**Roteiro SOLID 07**

Nos **roteiros 06 e 07** iremos analisar algumas boas práticas que serão úteis no entendimento de Padrões de Projeto. Estas boas práticas são desdobramentos dos princípios vistos no SOLID dos roteiros anteriores.

1 – Programe para uma interface e não para uma implementação

2 – Favoreça o uso de Composição sobre Herança

**Neste roteiro em específico daremos foco ao item 2**

**Pacote : roteiro7.parte1**

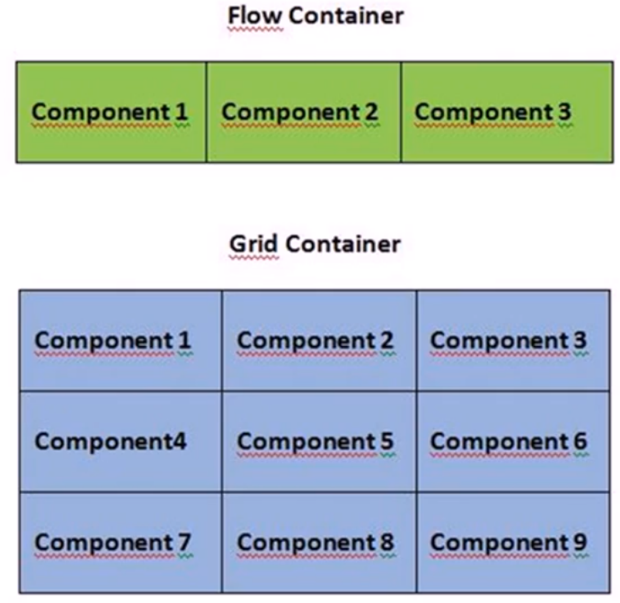
1 – Dentro do projeto criar um pacote chamado **roteiro7.parte1**

2 – Para este roteiro temos um novo cenário. Trata-se de um pequeno projeto que tem como requisito simular a criação de containers para agrupar componentes de interface gráfica com o usuário (Botões, campo de texto, checkboxs, labels e etc ). Precisaremos de 2 grupos de containers.

- Um chamado Flow Container: agrupa os componentes da esquerda para a direita em uma única linha.

- O outro chamado Grid Container: agrupa os componentes em uma matriz de linhas e colunas, inserindo os componentes da esquerda para a direita e de cima para baixo.

Cada um dos containers deve permitir a inserção e remoção dos containers, bem como a renderização de si próprio após ter sido configurado. Também deveremos implementar a funcionalidade de fechar um container que faz com que o mesmo não seja mais exibido em tela.



3 – Inicialmente temos a primeira classe chamada **Component**. Esta classe deve ser criada no pacote **roteiro7.parte1.**

package roteiro7.parte1;

public class Component {

private String nome;

public Component(String nome) {

this.nome = nome;

}

public String getNome() {

return nome;

}

public void setNome(String nome) {

this.nome = nome;

}

@Override

public String toString() {

return nome;

}

}

Ela deve ter um atributo privado para representar o nome do componente e um construtor que deverá receber o nome do componente como parâmetro. Além disso, deve-se criar os getters e setters deste atributo. Usaremos também um recurso da OO que é sobrecarga de métodos, e iremos sobrescrever o método toString() para retornar nome do componente que criamos quando formos imprimir.

4 – A segunda classe a ser criada chamaremos de **Container**, mas esta deve ser uma classe **abstrata**.

Dentro dela criaremos **3 métodos abstratos** (sem implementação, semelhante a uma interface) :

package roteiro7.parte1;

public abstract class Container {

public abstract void addComponent (Component c);

public abstract void removeComponent (Component c);

public abstract void doLayout();

public void dispose(){

System.out.println("Fechando o Container ");

}

}

**addComponent** – para adicionar um componente ao container.

**removeCompoponent** – para remover um determinado componente do container.

**doLayout** – para renderizar e organizar os componentes dentro do container.

E **um método concreto** :

**dispose** – para fechar o container.

O próximo passo será criar as classes **FlowContainer** e **GridContainer** usando o conceito de herança a partir da Classe **Container**.

Elas serão **subclasses concretas** da classe Container, e por isso os métodos abstratos serão implementados apenas nas classes específicas de cada container respeitando as suas características.

5 – Crie agora a classe **FlowContainer** como subclasse da classe **Container** por meio de herança

package roteiro7.parte1;

import java.util.ArrayList;

import java.util.List;

public class FlowContainer extends Container {

private List<Component> elements;

public FlowContainer() {

this.elements = new ArrayList<Component>();

}

@Override

public void addComponent(Component c) {

this.elements.add(c);

}

@Override

public void removeComponent(Component c) {

this.elements.remove(c);

}

@Override

public void doLayout() {

System.out.println("O Container utilizado é o FlowContainer");

System.out.println("Estes são os elementos presentes no container");

System.out.println(elements);

System.out.println("Usando o método dispose como herança para fechar o container");

this.dispose();

System.out.println("--------------------------------");

}

}

Observe que comparando os itens 5 e 6, foram criadas implementações específicas para cada tipo de container nos métodos : addComponent, removeComponent, doLayout. Estes métodos foram especificados na superclasse abstrata, mas a implementação foi feita na classe concreta.

