

INSTITUTO FEDERAL
ESPIRITO SANTO

POO2: Padrões Estruturais



INSTITUTO FEDERAL
ESPIRITO SANTO

Padrões Estruturais

- Flyweight: utiliza um objeto para economizar o máximo de memória e compartilhar o máximo de dados entre objetos;
- Adapter: converte uma interface de um terceiro interface que o programa entenda;
- Decorator: adiciona responsabilidades à um objeto dinamicamente;
- Facade/Fachada: prover uma interface comum de acesso a diversos subsistemas;
- Composite: realiza a composição de objetos em uma estrutura de árvore, formando uma entidade todo-parte;

PADRÕES DE PROJETO

- **Flyweight:**

- **Propósito:** Usar o compartilhamento para suportar de forma eficiente grandes quantidades de objetos;
 - Alguns programas precisam utilizar/criar uma grande quantidade de objetos de um tipo particular que podem consumir rapidamente a memória;
 - Um grande número de objetos podem ser substituídos por um grupo relativamente poucos objetos compartilhados;
 - Uma característica muito importante do Flyweight é que os objetos representados por ele precisam ser imutáveis. Em outras palavras, depois de criados, o estado interno desses objetos não pode ser mais alterado.

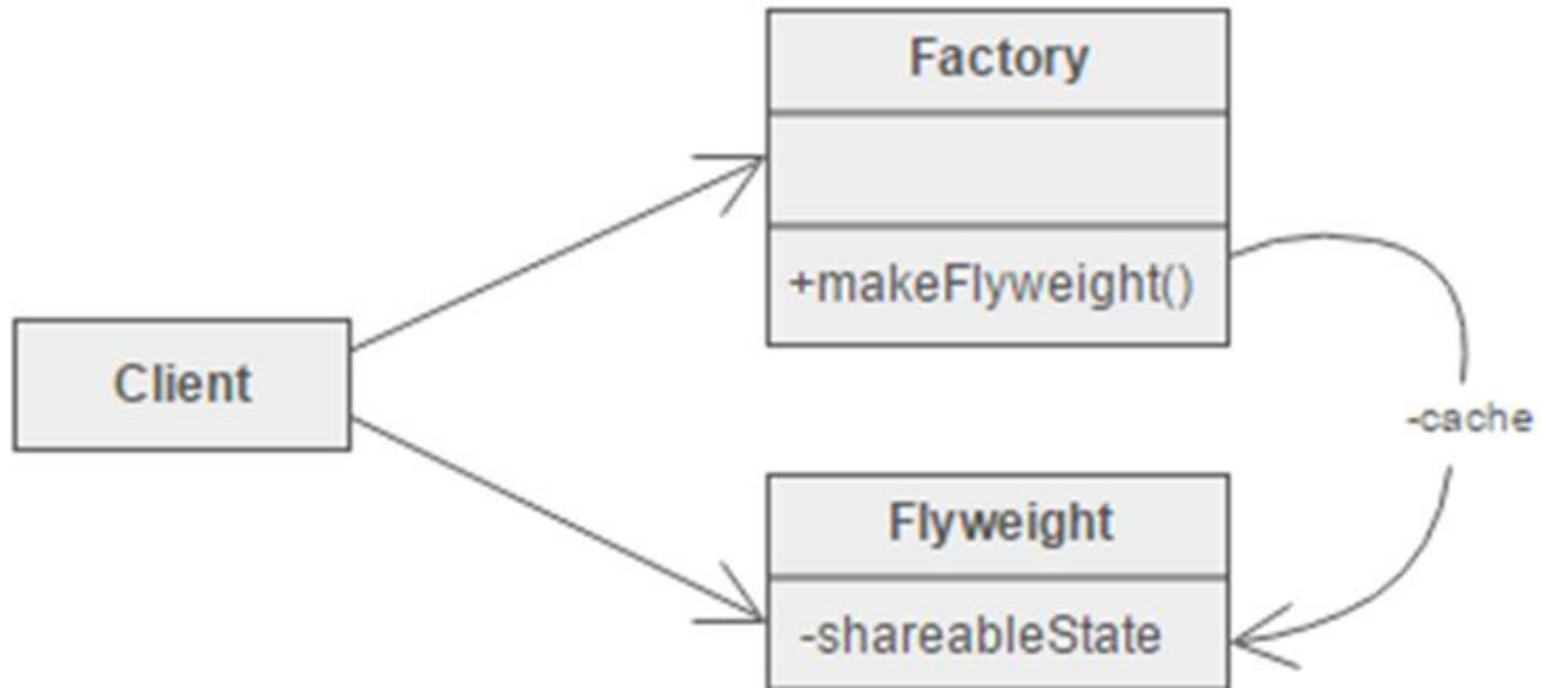
PADRÕES ESTRUTURAIS

FlyWeight – Peso Mosca

PADRÕES DE PROJETO

The *Flyweight* pattern allows you to reference a multitude of objects of the same type and having the same state, but only by instantiating the minimum number of actual objects needed. This is typically done by allocating a 'pool' of objects which can be shared, and this is determined by a 'flyweight factory' class.

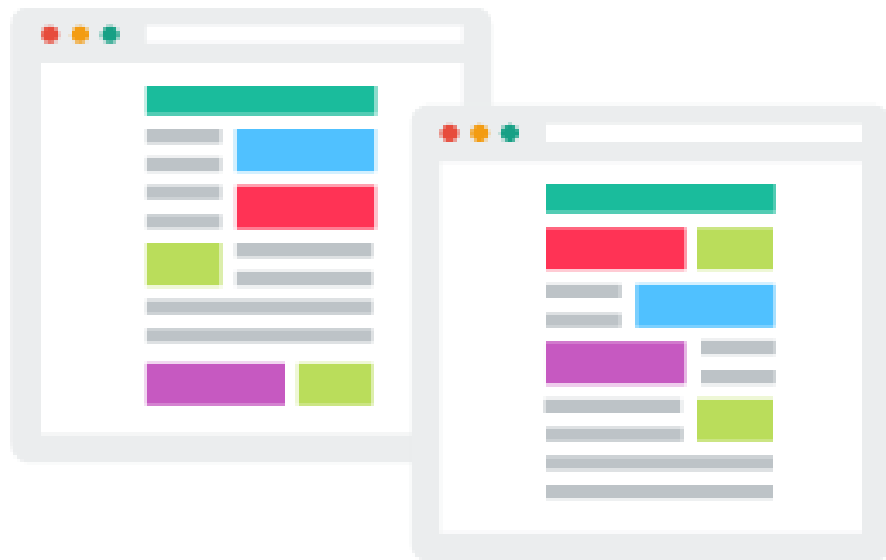
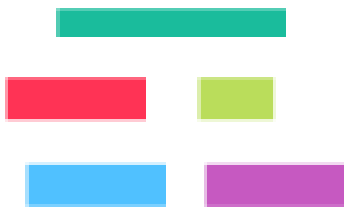
Flyweight



Flyweight

The Flyweight uses sharing to support large numbers of objects efficiently. Modern web browsers use this technique to prevent loading same images twice. When browser loads a web page, it traverse through all images on that page. Browser loads all new images from Internet and places them the internal cache. For already loaded images, a flyweight object is created, which has some unique data like position within the page, but everything else is referenced to the cached one.

Browser loads images
just once and then
reuses them from pool:



Ferramenta de Desenvolvedor do Chrome

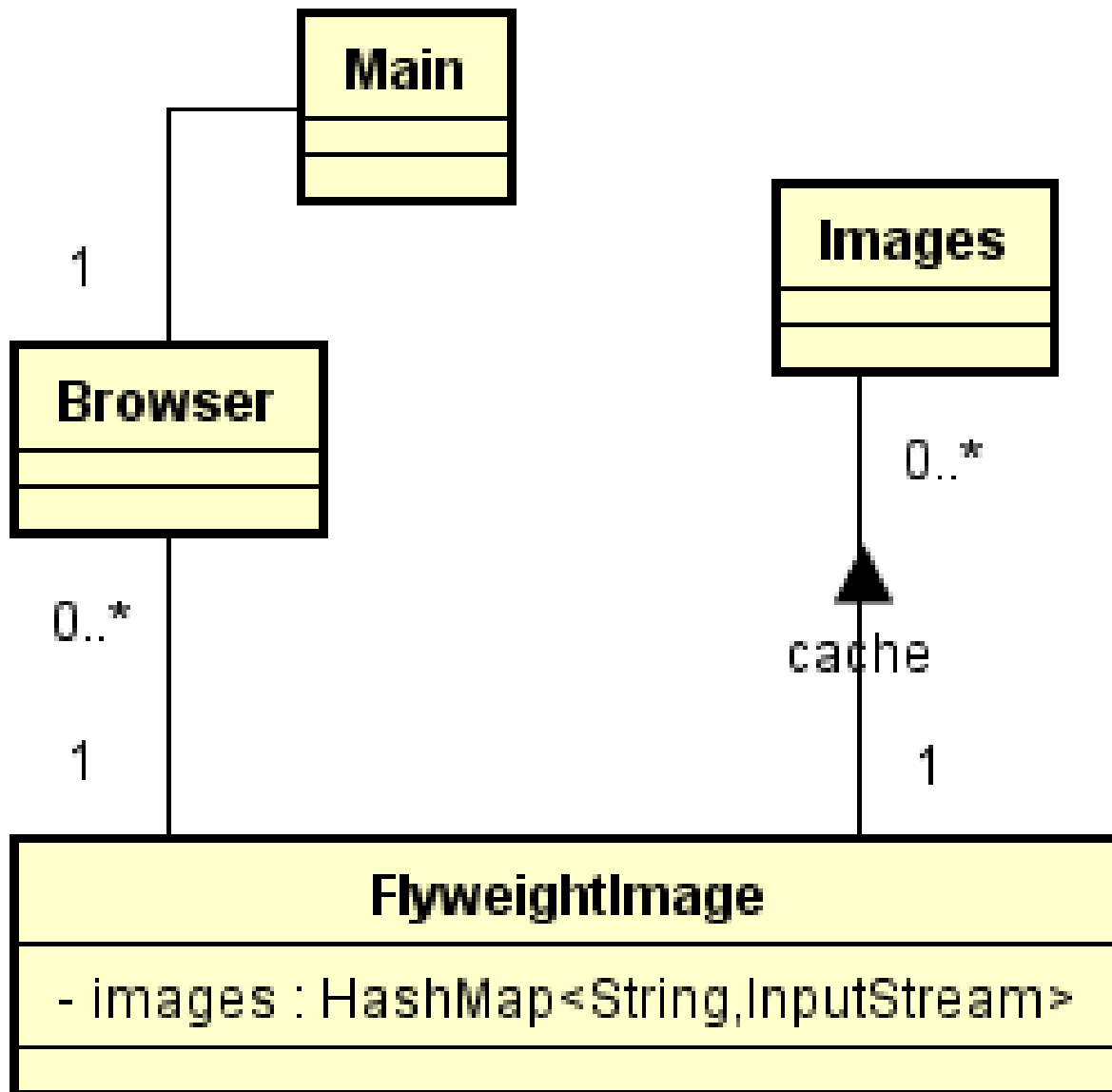
The screenshot shows the Google homepage with the Chrome DevTools Network tab open. The page URL is https://www.google.com.br/?gfe_rd=cr&ei=QK7qV7S8EOfM8AfNjI3gAw. The Network tab displays a list of resources loaded on the page, including images, scripts, and JSON files. The timeline at the top of the Network tab shows the loading sequence of these resources.

Name	Status	Type	Initiator	Size	Time
data:image/gif;base...	200	gif	?gfe_rd=cr...	(from cache)	0
nav_logo242.png	200	png	Other	(from cache)	8
rs=ACT90oEeQrviDGWcelA9GAoWTSwIweXQjw	200	script	?gfe_rd=cr...	(from cache)	15
rs=AA2YrTtjxNZ90Qk70z807QYMONHzAWeDEQ	200	script	?gfe_rd=cr...	(from cache)	13
SupportLanguageList.json	200	xhr	jQuery-1.7...	(from cache)	5
dn/	200	docu...	rs=AA2YrTtj...	(from cache)	3
cb=gapi.loaded_0	200	script	rs=AA2YrTtj...	(from cache)	17
data:image/gif;base...	200	gif	rs=ACT90o...	(from cache)	1
data:image/png;base...	200	png	rs=ACT90o...	(from cache)	1
data:image/gif;base...	200	gif	rs=ACT90o...	(from cache)	0
rs=ACT90oEeQrviDGWcelA9GAoWTSwIweXQjw	200	script	rs=ACT90o...	(from cache)	14
gen_204?v=3&s=webhp&atyp=csi&ei=QK7qV_qIHL...	204	text/...	Other	40 B	192
tia.png	200	png	Other	(from cache)	10

Name	S...	T...	In...	Size	Time	Timelin
dn/	2...	d...	rs...	(from cache)	3 ms	
cb=gapi.loaded_0	2...	s...	rs...	(from cache)	17 ms	
data:image/gif;base...	2...	gif	rs...	(from cache)	1 ms	
data:image/png;base...	2...	p...	rs...	(from cache)	1 ms	
data:image/gif;base...	2...	gif	rs...	(from cache)	0 ms	
rs=ACT90oEeQrviDGWcelA9GAoWTSwl...	2...	s...	rs...	(from cache)	14 ms	
gen_204?v=3&cs=webhp&atyp=csi&ei...	2...	t...	O...	40 B	192 ms	
tia.p	2...	p...	O...	(from cache)	10 ms	
gcoss	2...	x...	rs...	826 B	913 ms	
search	2...	x...	rs...	180 B	196 ms	
dn.js	2...	s...	(l...	(from cache)	4 ms	
frame?sourceid=1&hl=pt-BR&origin=h...	2...	d...	c...	(from cache)	10 ms	
rs=AGLTcCP6GkOjxhPrYd37cLbTo-KVid...	2...	s...	fr...	(from cache)	11 ms	

Imagens sendo
carregadas do cache

Código java - FlyWeight Browser Images



```

public class Browser {

    FlyweightImage flyWeightImage = new FlyweightImage();

    public void getAllImagesFrom(String url ) throws ParseException,
        MalformedURLException, IOException{
        Parser parser = new Parser(url);
        NodeList list = parser.parse(new TagNameFilter("IMG"));
        for ( SimpleNodeIterator iterator = list.elements(); iterator.hasMoreNodes(); )
        {
            Tag tag = (Tag) iterator.nextNode();
            // System.out.println(tag.getAttribute("src"));
            flyWeightImage.addImage(tag.getAttribute("src"));
        }
    }

    public InputStream getImage(String urlImage){
        return flyWeightImage.getImage(urlImage);
    }

}

