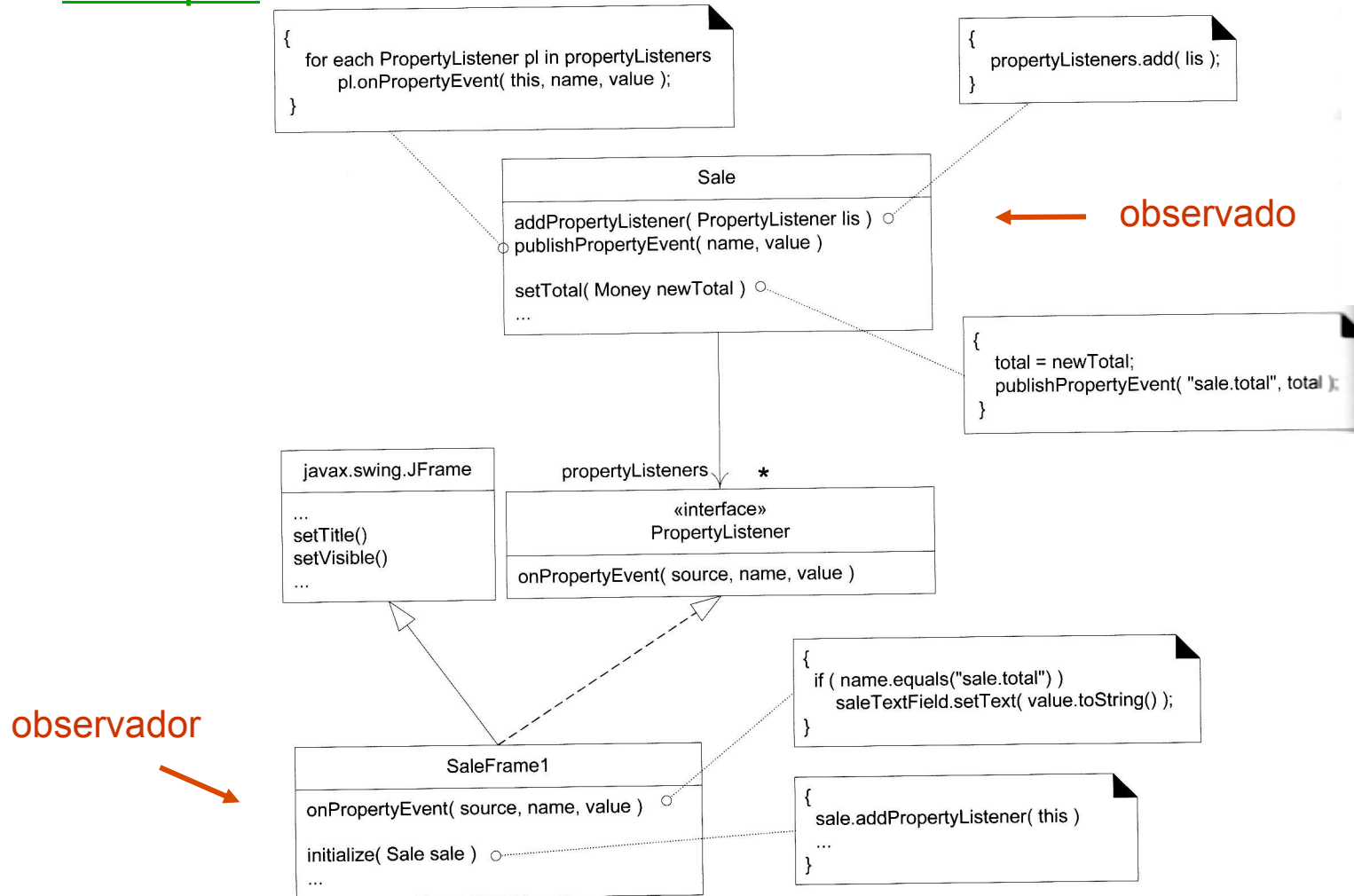
Exemplo 1:

Em um programa...

```
Subject wwind = new Watch2Windows();  
Observer cframe = new ColorFrame(wwind);  
Observer lframe = new ListFrame(wwind);
```

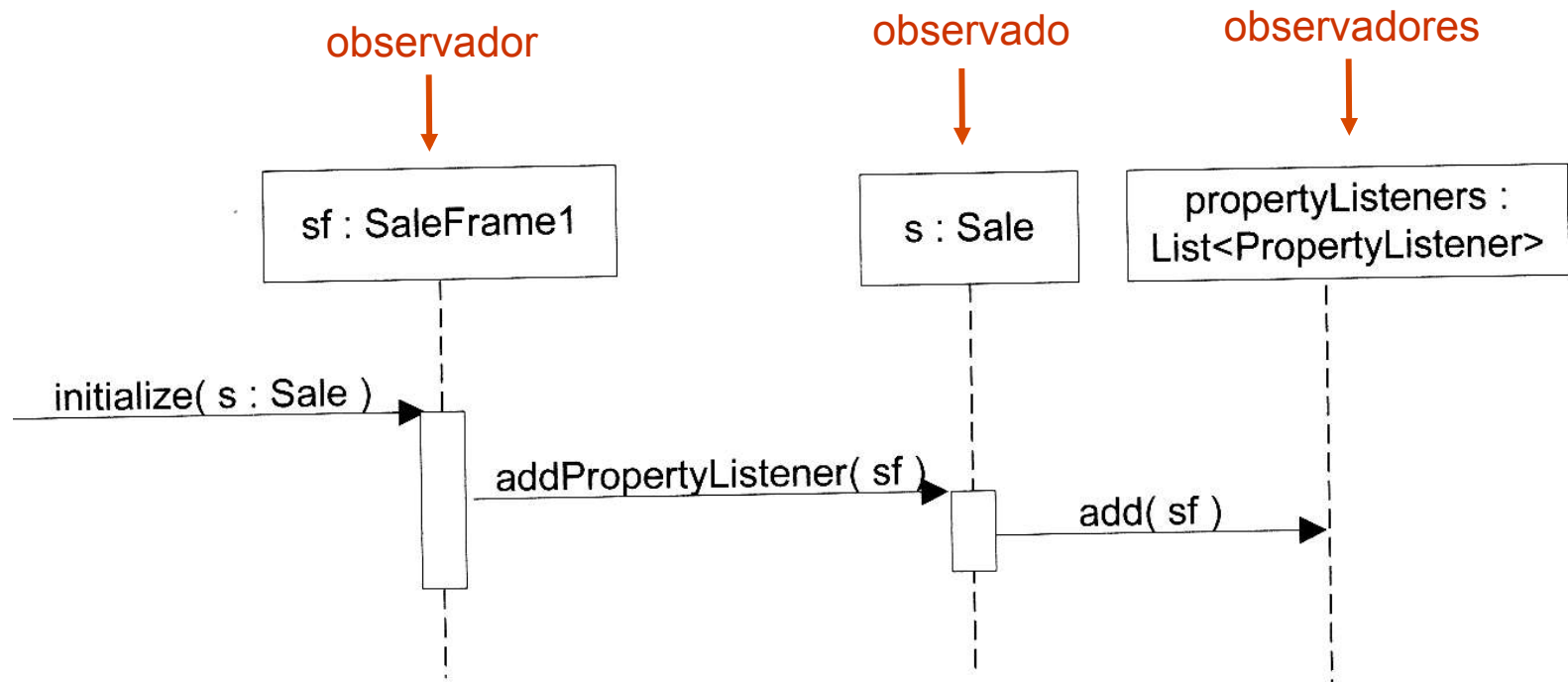
Observador

Exemplo 2 (livro do Larman):



Observador

Exemplo: observador se registrando no observado



Observador

Exemplo: observado notificando os observadores