6 – Utilizando o mesmo recurso, crie a classe **GridContainer**.

package roteiro7.parte1;

import java.util.Arrays;

import java.util.List;

public class GridContainer extends Container {

private Component[][] elements;

private int lineCounter = 0;

private int columnCounter = 0;

private int ColumnMax;

private int lineMax;

public GridContainer(int lineMax, int ColumnMax) {

this.lineMax = lineMax;

this.ColumnMax = ColumnMax;

this.elements = new Component[lineMax][ColumnMax];

}

@Override

public void addComponent(Component c) {

if (this.lineCounter == this.lineMax && this.columnCounter == this.ColumnMax) {

System.out.println("Não é possivel adicionar novos elementos");

} else {

this.elements[this.lineCounter][this.columnCounter] = c;

this.columnCounter++;

if (columnCounter == this.ColumnMax) {

this.lineCounter++;

if (this.lineCounter < this.lineMax) {

this.columnCounter = 0;

}

}

}

}

@Override

public void removeComponent(Component c) {

for (int i = 0; i < this.lineMax; i++) {

for (int j = 0; j < this.ColumnMax; j++) {

if (this.elements[i][j] == c) {

this.elements[i][i] = null;

}

}

}

}

@Override

public void doLayout() {

System.out.println("O Container utilizado é o GridContainer");

System.out.println("Estes são os elementos presentes no container");

System.out.println(Arrays.deepToString(elements));

System.out.println("Usando o método dispose como herança para fechar o container");

this.dispose();

System.out.println("--------------------------------");

}

}

7 – Para fazer os testes de execução crie a classe **TesteHeranca**

package roteiro7.parte1;

public class TesteHeranca {

public static void main(String[] args) {

Component button = new Component("Button");

Component textBox = new Component("TextBox");

Component textField = new Component("TextField");

Component checkBox = new Component("CheckBox");

Component combobox = new Component("ComboBox");

Component label = new Component("Label");

Component radiobutton = new Component("RadioButton");

// Criando um GridContainer para adicionar os elementos

Container c1 = new GridContainer(2, 2);

c1.addComponent(button);

c1.addComponent(textBox);

c1.addComponent(textField);

c1.addComponent(checkBox);

c1.doLayout();

System.out.println(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

// Criando um FlowContainer para adicionar os elementos

Container c2 = new FlowContainer();

c2.addComponent(combobox);

c2.addComponent(label);

c2.addComponent(radiobutton);

c2.doLayout();

}

}

8 – Utilize uma ferramenta de software qualquer para geração do diagrama de classes para esta etapa do projeto (Sugestão : Astah Community). Obs.: Adicione aqui o diagrama para que seja disponibilizado no teams

**Pacote : roteiro7.parte2**

Imagine agora que o nosso projeto precise evoluir e teremos que encontrar uma solução para que os containers possam ter bordas sólidas ou bordas pontilhadas, ou simplesmente não tenham bordas.

1 – No mesmo projeto crie o pacote roteiro7.parte2

2 – Copie todas as classes criadas na parte1 para o novo pacote.

3 – Crie agora a classe **FlowContainerBordaSolida** como subclasse da classe **Container**.

package roteiro7.parte2;

import java.util.ArrayList;

public class FlowContainerBordaSolida extends Container{

private ArrayList<Component> elements;

public FlowContainerBordaSolida() {

this.elements = new ArrayList<Component>();

}

@Override

public void addComponent(Component c) {

this.elements.add(c);

}

@Override

public void removeComponent(Component c) {

this.elements.remove(c);

}

@Override

public void doLayout() {

System.out.println("O Container utilizado é o FlowContainer");

System.out.println("Este container contêm bordas solidas");

System.out.println("Estes são os elementos presentes no container");

System.out.println(elements);

System.out.println("Usando o método dispose como herança para fechar o container");

this.dispose();

System.out.println("--------------------------------");

}

}

Obs.: o trecho de código destacado em laranja é idêntico ao que foi feito na classe FlowContainer.

Somente o método doLayout() foi modificado, acrescentando uma mensagem para identificar que se trata de um container com bordas sólidas.

4 – De forma semelhante crie agora a classe **FlowContainerBordaPontilhada** como subclasse da classe **Container**, e no método doLayout coloque a mensagem abaixo para identificar que se trata de um container com bordas pontilhadas.

System.out.println("Este container contêm bordas pontilhadas");

5 – O mesmo deve ser feito para criar as classes **GridContainerBordaSolida** e **GridContainerBordaPontilhada**. As duas novas classes também são subclasses da classe Container. Obs.: Lembre-se de identificar a borda no método doLayout com uma mensagem como foi anteriormente (itens 3 e 4).