```

```

public class FlyweightImage {
    HashMap<String,InputStream> image = new HashMap();
    public void addImage(String url) throws MalformedURLException{
        InputStream image = null;
        if(!this.image.containsKey(url)) {
            try {
                image = new URL(url).openStream();

            } catch (IOException ex) {
                Logger.getLogger(FlyweightImage.class.getName()).log(I
            }
            this.image.put(url, image);
        }
    }
}

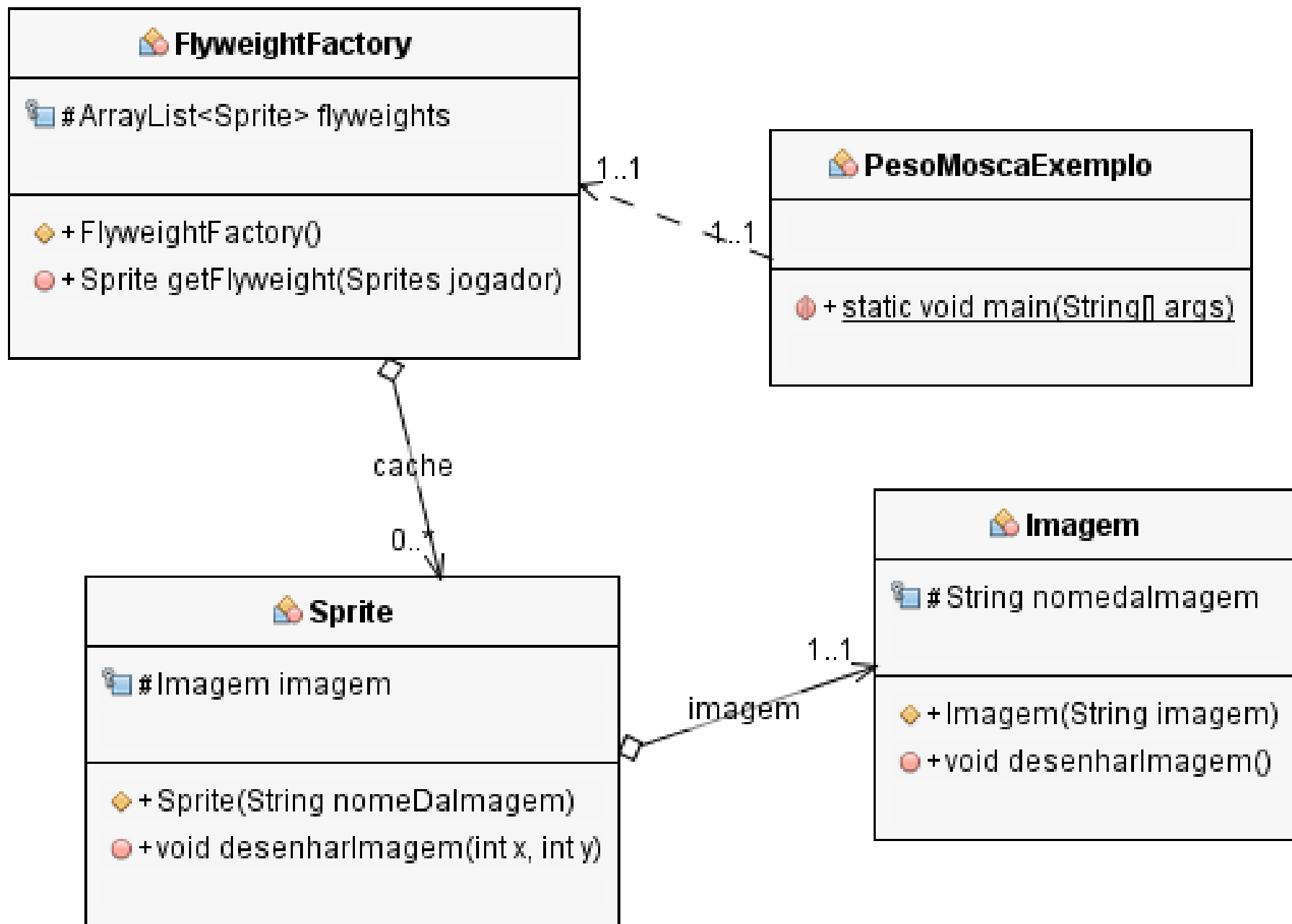
```

```
public class Main {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) throws Exception {  
  
        Browser browser = new Browser();  
        browser.getAllImagesFrom("http://www.yahoo.com/");  
  
        InputStream image = browser.getImage("image_url");  
  
    }  
}
```

Cache de imagens em jogos

Sprite - é um objeto gráfico bi ou tridimensional que se move numa tela sem deixar traços de sua passagem (como se fosse um "espírito")

Exemplo sprite




```
public class Imagem {  
    protected String nomedaImagem;  
  
    public Imagem(String imagem) {  
        nomedaImagem = imagem;  
    }  
  
    public void desenharImagem() {  
        System.out.println(nomedaImagem + " desenhada!");  
    }  
}
```

```
public class Sprite {  
    protected Imagem imagem;  
  
    public Sprite(String nomeDaImagem) {  
        imagem = new Imagem(nomeDaImagem);  
    }  
  
    public void desenharImagem(int x, int y) {  
        imagem.desenharImagem();  
        System.out.println("No ponto (" + x + ", " + y + ")!");  
    }  
}
```

```

public class FlyweightFactory {
    protected ArrayList<Sprite> flyweights;
    public enum Sprites {
        JOGADOR, INIMIGO_1, INIMIGO_2, CENARIO_1
    }
    public FlyweightFactory() {
        flyweights = new ArrayList<Sprite>();
        flyweights.add(new Sprite("jogador.png"));
        flyweights.add(new Sprite("inimigo1.png"));
        flyweights.add(new Sprite("inimigo2.png"));
        flyweights.add(new Sprite("cenario1.png"));
    }
    public Sprite getFlyweight(Sprites jogador) {
        switch (jogador) {
            case JOGADOR:
                return flyweights.get(0);
            case INIMIGO_1:
                return flyweights.get(1);
            case INIMIGO_2:
                return flyweights.get(2);
            default:
                return flyweights.get(3);
        }
    }
}

```



Usando....

```
import pesomoscaexemplo.FlyweightFactory.Sprites;

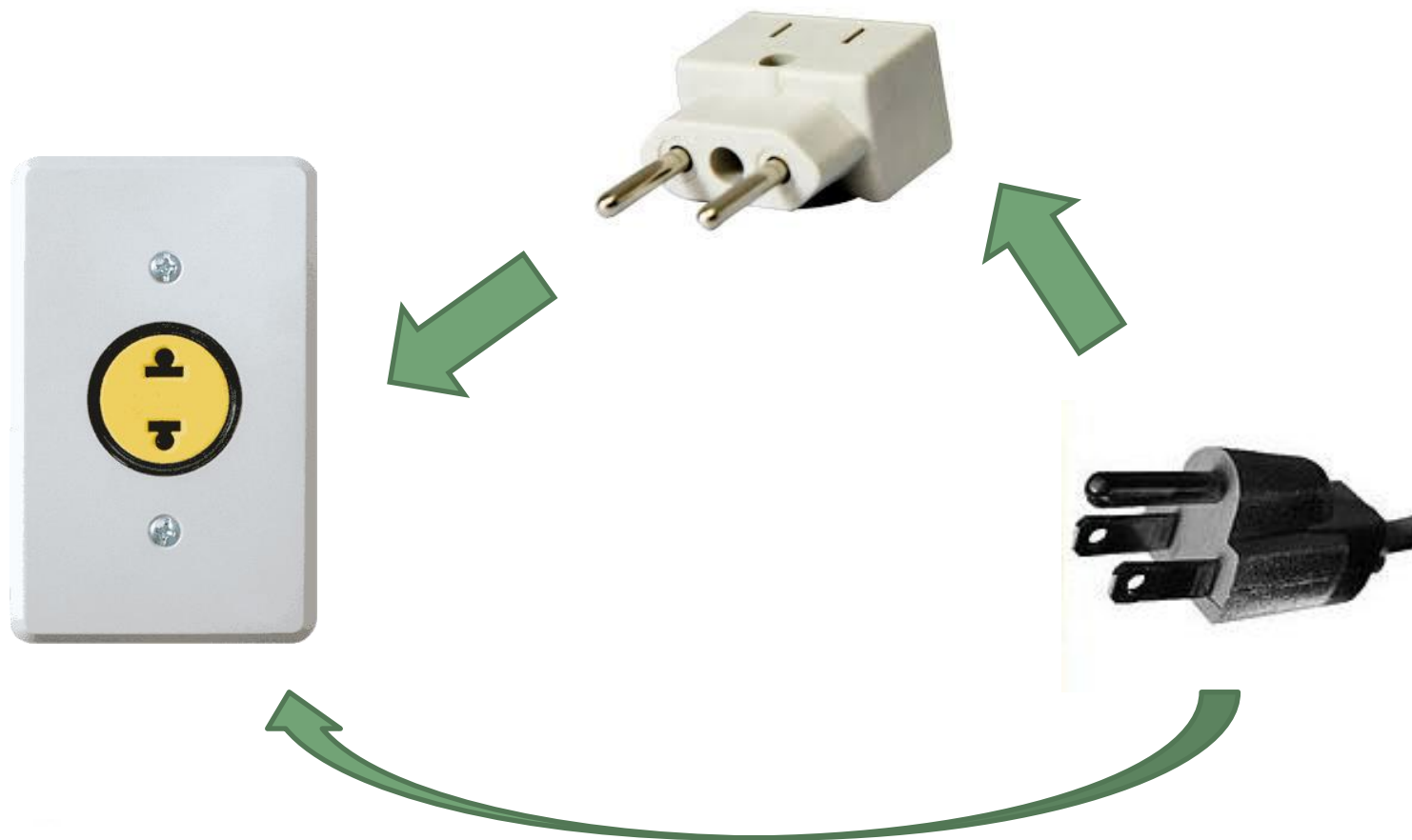
public class PesoMoscaExemplo {

    public static void main(String[] args) {
        FlyweightFactory factory = new FlyweightFactory();
        factory.getFlyweight(Sprites.JOGADOR).desenharImagem(5,5);
        factory.getFlyweight(Sprites.INIMIGO_1).desenharImagem(10,10);
        factory.getFlyweight(Sprites.INIMIGO_1).desenharImagem(20,20);
        factory.getFlyweight(Sprites.INIMIGO_1).desenharImagem(30,10);
        factory.getFlyweight(Sprites.INIMIGO_2).desenharImagem(40,40);
        factory.getFlyweight(Sprites.INIMIGO_2).desenharImagem(20,35);
    }
}
```

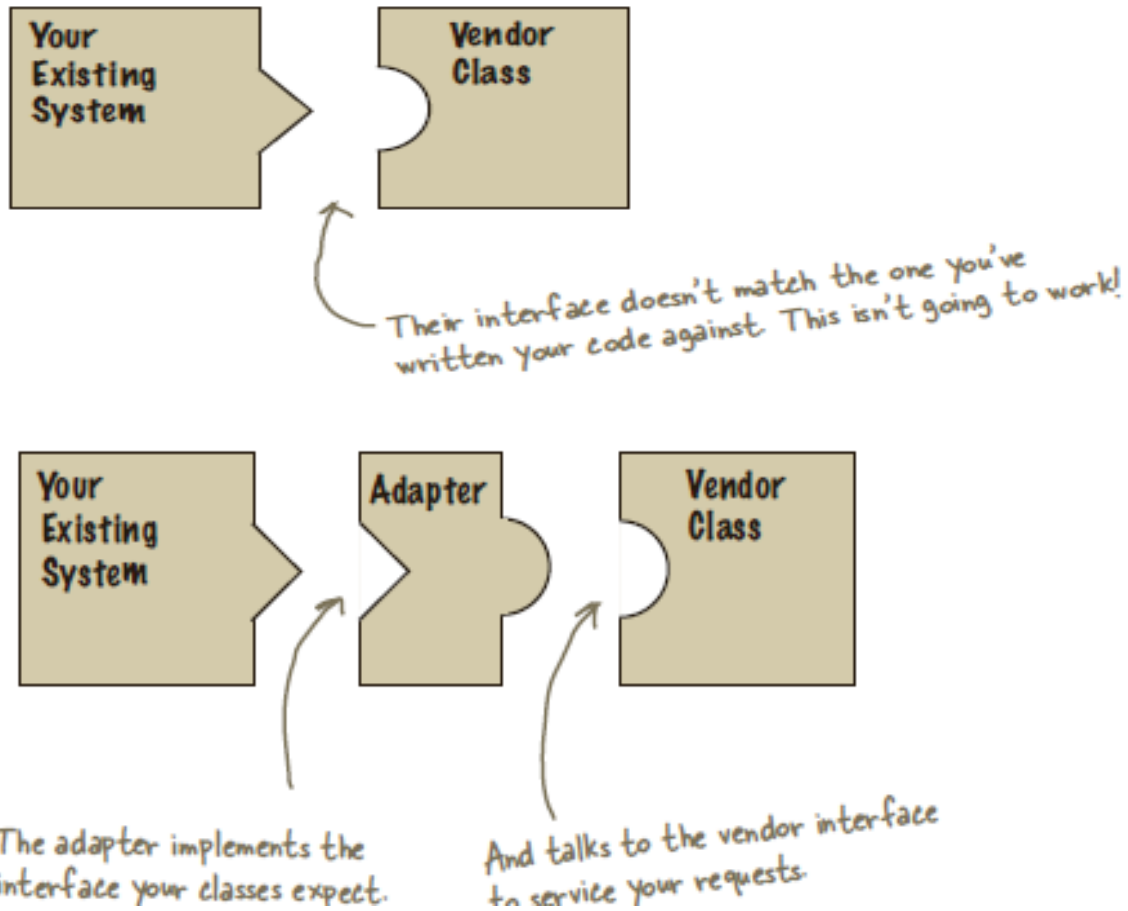
PADRÕES ESTRUTURAIS

Adapter

Padrão Adapter



Adapter



PADRÕES DE PROJETO

- Adapter:
 - **Propósito:** converte uma interface de um terceiro em uma interface que o programa entende;
 - **Problema:**
Não temos acesso ao código fonte da classe que queremos utilizar.

PADRÕES DE PROJETO

Adapter: Exemplo genérico de Adapter

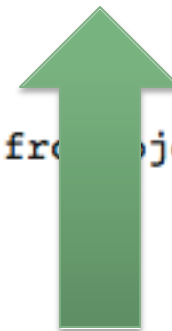
```
public class ObjectAdapter extends ClassAdaptingTo {  
    private ClassAdaptingFrom fromObject;  
  
    public ObjectAdapter(ClassAdaptingFrom fromObject) {  
        this.fromObject = fromObject;  
    }  
  
    // Overridden method  
    public void methodInToClass() {  
        fromObject.methodInFromClass();  
    }  
}
```



PADRÕES DE PROJETO

- Adapter: Exemplo genérico de Adapter

```
public class ObjectAdapter extends ClassAdaptingTo {  
    private ClassAdaptingFrom fromObject;  
  
    public ObjectAdapter(ClassAdaptingFrom fromObject) {  
        this.fromObject = fromObject;  
    }  
  
    // Overridden method  
    public void methodInToClass() {  
        fromObject.methodInFromClass();  
    }  
}
```

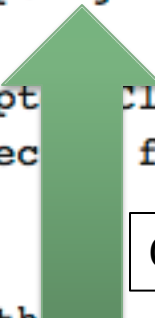


Classe que eu quero a adaptação

PADRÕES DE PROJETO

- Adapter: Exemplo genérico de Adapter

```
public class ObjectAdapter extends ClassAdaptingTo {  
    private ClassAdaptingFrom fromObject;  
  
    public ObjectAdapter(ClassAdaptingFrom fromObject) {  
        this.fromObject = fromObject;  
    }  
  
    // Overridden method  
    public void methodInToClass() {  
        fromObject.methodInFromClass();  
    }  
}
```



Classe de terceiros

PADRÕES DE PROJETO

- Adapter: Exemplo genérico de Adapter

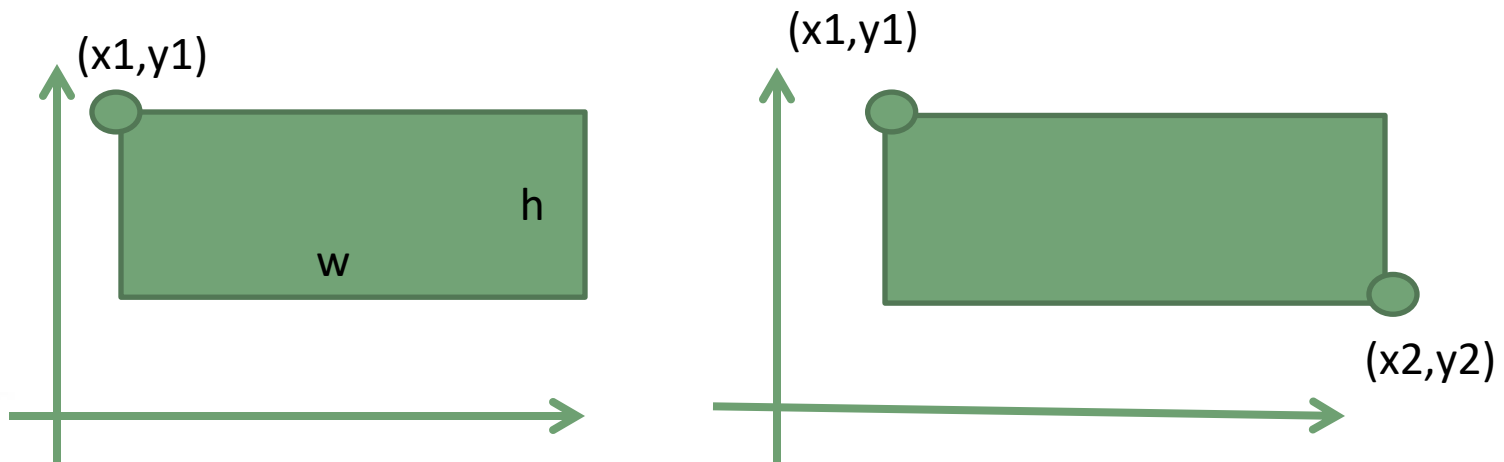
```
public class ObjectAdapter extends ClassAdaptingTo {  
    private ClassAdaptingFrom fromObject;  
  
    public ObjectAdapter(ClassAdaptingFrom fromObject) {  
        this.fromObject = fromObject;  
    }  
  
    // Overridden method  
    public void methodInToClass() {  
        fromObject.methodInFromClass();  
    }  
}
```



Método adaptado

Exemplo Adapter

Dado um componente antigo de Retângulo em que a sua exibição requer as informações em x_1 , y_1 , w e h . Se o cliente utilizar informações do retângulo no seguinte formado x_1 , y_1 , x_2 , y_2 pode-se utilizar o padrão adapter para utilizar adaptar para o componente antigo.



Adapter - Rectangle

Informe a largura e a altura e clique na posição para desenhar o retângulo

Altura

Largura



Clique para marcar o primeiro Ponto e o segundo clique para marcar o segundo ponto



Classe Original do Java

```
package rectangle;

import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class DrawRectangle extends JPanel {

    private int x;
    private int y;
    private int w;
    private int h;
    private JFrame f;

    public DrawRectangle(int x, int y, int w, int h) {
        Compiled Code
    }

    public void paint(Graphics g) {
        Compiled Code
    }
}
```

Esta classe
trabalha com
largura e
altura

DrawRecNew

```
private void formMouseClicked(java.awt.event.MouseEvent evt) {  
    // Posicao clicada e possicao passada pelo usuario  
    DrawRectangle rectangle = new DrawRectangle(evt.getXOnScreen(), evt.getYOnScreen(),  
    Integer.parseInt(this.largura.getText()), Integer.parseInt(this.altura.getText()));  
}
```

Largura

Altura

```
import rectangle.DrawRectangle;
```

Adapter encapsula a
classe rectangle

```
public class AdatperRectangle {  
    private DrawRectangle rectangle;  
    AdatperRectangle(int x1, int y1, int x2, int y2){  
        int w = x2 - x1 ;  
        int h = y2 - y1;  
        rectangle = new DrawRectangle(x1, y1,w , h);  
    }  
}
```



Exemplo de utilização do método com um ArrayList e um String[]

[https://github.com/felipefo/poo2/tree/master/Padrões de Projeto/Estrutural/Adapter/ListaAdapter](https://github.com/felipefo/poo2/tree/master/Padrões%20de%20Projeto/Estrutural/Adapter/ListaAdapter)

Exemplo de utilização do método com um ArrayList

```
import java.util.ArrayList;
import java.util.List;

public class ListaAdapterArrayList {
    public static void main(String[] args) {
        ArrayList lista = new ArrayList();
        lista.add("Pastel");
        lista.add("Quibe");
        processaLista(lista);
    }

    public static void processaLista(List<String> lista){
        for(int index=0; index<lista.size(); index++){
            String value = lista.get(index);
            System.out.println(value);
        }
    }
}
```



Como adaptar uma String[] para um List?

```
import java.util.List;

public class ListaAdapterArray {
    public static void main(String[] args) {
        String[] lista = new String[ 2 ];
        lista[0] = "Pastel";
        lista[1] = "Quibe";
        processaLista(lista);
    }

    public static void processaLista(List<String> lista) {
        for(int index=0; index<lista.size(); index++) {
            String value = lista.get(index);
            System.out.println(value);
        }
    }
}
```

Tenho um String[] e quero usar o mesmo método.... Como fazer???

E agora José?

Método recebe uma lista

Primeira Solução

```
public class ListaAdapterArray {  
    public static void main(String[] args) {  
        String[] lista = new String[2];  
        lista[0] = "Pastel";  
        lista[1] = "Quibe";  
        ArrayList listaArrayList = new ArrayList();  
        int index = 0;  
        for (String valor: lista) {  
            if(valor != null)  
                listaArrayList.add(valor);  
            System.out.print(index);  
            index++;  
        }  
        System.out.println("");  
        processaLista(listaArrayList);  
    }  
    public static void processaLista(List<String> lista){  
        for(int index=0; index<lista.size(); index++){  
            String value = lista.get(index);  
            System.out.println(value);  
        }  
    }  
}
```

0,1 -> várias
iterações!! Se fosse
um array maior??
Muito ineficiente!

Segunda Solução

Do tipo List, pois implementa List

```
public class ArrayStringAdapter implements List {
    private String[] lista;
    public ArrayStringAdapter(String[] lista){
        this.lista = lista;
    }
    @Override
    public int size() {
        return this.lista.length;
    }

    @Override
    public Object get(int i) {
        return lista[i];
    }

    @Override
    public boolean isEmpty() {
        throw new UnsupportedOperationException("Not supported yet."); //
    }
}
```

Classe de adaptação



Segunda Solução

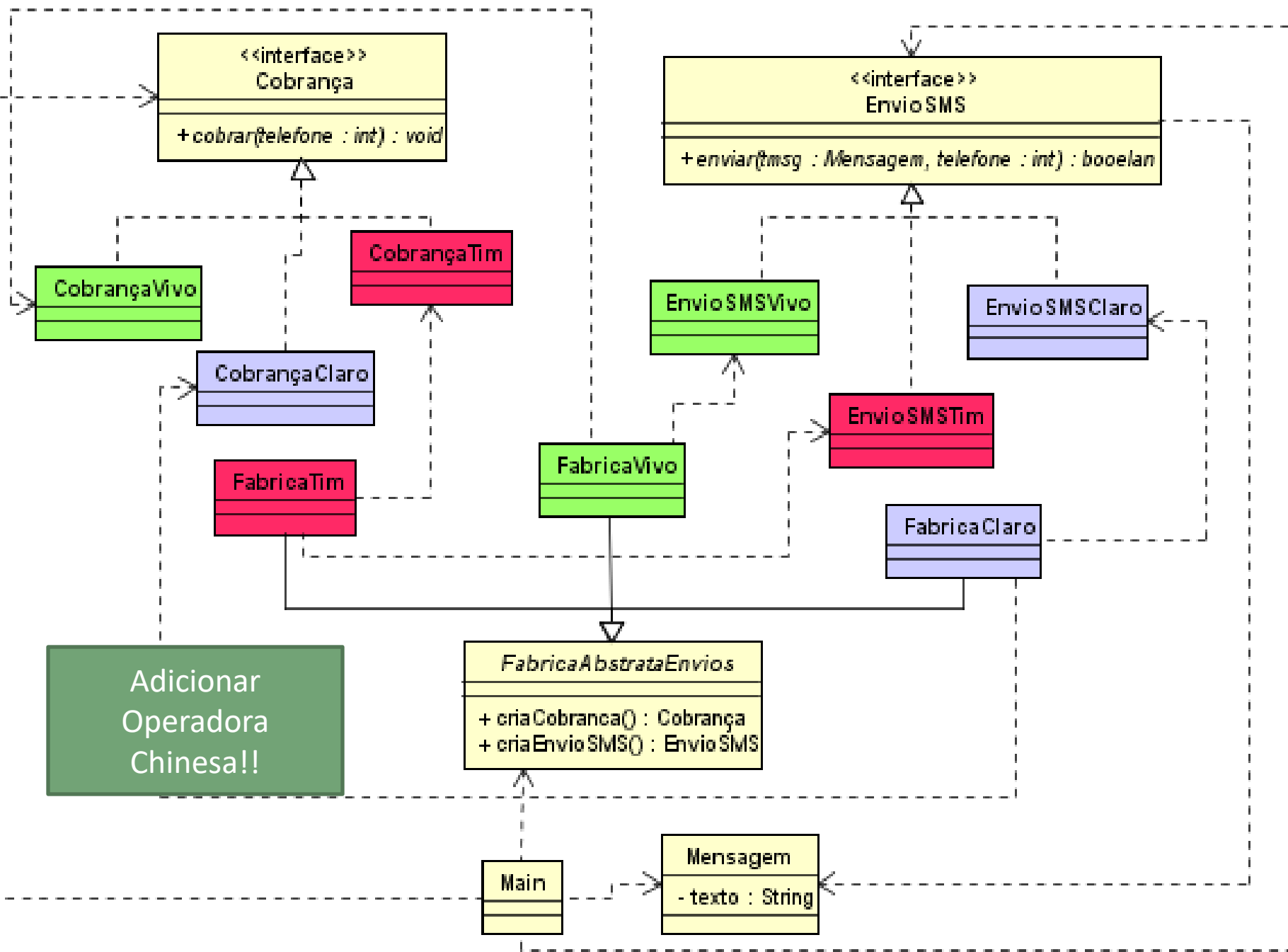
```
public class ListaAdapterArrayComAdapter {  
  
    public static void main(String[] args) {  
        String[] lista = new String[2];  
        lista[0] = "Pastel";  
        lista[1] = "Quibe";  
        ArrayStringAdapter listaAdapter = new ArrayStringAdapter(lista);  
        processaLista(listaAdapter);  
    }  
  
    public static void processaLista(List<String> lista) {  
        for(int index=0; index<lista.size(); index++){  
            String value = lista.get(index);  
            System.out.println(value);  
        }  
    }  
}
```

Usando o
Adapter!

Envio de SMS – Nova operadora

Uma empresa Chinesa entrou no mercado de telefonia brasileiro e o seu sistema precisa ser atualizado para trabalhar com ela.

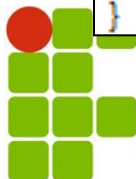
Para a sua sorte a operadora chinesa já tem uma API para ser utilizada! :D



Biblioteca Chinesa

```
public interface EnviadorSMS {  
  
    public boolean enviarSMS(String destino, String[] msgs) ;  
  
}
```

```
public class EnvioSMSChinesa implements EnviadorSMS{  
  
    @Override  
    public boolean enviarSMS(String telefone, String[] msgs) {  
        System.out.println("Enviando peloa chinesa" + telefone  
            + "MSG:" + msgs);  
        return true;  
    }  
}
```



```
public interface EnvioSMS {  
    public boolean enviar(Mensagem msg, int telefone);  
}
```

Seu sistema

```
public interface EnviadorSMS {  
    public boolean enviarSMS(String destino, String[] msgs) ;  
}
```

Chinês

E agora José?

Relação entre os dois:
inteiro -> String
Mensagem -> String[]



Main

```
public static void main(String[] args) {  
  
    String escolha = JOptionPane.showInputDialog("Tim, Vivo ou Chinesa?");  
    int telefone = Integer.parseInt(JOptionPane.showInputDialog("Telefone?"));  
    String texto = JOptionPane.showInputDialog("Mensagem");  
    Mensagem msg = new Mensagem(texto);  
    FabricaAbstrataEnvios fabrica = null;  
    if(escolha.equalsIgnoreCase("tim")) {  
        fabrica = new FabricaTim();  
    } else if(escolha.equalsIgnoreCase("vivo")) {  
        fabrica = new FabricaVivo();  
    } else if(escolha.equalsIgnoreCase("chinesa")) {  
        fabrica = new FabricaChinesa();  
    }  
    EnvioSMS envio = fabrica.criaEnvioSMS();  
    if(envio.enviar(msg, telefone)) {  
        Cobranca cobranca = fabrica.criaCobranca();  
        cobranca.cobrar(telefone);  
    }  
}
```



Classe FabricaVivo

```
public class FabricaVivo extends FabricaAbstrataEnvios{  
  
    @Override  
    public Cobranca criaCobranca() {  
        return new CobrancaVivo();  
    }  
  
    @Override  
    public EnvioSMS criaEnvioSMS() {  
        return new EnvioSMSVivo();  
    }  
}
```

Fabrica Chinesa

```
public class FabricaChinesa extends FabricaAbstrataEnvios{  
  
    @Override  
    public Cobranca criaCobranca() {  
        return new CobrancaChinesa();  
    }  
    @Override  
    public EnvioSMS criaEnvioSMS() {  
        return new EnvioSMSChinesa();  
    }  
}
```

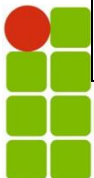
Incompatível!


```
public class FabricaChinesa extends FabricaAbstrataEnvios{  
  
    @Override  
    public EnvioSMS criaEnvioSMS() {  
        return new EnvioSMSChinesa();  
    }  
}
```

Adapter?



```
public class EnvioSMSChinesa implements EnviadorSMS{  
  
    @Override  
    public boolean enviarSMS(String telefone, String[] msgs) {  
        System.out.println("Enviando pela chinesa" + telefone  
            + "MSG:" + msgs);  
        return true;  
    }  
}
```



Resposta não está aqui!!!

Nem aqui!!! Vamos Pense um pouco!!

```

public class AdapterSMSChines implements EnvioSMS{

    EnviadorSMS envioChines = new EnvioSMSChinesa();

    @Override
    public boolean enviar(Mensagem msg, int telefone) {
        String texto = msg.texto;
        String[] textoChines = new String[texto.length()];
        for(int i = 0; i < texto.length(); i++)
        {
            textoChines[i] = String.valueOf(texto.charAt(i));
        }
        return envioChines.enviarSMS(String.valueOf(telefone), textoChines);
    }
}

```

FabricaChinesa

```
public class FabricaChinesa extends FabricaAbstrataEnvios{  
  
    @Override  
    public Cobranca criaCobranca() {  
        return new CobrancaChinesa();  
    }  
  
    @Override  
    public EnvioSMS criaEnvioSMS() {  
        return new AdapterSMSChines();  
    }  
}
```

Adapter SMS - Código JAVA

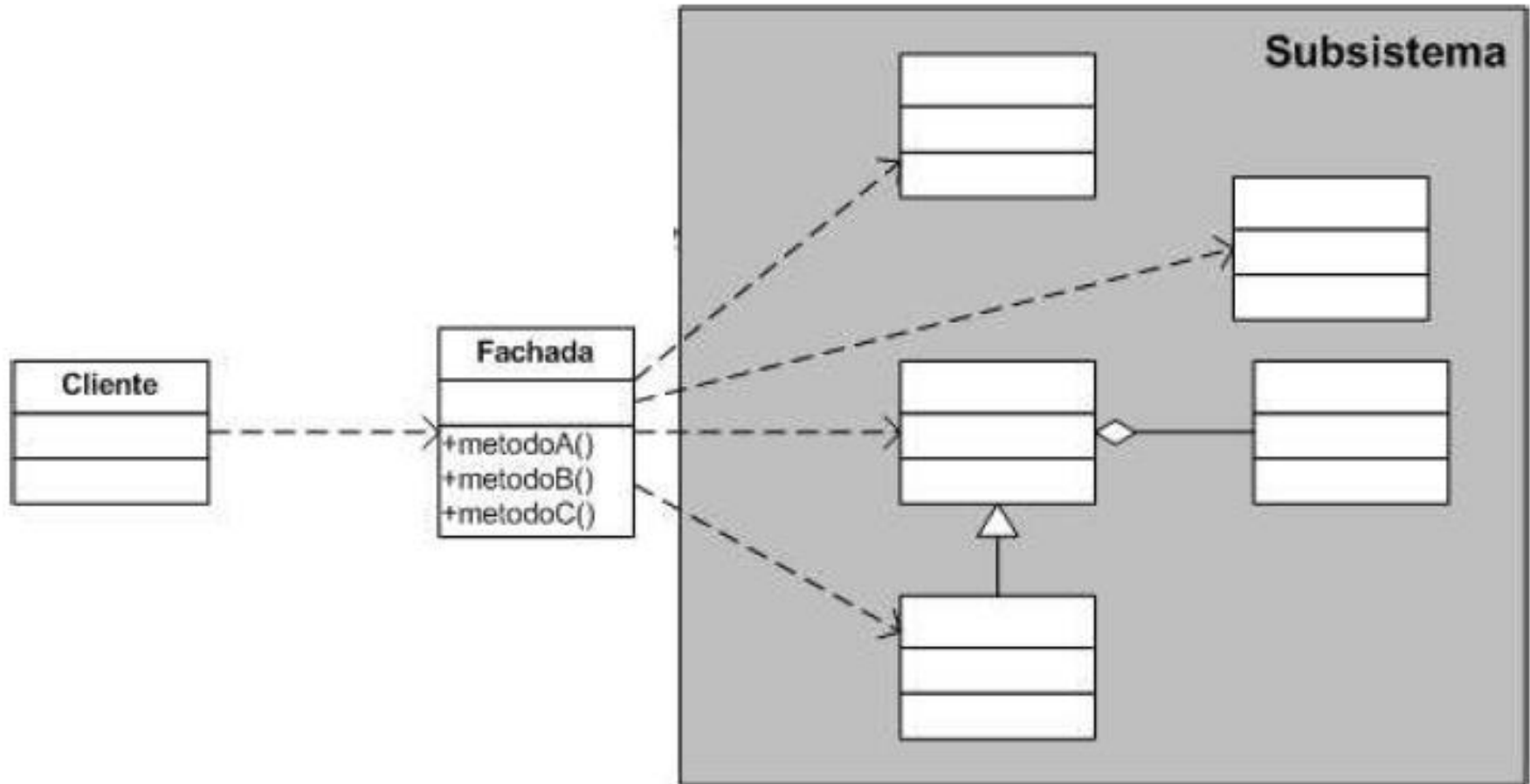
PADRÕES DE PROJETO

Aplicabilidade (Quando usar?):

- Usar uma classe existente, mas sua interface não corresponde à interface de que necessita;
- Criar uma classe reutilizável que coopere com classes não-relacionadas ou não previstas, ou seja, classes que não necessariamente tenham interfaces compatíveis;

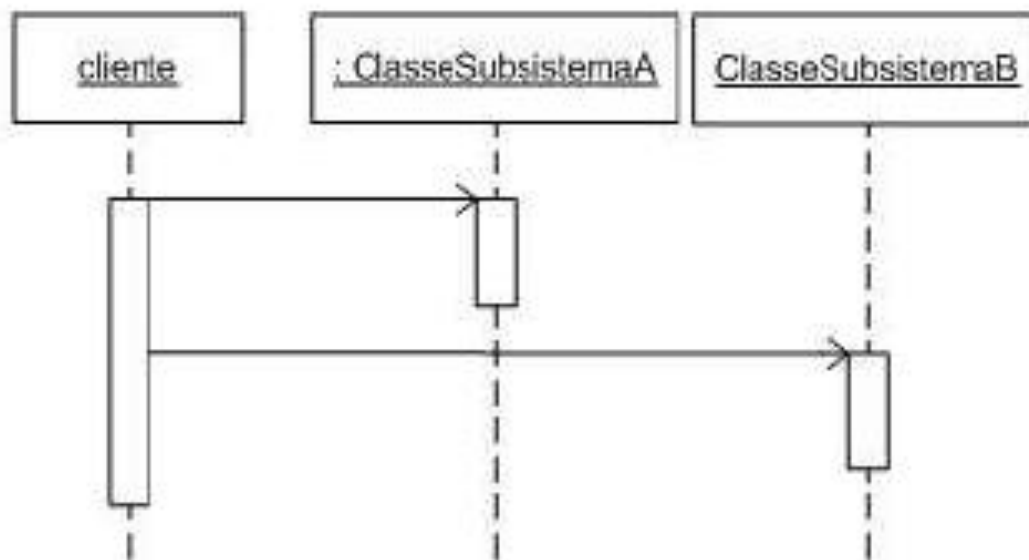
Facade

Facade

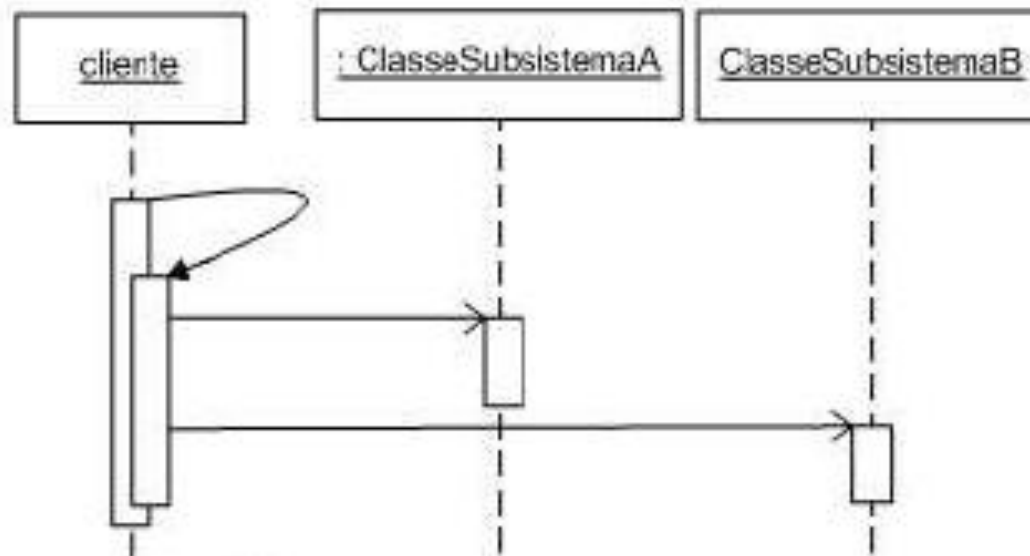


Facade

- A partir desse padrão, é possível isolar um conjunto de classes do resto da aplicação, deixando a fachada como o único ponto de contato.
- Padrão importante para bibliotecas e componentes



1. extrair método que interage com as classes do subsistema



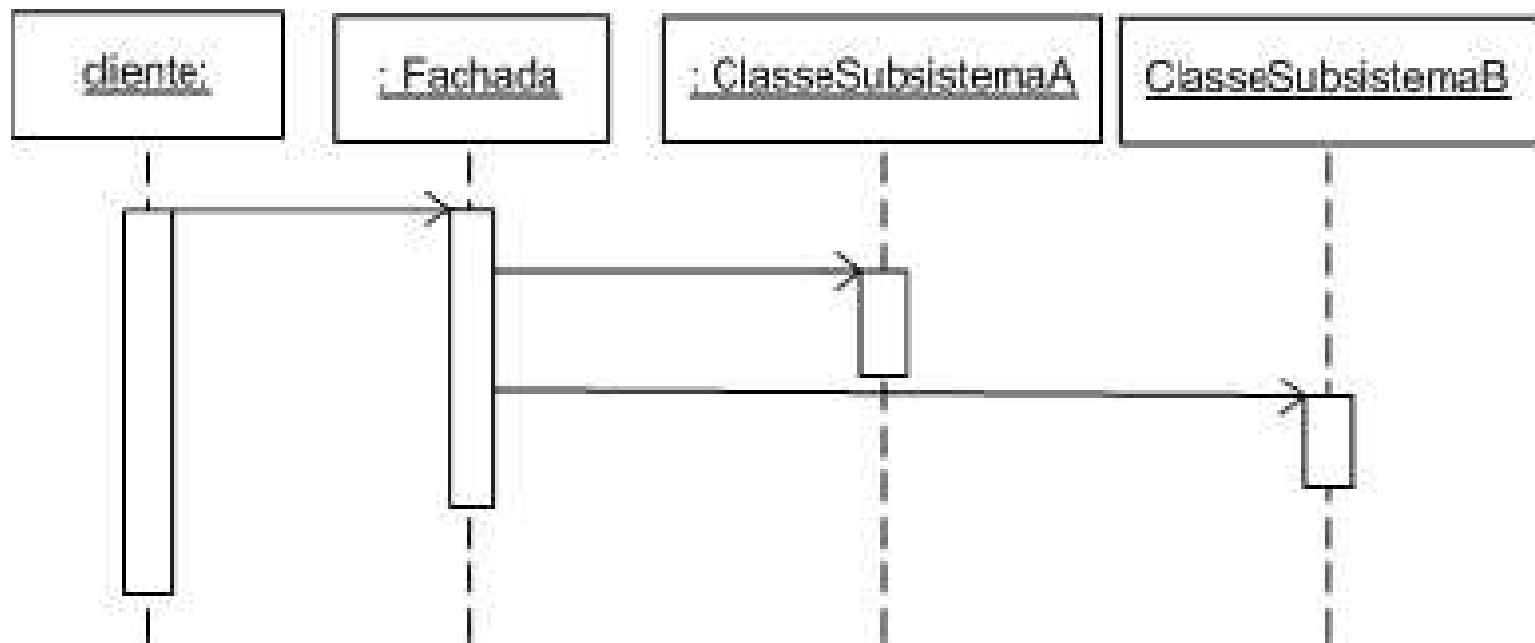


Figura 9.2: Refatorando para a criação de um Facade

Classes da Biblioteca

```
public class HotelBooker{  
    public ArrayList<Hotel> getHotelNamesFor(Date from, Date to)  
    {  
        //returns hotels available in the particular date range }  
    }  
}
```

```
public class FlightBooker{  
    public ArrayList<Flight> getFlightsFor(Date from, Date to)  
    {  
        //returns flights available in the particular date range  
    }  
}
```

Classe Facade

```
public class TravelFacade{  
  
    private HotelBooker hotelBooker;  
    private FlightBooker flightBooker;  
  
    public void getFlightsAndHotels(Date from, Data to) {  
  
        ArrayList<Flight> flights = flightBooker.getFlightsFor(from, to);  
        ArrayList<Hotel> hotels = hotelBooker.getHotelsFor(from, to);  
        //process and return  
  
    }  
}
```

```
public class Client{ public static void  
main(String[] args) {
```

```
    TravelFacade facade = new TravelFacade();  
    facade.getFlightsAndHotels(from, to);
```

```
}
```

```
}
```

Exemplo Gerador de Arquivos

No gerador de arquivos do exemplo a seguir é um caso no qual foram criadas diversas classes para solucionar o problema de gerar um arquivo a partir de um mapa de propriedades.


```

public class GeradorArquivoFacade{
    public void gerarXMLCompactado(
        String nome, Map<String,Object> propriedades
    {
        GeradorArquivo g = new GeradorXML();
        g.setProcessador(new Compactador());
        g.gerarArquivo(nome, propriedades);
    }

    public void gerarPropriedadesCompactado(
        String nome, Map<String,Object> propriedades
    {
        GeradorArquivo g = new GeradorPropriedades();
        g.setProcessador(new Compactador());
        g.gerarArquivo(nome, propriedades);
    }
}

```

Facade para o acesso
a diversos outros
objetos e métodos

Facade para o
acesso a diversos
outros objetos e
métodos



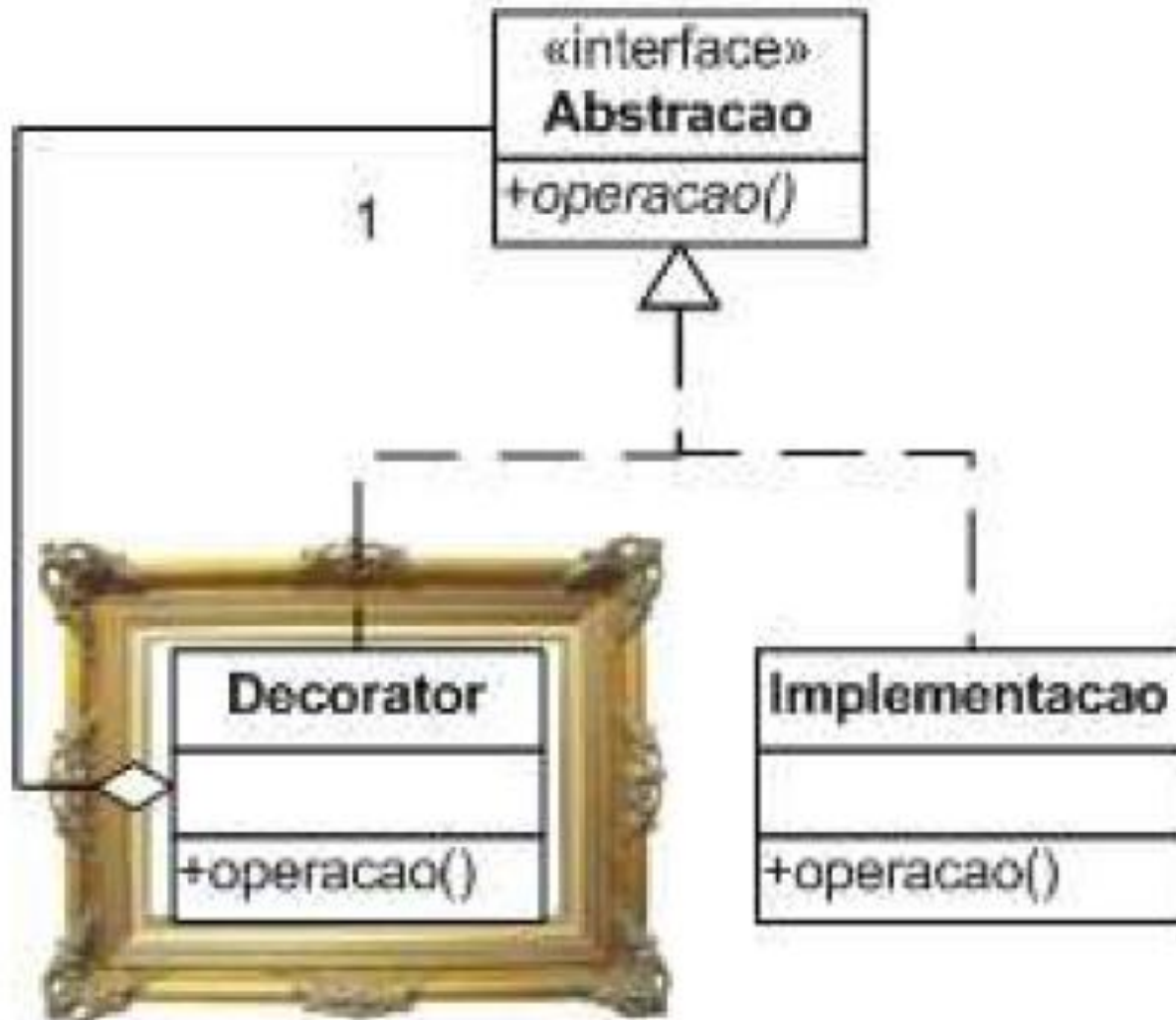
Padrão Decorator

Decorator:

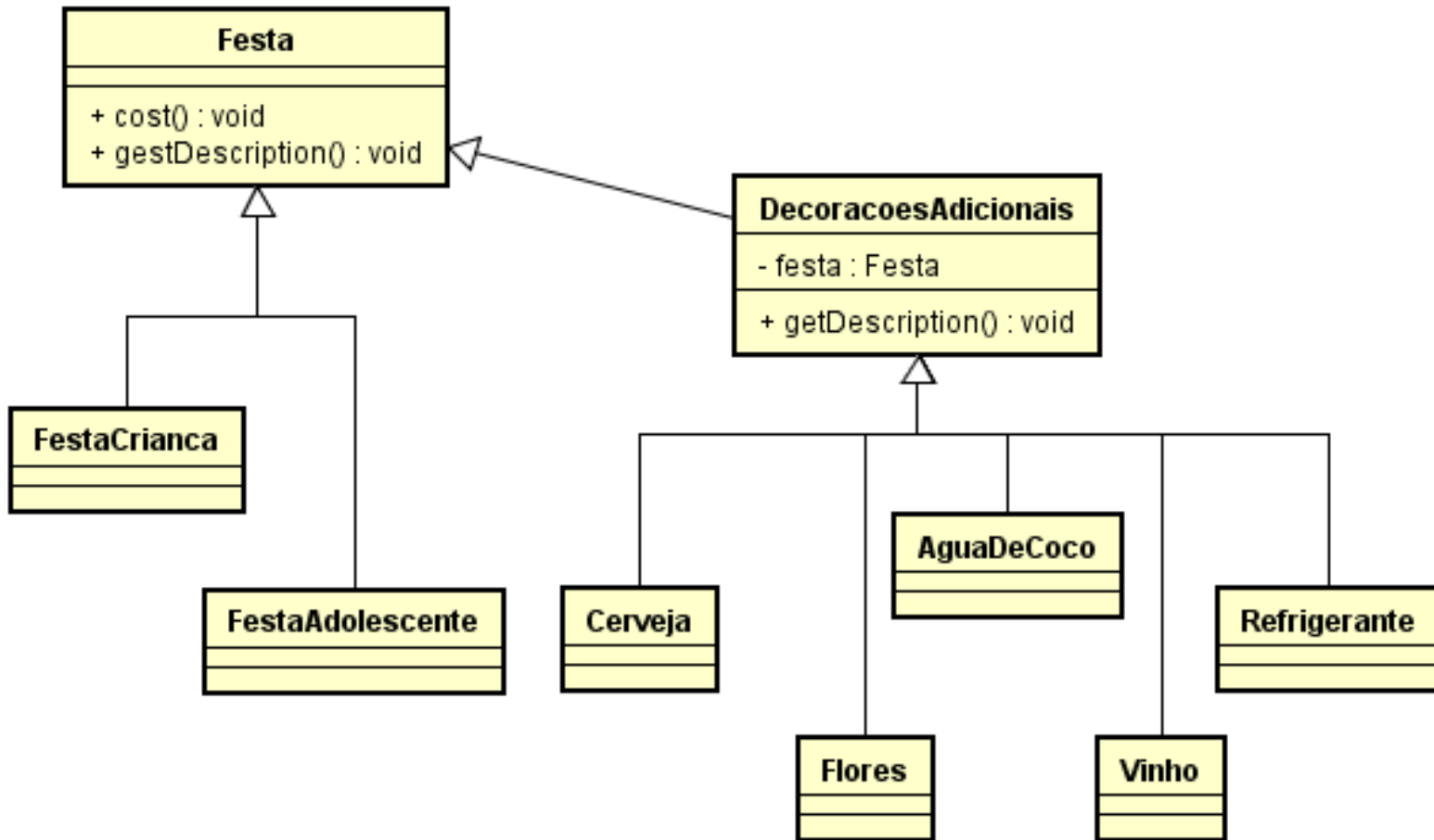
- Propósito: adicionar responsabilidades à um objeto dinamicamente;

Decorator usa a composição recursiva. Isso significa que ele é composto por uma classe que possui a mesma abstração que eles. O padrão Decorator recebeu esse nome relacionado ao fato de "decorar" uma classe existente adicionando uma nova funcionalidade.

Decorator



Padrão Decorator



Exemplo

Decorator - Festa

<http://localhost:8080/DecoratorFesta/>

Padrão Decorator - Festa

Tipo da Festa:

- ☐ Criança - R\$ 5.000,00
- ☐ Adolescente - R\$ 8.000,00

Acessórios da Festa:

- ☐ Decoração com Flores - R\$ 200,00
- ☐ Cerveja - R\$ 500,00
- ☐ Vinho - R\$ 800,00
- ☐ Refrigerante - R\$ 400,00
- ☐ Água de Coco - R\$ 200,00

Enviar

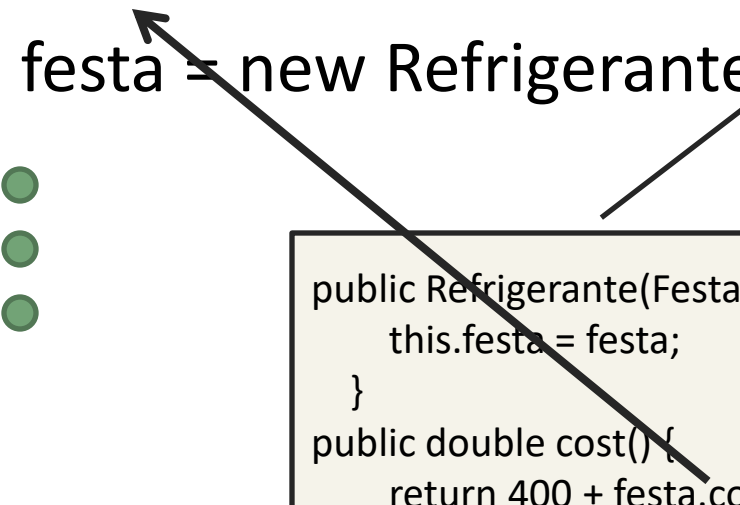
```
String tipo_festa = request.getParameter("tipo_festa");
if (tipo_festa.equalsIgnoreCase("Crianca"))
    festa = new FestaCrianca();
else if (tipo_festa.equalsIgnoreCase("Adolescente"))
    festa = new FestaAdolescente();

if(request.getParameterMap().containsKey("Flores"))
    festa = new Flores(festa);
if(request.getParameter("Cerveja").equalsIgnoreCase("Cerveja"))
    festa = new Cerveja(festa);
if(request.getParameterMap().containsKey("Vinho"))
    festa = new Vinho(festa);
if(request.getParameterMap().containsKey("Refrigerante"))
    festa = new Refrigerante(festa);
if(request.getParameterMap().containsKey("aguadecoco"))
    festa = new AguadeCoco(festa);
```

Padrão Decorator - Festa

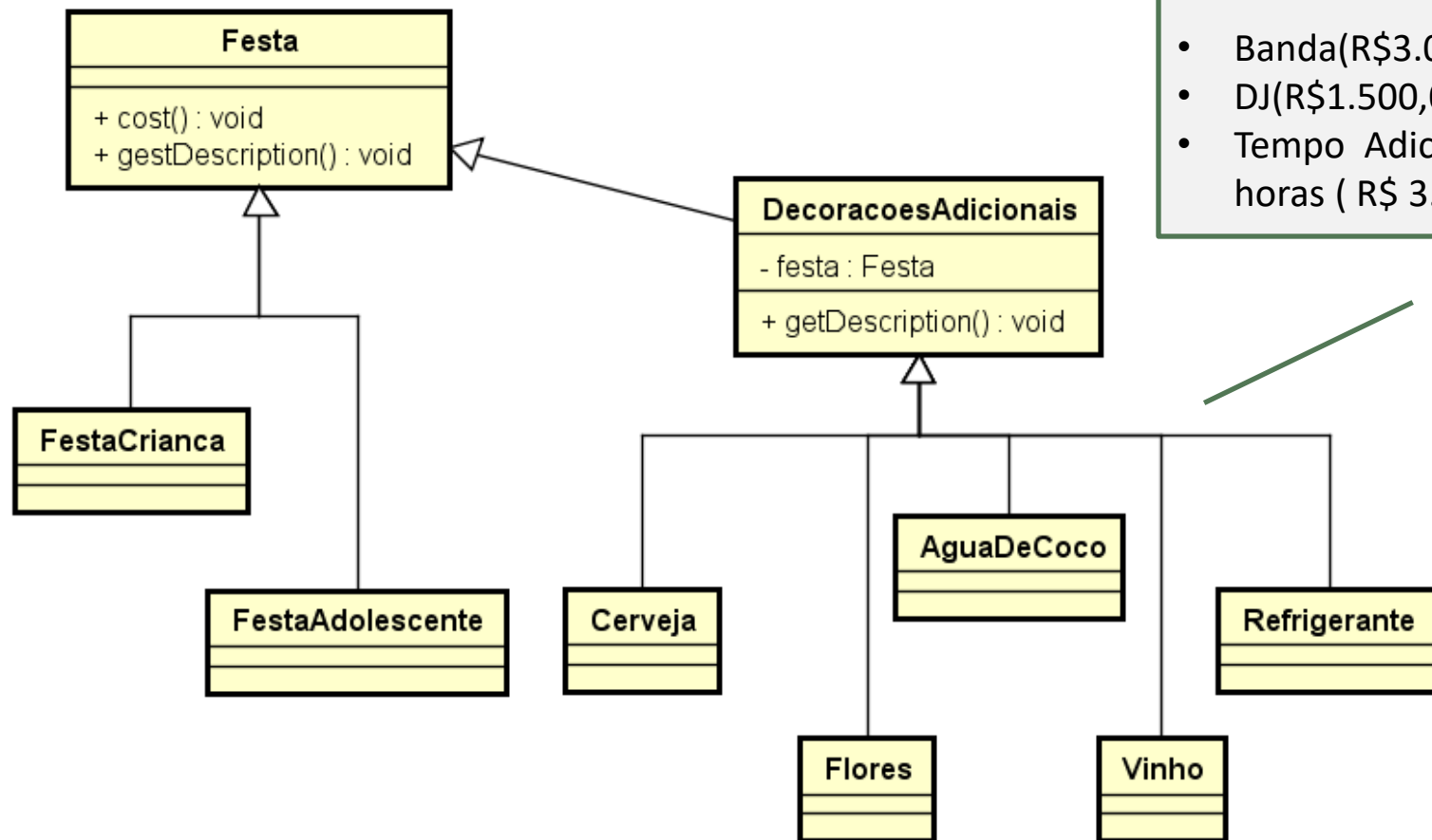
1. festa = new FestaAdolescente();
2. festa = new Flores(festa); ———
3. festa.cost() = 200 + 8.000,00
4. festa = new Refrigerante(festa);

```
public Flores(Festa festa) {  
    this.festa = festa;  
}  
public double cost() {  
    return 200 + festa.cost();  
}  
}
```



```
public Refrigerante(Festa festa) {  
    this.festa = festa;  
}  
public double cost() {  
    return 400 + festa.cost();  
}  
}
```

Padrão Decorator - Festa

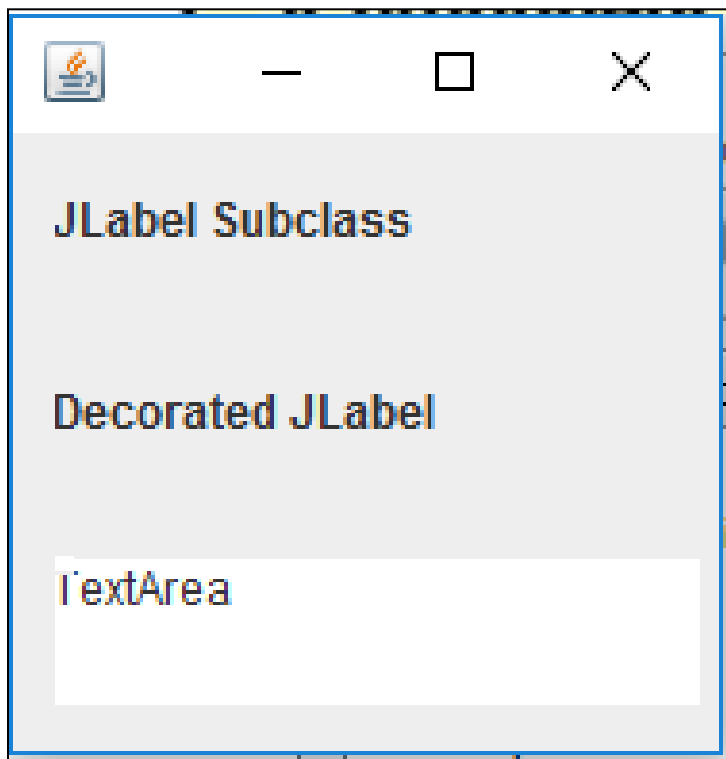


Adicione os seguintes acessórios a Festa:

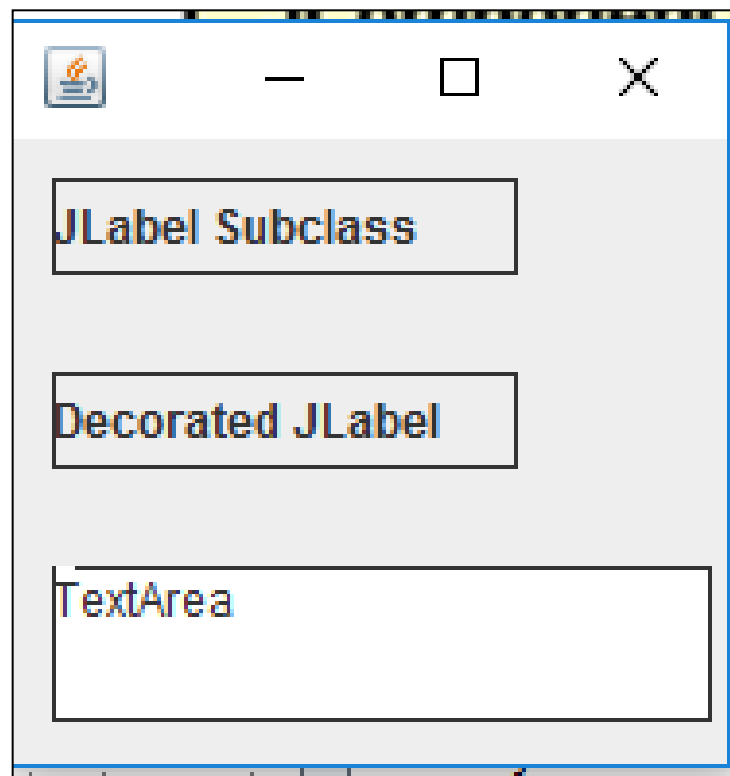
- Banda(R\$3.000,00)
- DJ(R\$1.500,00)
- Tempo Adicional de 2 horas (R\$ 3.000,00)



Antes

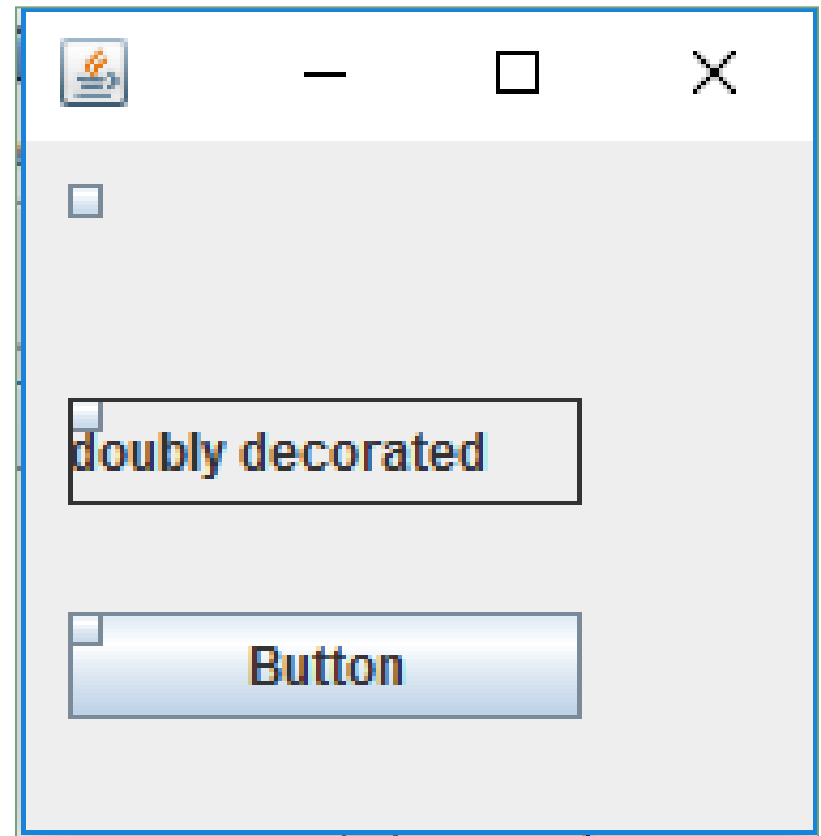
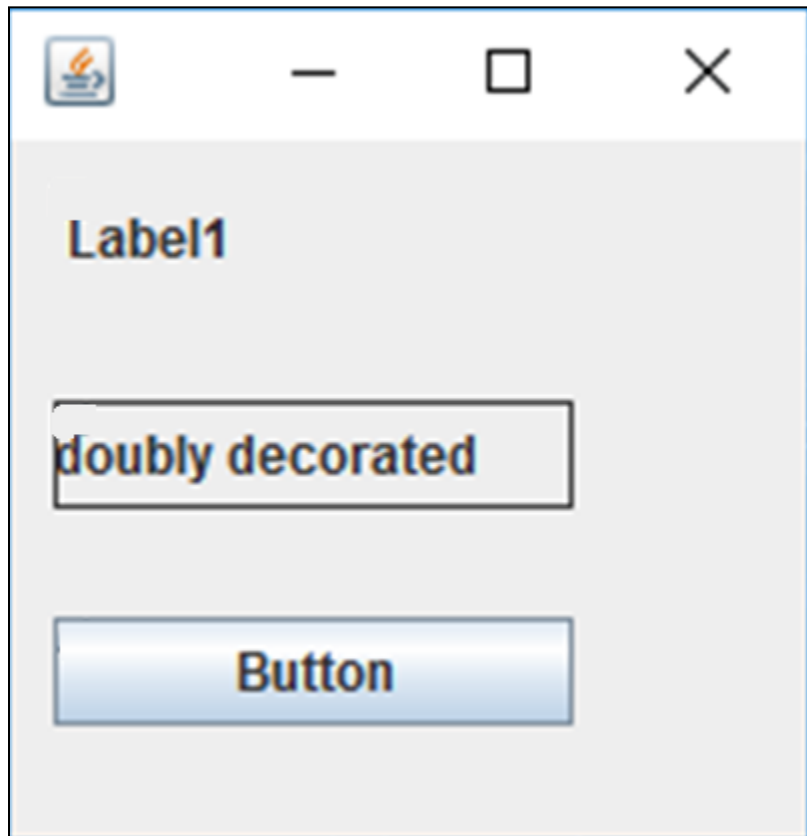


Depois

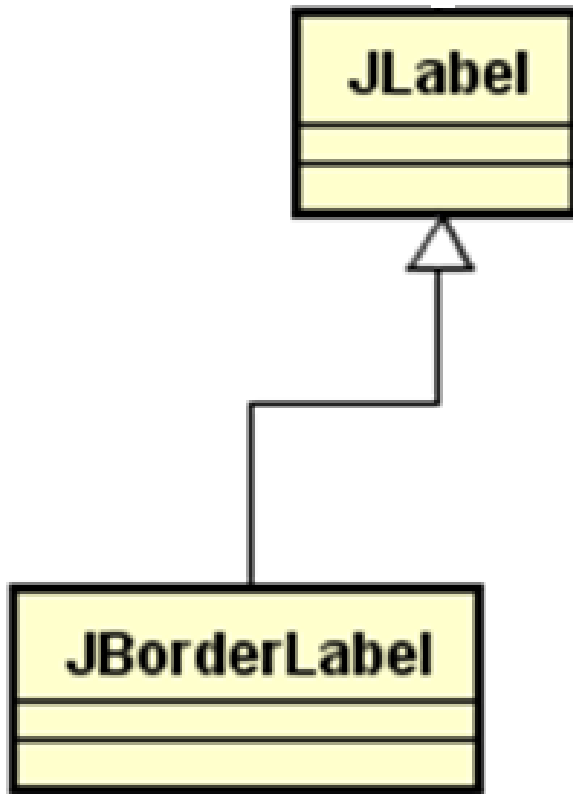


Antes

Depois

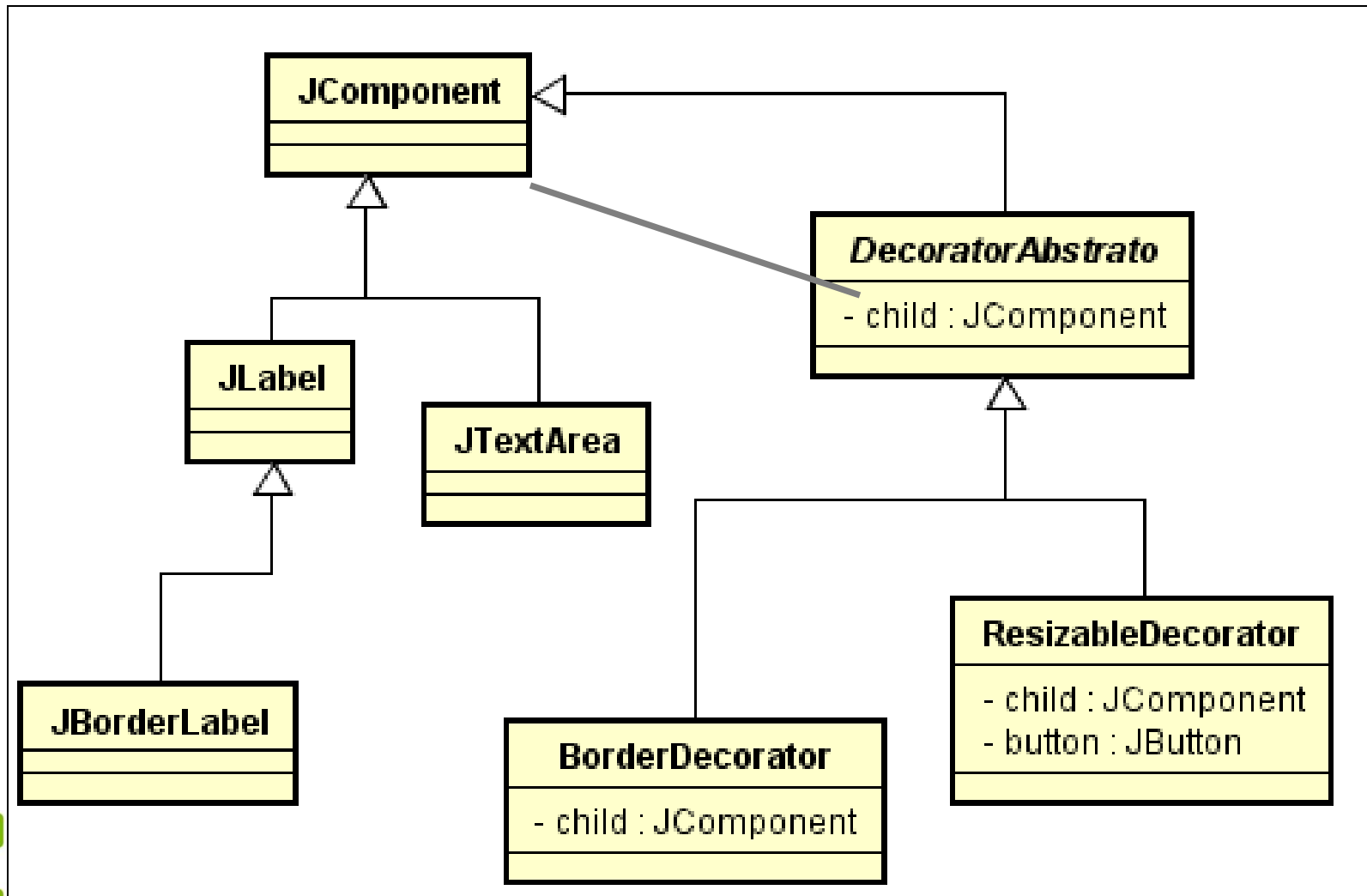


Posso estender e implementar a borda diretamente no JLabel



Mas se eu quiser adicionar a borda em um TextArea?? Vou ter que estender o TextArea também?

Decorator – Componentes de Interface



```
public class JBorderLabel extends JLabel {

    public JBorderLabel() {
        super();
    }

    public JBorderLabel(String text) {
        super(text);
    }

    public JBorderLabel(Icon image) {
        super(image);
    }

    public JBorderLabel(String text, Icon image, int horizontalAlignment) {
        super(text, image, horizontalAlignment);
    }

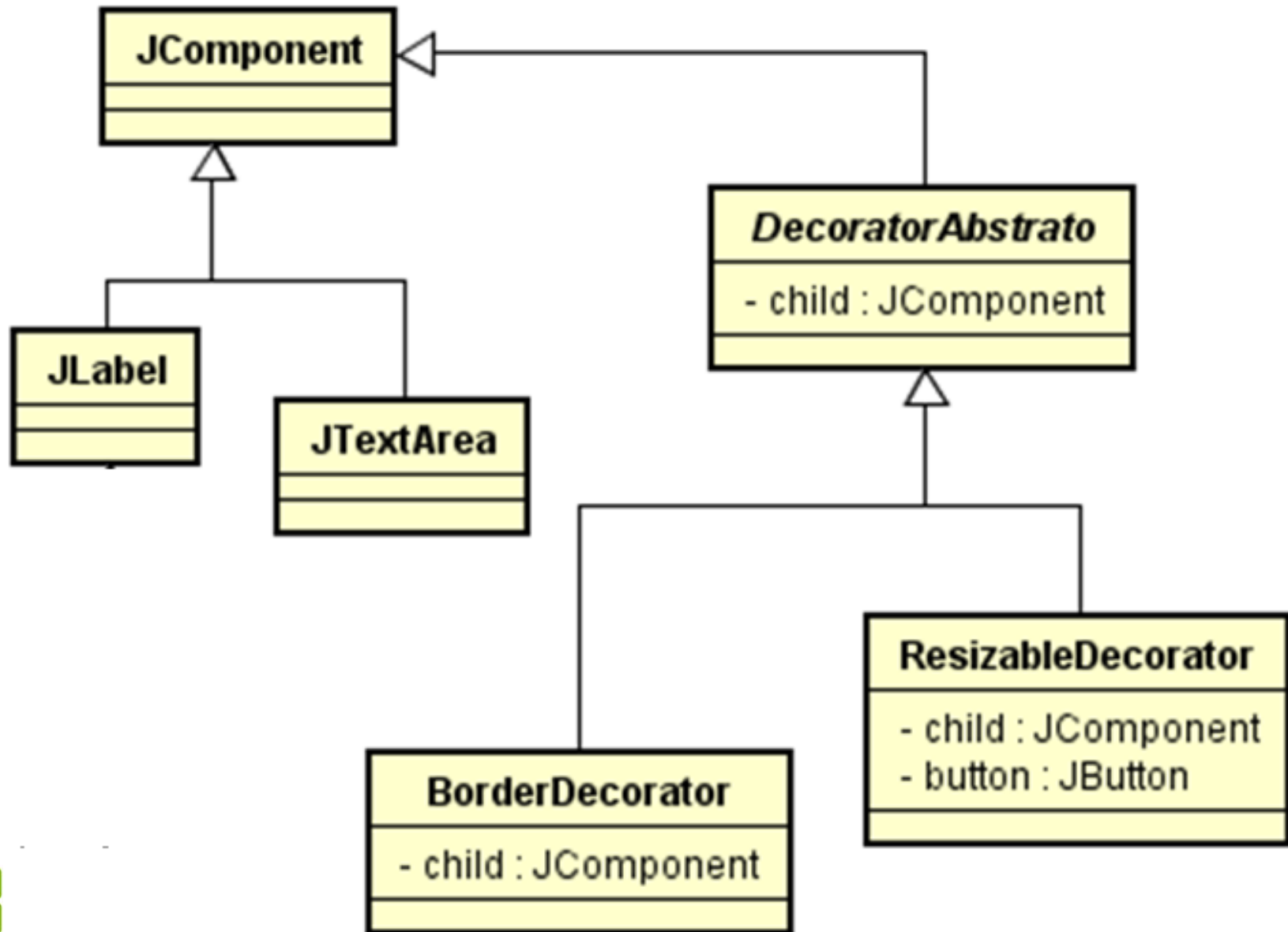
    public JBorderLabel(String text, int horizontalAlignment) {
        super(text, horizontalAlignment);
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        int height = this.getHeight();
        int width = this.getWidth();
        g.drawRect(0, 0, width - 1, height - 1);
    }
}
```

Desenhando
a borda

Sem o padrão

```
public class Frame1 extends JFrame {  
  
    JBorderLabel label1 =  
        new JBorderLabel("JLabel Subclass");//sem o padrao...
```



Utilizando o Padrão

```
package decorator;

import javax.swing.JComponent;

public abstract class DecoratorAbstrato extends JComponent {
    protected JComponent child;
}
```



```
package decorator;
```

```
- import javax.swing.JComponent;  
  import java.awt.Graphics;  
  import java.awt.BorderLayout;
```

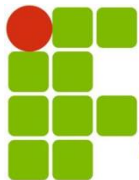
```
public class BorderDecorator extends DecoratorAbstrato{
```

```
    // decorated component
```

```
- public BorderDecorator(JComponent component) {  
    child = component;  
    this.setLayout(new BorderLayout());  
    this.add(child);  
}
```

```
- public void paint(Graphics g) {  
    super.paint(g);  
    int height = this.getHeight();  
    int width = this.getWidth();  
    g.drawRect(0, 0, width - 1, height - 1);  
}
```

Desenhando
a borda



Decorando o Objeto
Label e TextArea

```
BorderDecorator label2 =  
    new BorderDecorator(new JLabel("Decorated JLabel"));  
BorderDecorator textArea = new BorderDecorator(  
    new JTextArea("TextArea"));
```

Percebam que a decoração serve para ambos
os Tipos pois eles são do tipo JComponent
A vantagem é que ele fica
independente da decoração

Criando um outro Decorador...

Resizable...

```

public class ResizableDecorator extends DecoratorAbstrato {

    private JButton button = new JButton();
    boolean minimum = false;
    private Rectangle r;

    public ResizableDecorator(JComponent component) {
        child = component;
        this.setLayout(new BorderLayout());
        this.add(child);
        child.setLayout(null);
        button.setBounds(0, 0, 8, 8);
        child.add(button);
        button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                button_actionPerformed(e);
            }
        });
    }

    void button_actionPerformed(ActionEvent e) {
        if (minimum) {
            this.setBounds(r);
        }
        else {
            r = this.getBounds();
            this.setBounds(r.x, r.y, 8, 8);
        }
    }
}

```

Botão para
aumentar e
diminuir

Quando clica no
botão minimiza
ou maximiza...



```
import java.awt.event.*;
```

```
public class Frame1 extends JFrame {
```

```
    JBorderLabel label1 =
```

```
        new JBorderLabel("JLabel Subclass");
```

```
    BorderDecorator label2 =
```

```
        new BorderDecorator(new JLabel("Decorated JLabel"));
```

```
    BorderDecorator textArea = new BorderDecorator(  
        new JTextArea("TextArea"));
```

```
    public Frame1() {
```

```
        try {
```

```
            this.setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
            getContentPane().setLayout(null);
```

```
            label1.setBounds(new Rectangle(10, 10, 120, 25));
```

Decorando o Objeto



```

import javax.swing.*;
import java.awt.event.*;

public class Frame2 extends JFrame {
    ResizableDecorator label1 =
        new ResizableDecorator(new JLabel("Label1"));
    ResizableDecorator button1 =
        new ResizableDecorator(new JButton("Button"));
    BorderDecorator label2 =
        new BorderDecorator(new ResizableDecorator(
            new JLabel("doubly decorated")));

    public Frame2() {
        e.printStackTrace();
    }
}

```

Percebam que o JLabel
não foi alterado!! Qual a
vantagem???

Decorando o Objeto e
Redecorando o Objeto



Código - Decorator Windows

PADRÕES ESTRUTURAIS

Composite

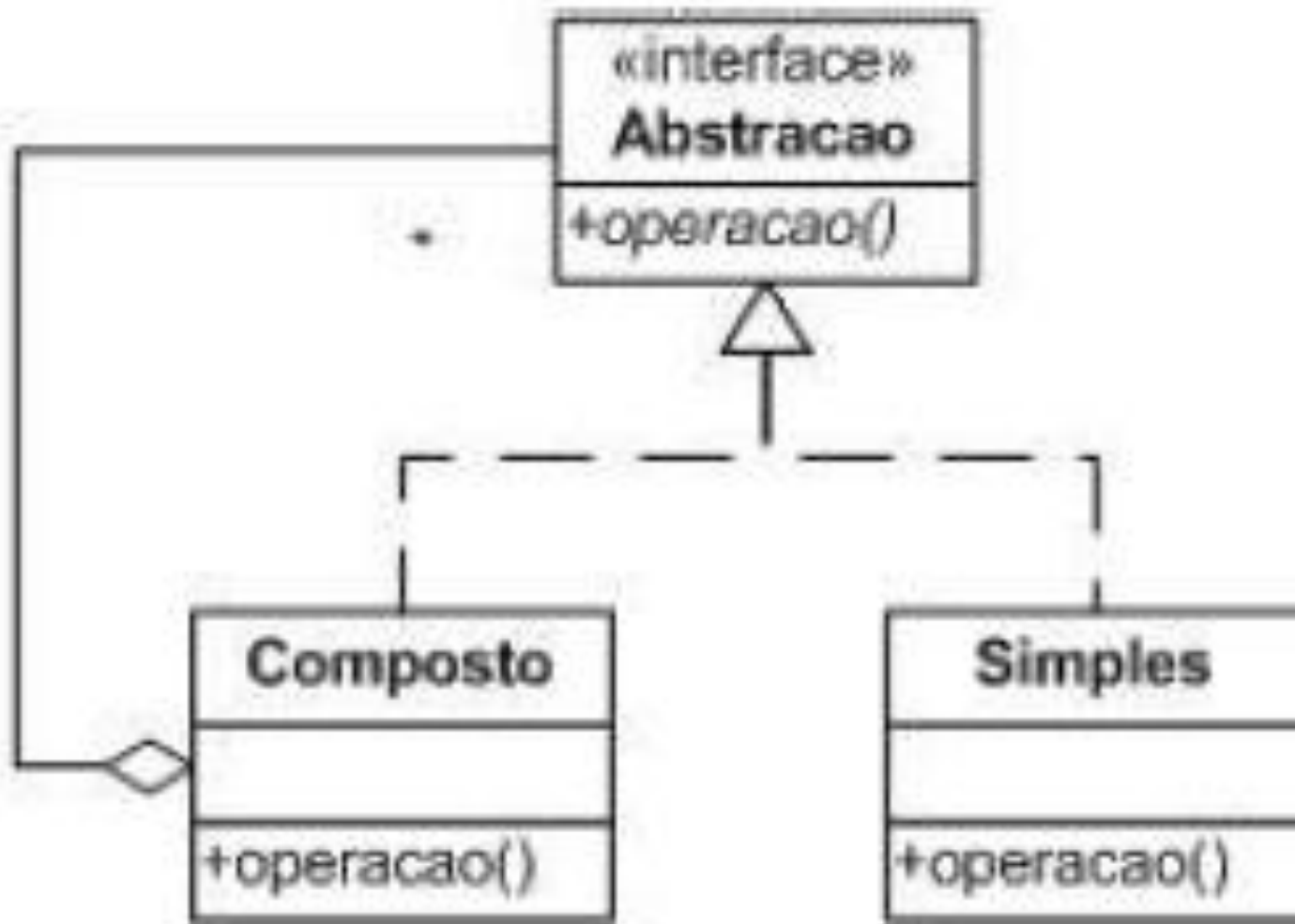
PADRÕES DE PROJETO

Composite:

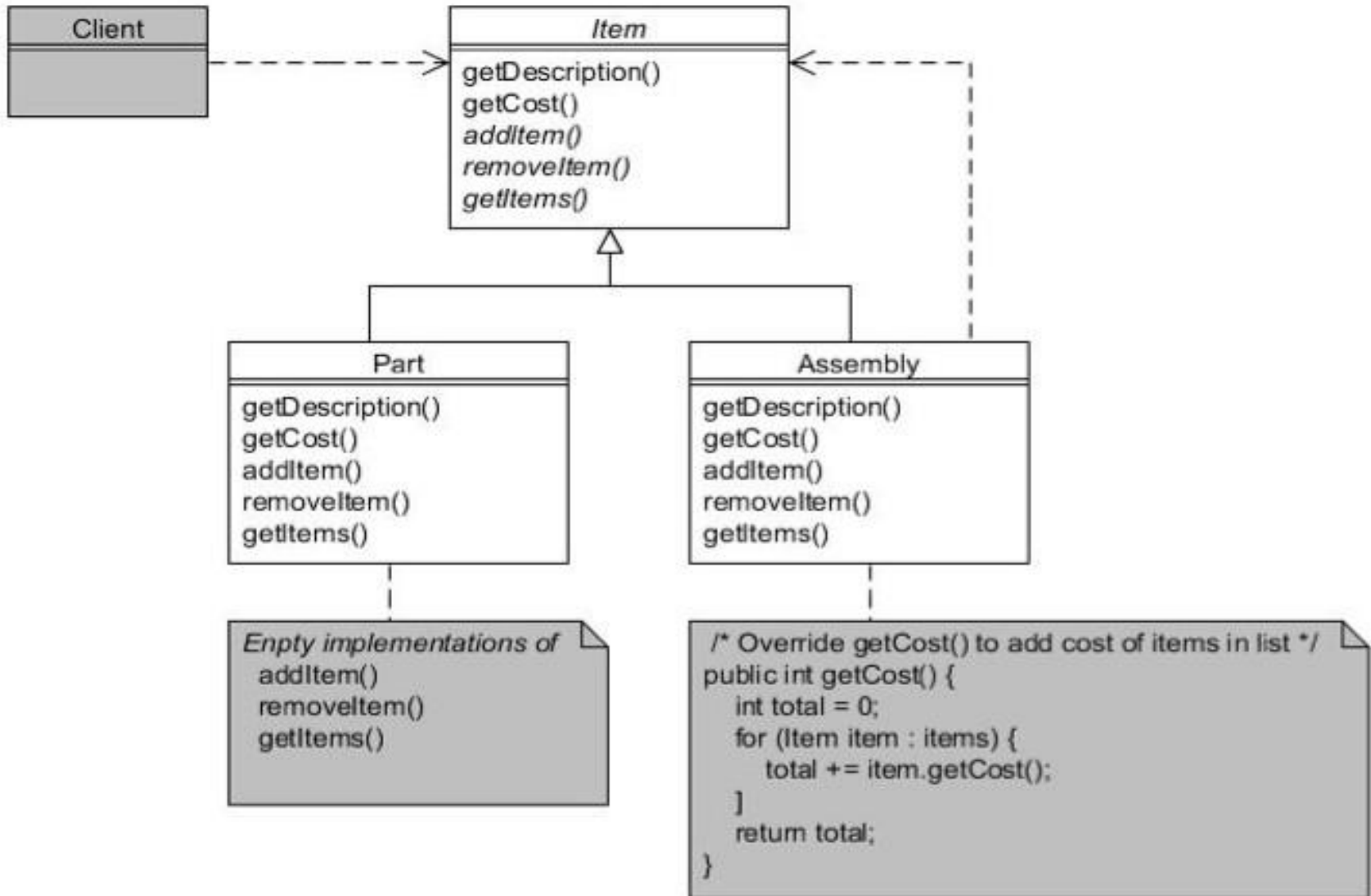
Propósito: tem o objetivo de permitir que a mesma abstração possa ser utilizada para uma instância simples e para seu conjunto.

– Problema:

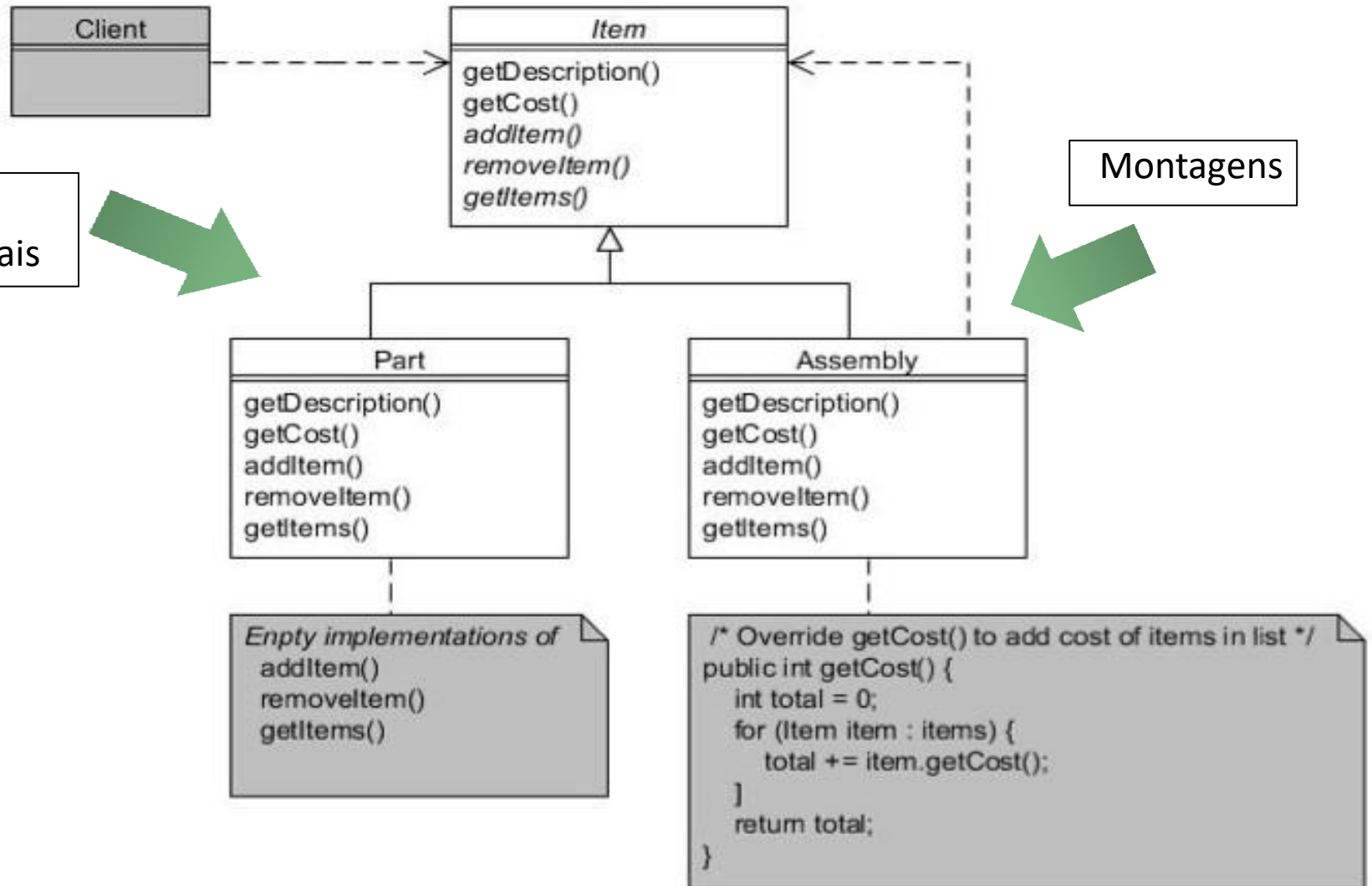
- A Motores SA constrói diversos componentes utilizando pequenas partes como porcas, parafusos, painéis e etc. Cada item possui um custo.
- Esses itens menores são utilizados para produzir componentes maiores;
- O valor dos componente menores é a soma de suas partes menores.



PADRÕES DE PROJETO



PADRÕES DE PROJETO



PADRÕES DE PROJETO

```
public abstract class Item {
    private String description;
    private int cost;

    public Item(String description, int cost) {
        this.description = description;
        this.cost = cost;
    }

    public String getDescription() {
        return description;
    }

    public int getCost() {
        return cost;
    }

    public abstract void addItem(Item item);
    public abstract void removeItem(Item item);
    public abstract Item[] getItems();

    public String toString() {
        return description + " (cost " + getCost() + ")";
    }
}
```

```
public class Assembly extends Item {
    private List<Item> items;

    public Assembly(String description) {
        super(description, 0);
        items = new ArrayList<Item>();
    }

    public void addItem(Item item) {
        items.add(item);
    }

    public void removeItem(Item item) {
        items.remove(item);
    }

    public Item[] getItems() {
        return items.toArray(new Item[items.size()]);
    }

    // Also have to override getCost() to add cost of items in list
    public int getCost() {
        int total = 0;
        for (Item item : items) {
            total += item.getCost();
        }
        return total;
    }
}
```

Lista de itens!

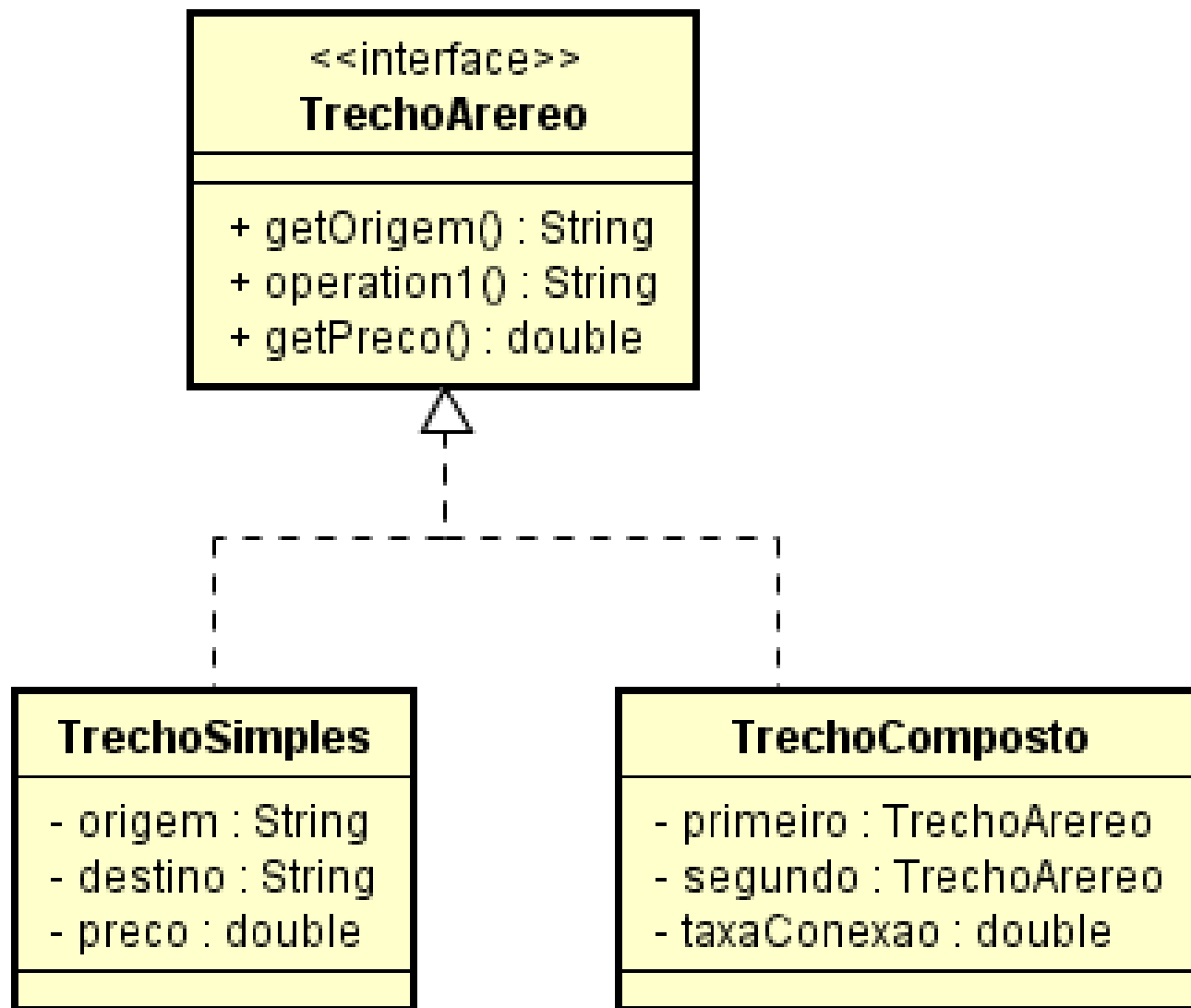
Composite

```
Item velocimetro = new Part("Velocimetro" , 200);
Item contadorDeGiros = new Part("Contador de Giros" , 300);
Item hodômetro = new Part("Hodometro" , 500);
Item marcadoDeCombustivel = new Part("Marcador de Combustivel" , 200);

Item painelInstrumentos = new Assembly("Painel Instrumentos");
painelInstrumentos.addItem(velocimetro);
painelInstrumentos.addItem(contadorDeGiros);
painelInstrumentos.addItem(hodômetro);
painelInstrumentos.addItem(marcadoDeCombustivel);
```

Exemplo de composição com trechos de viagem de uma companhia área.

Para Viajar de São Paulo -> Natal



Interface TrechoAereo

```
public interface TrechoAereo {  
    public String getOrigem();  
    public String getDestino();  
    public double getPreco();  
}
```

```
public class TrechoSimples implements TrechoAereo{
    private String origem;
    private String destino;
    private double preco;
    public TrechoSimples(String origem, String destino,
        double preco) {
        this.origem = origem;
        this.destino = destino;
        this.preco = preco;
    }
    public String getOrigem() {
        return origem;
    }
    public String getDestino() {
        return destino;
    }
    public double getPreco() {
        return preco;
    }
}
```



```

public class TrechoComposto implements TrechoAereo {
    private TrechoAereo primeiro;
    private TrechoAereo segundo;
    private double taxaconexao;

    public TrechoComposto(TrechoAereo primeiro,
        TrechoAereo segundo, double taxaconexao) {
        this.primeiro = primeiro;
        this.segundo = segundo;
        this.taxaconexao = taxaconexao;
        if(primeiro.getDestino() != segundo.getOrigem())
            throw new RuntimeException("O destino do pri
meiro" +
                "não é igual a origem do segundo");
    }
    public String getOrigem() {
        return primeiro.getOrigem();
    }
    public String getDestino() {
        return segundo.getDestino();
    }
    public double getPreco() {
        return primeiro.getPreco() + segundo.getPreco()
            +
                taxaconexao;
    }
}

```



Usando o padrão

Veja a criação de trechos compostos:

```
TrechoSimples ts1 =  
    new TrechoSimples("São Paulo", "Brasília", 500);  
TrechoSimples ts2 = new TrechoSimples("Brasília", "Recife", 300);  
TrechoSimples ts3 = new TrechoSimples("Recife", "Natal", 350);  
TrechoComposto tc1 = new TrechoComposto(ts2, ts3, 30);  
TrechoComposto tc2 = new TrechoComposto(ts1, tc1, 50);
```

PADRÕES ESTRUTURAIS

Bridge

Bridge

Propósito: Cria uma ponte entre duas hierarquias ligadas por uma relação de composição permitindo que ambas variem de forma independente.

Problema:

- A Motores SA quer adicionar as seguintes funcionalidades aos motores:
 - Desligar e ligar do motor;
 - Aumento e decremento do poder do motor;
- O motor deve ser projetado e adicionado em um veículo sem a necessidade de nenhum componente ser modificado;
- Os componentes do carro devem ser projetados e adicionados ao veículo sem a necessidade de modificar o motor;

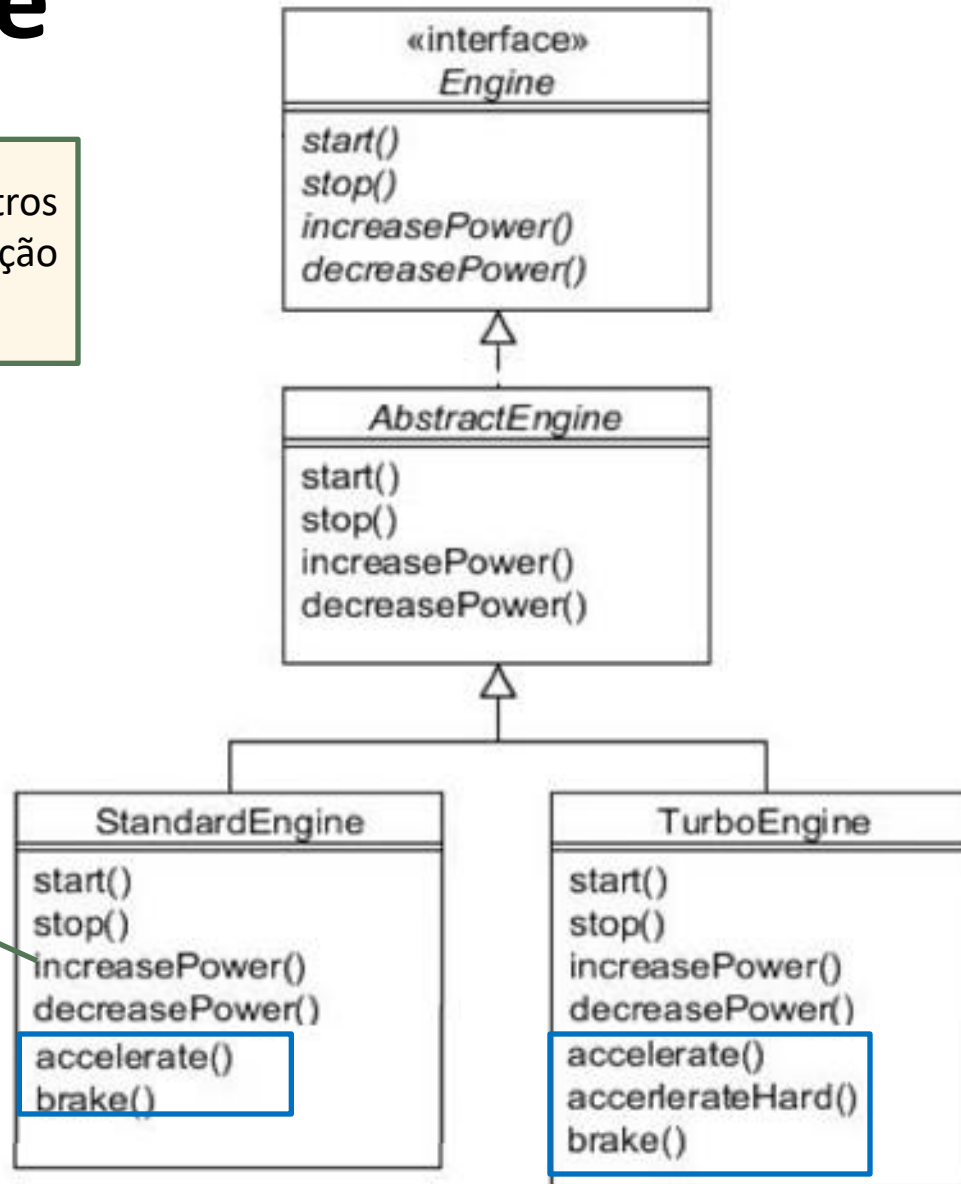
Bridge

Adicionar outros
Controles? Aceleração
Econômica

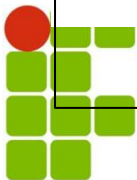
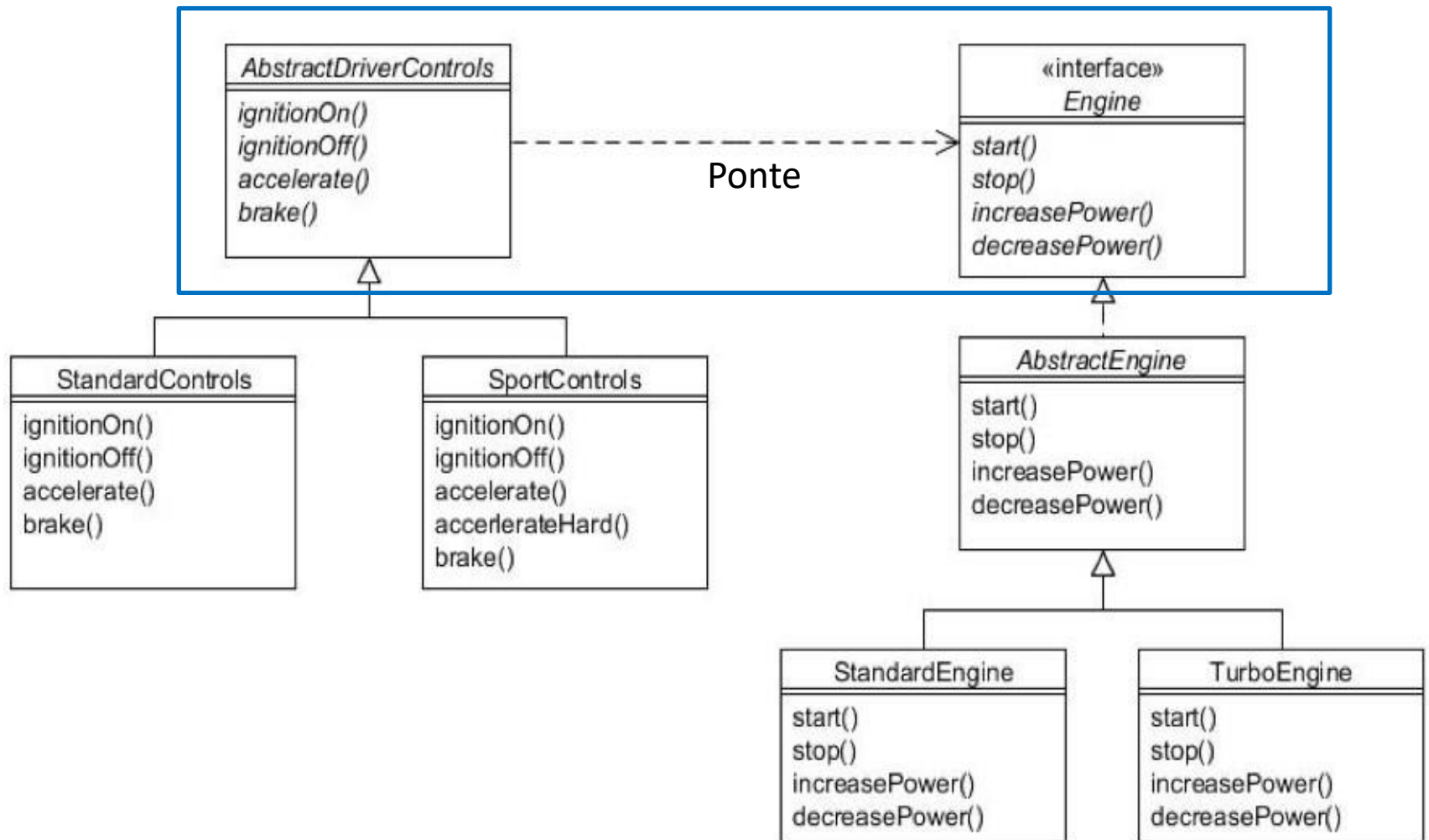
Adicionar o
GreenEngine?

Controles

Controles



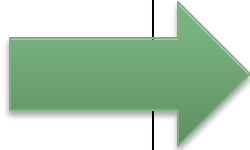
Bridge



PADRÕES DE PROJETO

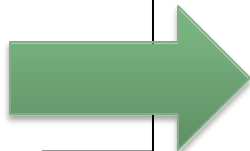
- **Bridge:**

interface



```
public interface Engine {  
    public int getSize();  
    public boolean isTurbo();  
  
    public void start();  
    public void stop();  
    public void increasePower();  
    public void decreasePower();  
}
```

implementação



```
public abstract class AbstractEngine implements Engine {  
    private int size;  
    private boolean turbo;  
    private boolean running;  
    private int power;  
  
    public AbstractEngine(int size, boolean turbo) {  
        this.size = size;  
        this.turbo = turbo;  
        running = false;  
        power = 0;  
    }
```

AbstractEngine(Cont.)

```
public void start() {
    running = true;
    System.out.println("Engine started");
}

public void stop() {
    running = false;
    power = 0;
    System.out.println("Engine stopped");
}

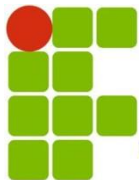
public void increasePower() {
    if (running && (power < 10)) {
        power++;
        System.out.println("Engine power increased to " + power);
    }
}

public void decreasePower() {
    if (running && (power > 0)) {
        power--;
        System.out.println("Engine power decreased to " + power);
    }
}
```



```
public abstract class AbstractDriverControls {  
    private Engine engine;  
  
    public AbstractDriverControls(Engine engine) {  
        this.engine = engine;  
    }  
  
    public void ignitionOn() {  
        engine.start();  
    }  
  
    public void ignitionOff() {  
        engine.stop();  
    }  
  
    public void accelerate() {  
        engine.increasePower();  
    }  
  
    public void brake() {  
        engine.decreasePower();  
    }  
}
```

Variando a implementação do motor. Posso passar diferentes motores aqui, ou seja, outros tipos de implementação de motores



Brigde

```
public class StandardControls extends AbstractDriverControls {  
    public StandardControls(Engine engine) {  
        super(engine);  
    }  
  
    // No extra features  
}
```

```
public class SportControls extends AbstractDriverControls {  
    public SportControls(Engine engine) {  
        super(engine);  
    }  
  
    public void accelerateHard() {  
        accelerate();  
        accelerate();  
    }  
}
```

O controle sport
acelera mais
rapidamente o carro

Utilizando...

//utilizando turbe engine com o turbo control

```
TurboEngine tegine = new TurboEngine(1500,true);
```

```
SportsControl sc = new SportsControl(tengine);
```

//utilizando o standard engine com o sportcontrol..

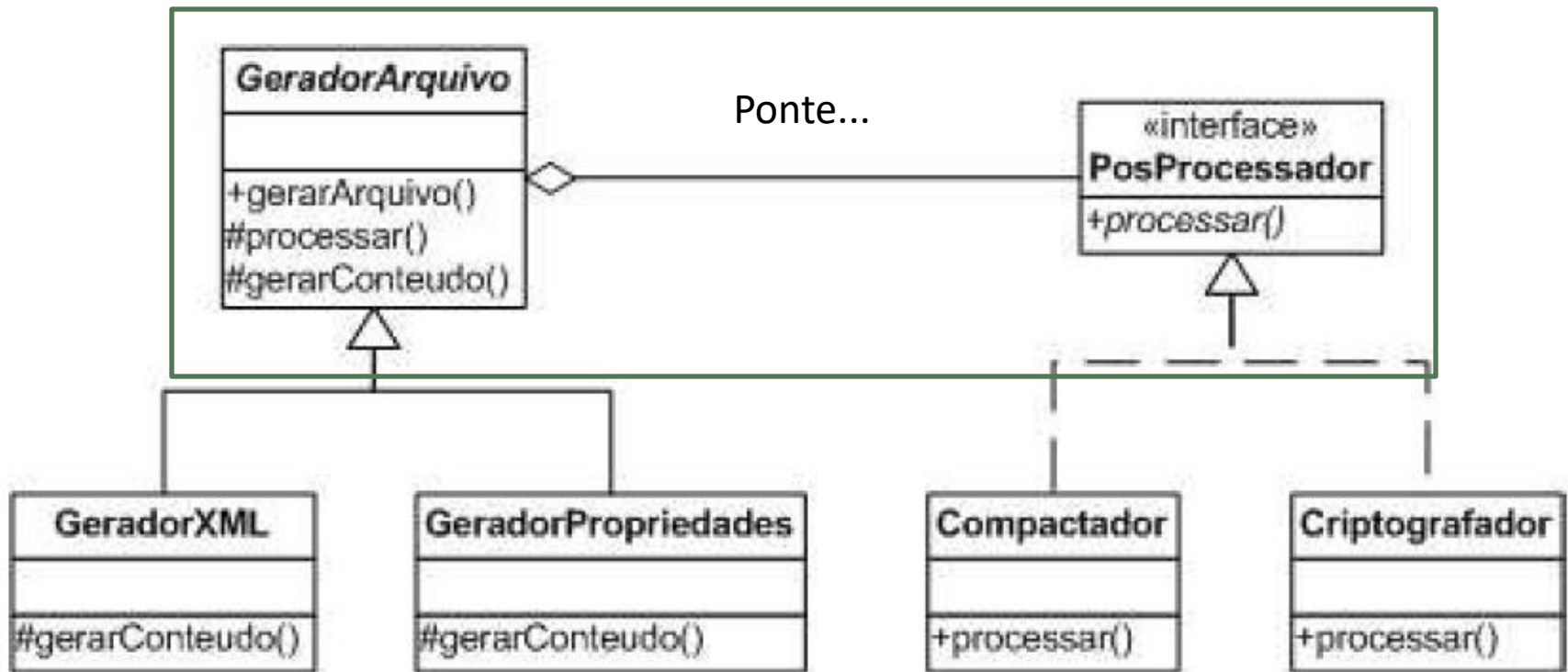
```
StandardEngine segine = new StandardEngine(1000, false);
```

```
SportsControl sc = new SportsControl(sengine);
```

//utilizando o standard engine com o standards control..

```
StandardsControl scontrol = new StandardsControl (sengine);
```

Bridge



Variando a implementação

```
public abstract class GeradorArquivo {  
    private PosProcessador processador;
```

```
    public void setProcessador(PosProcessador processado  
r){  
        this.processador = processador;  
    }  
}
```

```
    public final void gerarArquivo(String nome,  
                                   Map<String, Object> propr  
iedades)
```

```
        throws IOException {  
            String conteudo = gerarConteudo(propriedades);  
            byte[] bytes = conteudo.getBytes();  
            bytes = processador.processar(bytes);  
            FileOutputStream fileout = new FileOutputStream(  
nome);  
            fileout.write(bytes);  
            fileout.close();  
        }  
}
```

Utilizando o processador



Com o GeradorArquivo

```
GeradorArquivo ga = new GeradorArquivo();
```

```
PosProcessador proCripto = new Criptografador();
```

```
PosProcessador proCompac = new Compactador();
```

```
ga.setProcessador(proCripto);
```

Ou

```
ga.setProcessador(proCompac );
```

Com o GeradorXML

```
GeradorXML gXML = new GeradorXML();
```

```
PosProcessador proCripto = new Criptografador();
```

```
PosProcessador proCompac = new Compactador();
```

```
gXML.setProcessador(proCripto);
```

Ou

```
gXML.setProcessador(proCompac );
```

Brigde

- **Aplicabilidade:**

- Deseja evitar um vínculo permanente entre uma abstração e sua implementação. Isso pode acontecer, por exemplo, quando a implementação deve ser selecionada ou alterada em tempo de execução;
- Deseja compartilhar uma implementação entre múltiplos objetos e este fato deve estar oculto do cliente.

https://sourcemaking.com/design_patterns/

<http://www.oodeesign.com/decorator-pattern-gui-example-java-sourcecode.html>

<https://code.tutsplus.com/pt/tutorials/design-patterns-the-decorator-pattern--cms-22641>

<https://github.com/chantastic/css-patterns>

<http://archive.oreilly.com/pub/a/onjava/2003/02/05/decorator.html?page=1>

http://tads-ifrsrg.blogspot.com.br/2014/05/padrao-de-projeto-flyweight_1.html