6 – Faça os devidos testes usando a classe **TesteHerança** acrescentando containers com bordas pontilhadas e solidas

package roteiro7.parte2;

public class TesteHeranca {

public static void main(String[] args) {

Component button = new Component("Button");

Component textBox = new Component("TextBox");

Component textField = new Component("TextField");

Component checkBox = new Component("CheckBox");

Component combobox = new Component("ComboBox");

Component label = new Component("Label");

Component radiobutton = new Component("RadioButton");

// Criando um GridContainer para adicionar os elementos

Container c1 = new GridContainer(2, 2);

c1.addComponent(button); c1.addComponent(textBox);

c1.addComponent(textField); c1.addComponent(checkBox);

c1.doLayout();

System.out.println(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

// Criando um FlowContainer para adicionar os elementos

Container c2 = new FlowContainer();

c2.addComponent(combobox); c2.addComponent(label);

c2.addComponent(radiobutton);

c2.doLayout();

System.out.println(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

// Criando um FlowContainer com Pordas Pontilhadas para adicionar os elementos

Container c3 = new FlowContainerBordaPontilhada();

c3.addComponent(radiobutton); c3.addComponent(label);

c3.addComponent(combobox);

c3.doLayout();

System.out.println(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

// Criando um FlowContainer com Bordas Solidas para adicionar os elementos

Container c4 = new FlowContainerBordaSolida();

c4.addComponent(radiobutton); c4.addComponent(label);

c4.addComponent(combobox);

c4.doLayout();

}

}

7 – Utilize uma ferramenta de software qualquer para geração do diagrama de classes para esta etapa do projeto (Sugestão : Astah Community). Obs.: Adicione aqui o diagrama para que seja disponibilizado no teams

8 – Esta solução atendeu aos novos requisitos do projeto ? Existe alguma crítica a ser feita neste modelo ?

**Pacote : roteiro7.parte3**

Prosseguindo na evolução do projeto iremos utilizar melhor os recursos da herança na Orientação a Objetos.

Faremos com que as classes com bordas solidas ou pontilhadas herdem de **FlowContainer** e **GridContainer** e **não** de **Container**. Teremos assim dois níveis na hierarquia de classes.

1 - No mesmo projeto crie o pacote roteiro7.parte3

2 – Copie todas as classes criadas na parte2 para o novo pacote

3 – Faça as adaptações inicialmente na classe **FlowContainerBordaSolida** em que ela será subclasse da classe **FlowContainer**.

Observe que todo o código em duplicidade (atributos e métodos) foram eliminados da subclasse.

Para preservar o mesmo funcionamento da classe conforme a superclasse, usamos o comando **super()** no construtor da classe.

package roteiro7.parte3;

import java.util.ArrayList;

public class FlowContainerBordaSolida extends FlowContainer{

public FlowContainerBordaSolida() {

super();

}

@Override

public void doLayout() {

System.out.println("O Container utilizado é o FlowContainer");

System.out.println("Este container contêm bordas solidas");

System.out.println("Estes são os elementos presentes no container");

System.out.println(super.elements);

System.out.println("Usando o método dispose como herança para fechar o container");

this.dispose();

System.out.println("--------------------------------");

}

}

Da mesma forma usamos **super.elements** para acessar o atributo da superclasse

Obs.: o atributo **elements** foi declarado com a cláusula **private** na superclasse ela deve ser alterada para **protected**. Somente assim o atributo poderá ser acessado na subclasse (ou classe filha).

4 – O mesmo deve ser feito criando classe **FlowContainerBordaPontilhada** como subclasse da classe **FlowContainer**.

5 – As mesmas adaptações também devem ser feitas nas classes **GridContainerBordaSolida** e **GridContainerBordaPontilhada**.

6 - Faça os devidos testes usando a classe **TesteHerança**. Se a refatoração foi bem sucedida tudo deverá funcionar da mesma forma como antes.

7 – Utilize uma ferramenta de software qualquer para geração do diagrama de classes para esta etapa do projeto (Sugestão : Astah Community). Obs.: Adicione aqui o diagrama para que seja disponibilizado no teams

8 – As melhorias foram significativas? Ainda existe alguma crítica a ser feita neste modelo?

**Seguimos para o pacote2.parte4 !**

**Pacote : roteiro7.parte4**

Prosseguindo na evolução do projeto, percebe-se um excesso de classes criadas e uma má distribuição de responsabilidades. Neste caso, estamos nos referindo a funcionalidade/características inerentes a um container, e funcionalidade/características inerentes as bordas que podem estar em um container ou não.

1 - No mesmo projeto crie o pacote roteiro7.parte4

2 – Copie todas as classes criadas na parte3 para o novo pacote

3 – Conforme visto no roteiro anterior, este caso se encaixa bem no princípio de se programar para uma interface e não para uma implementação. Deve-se criar a interface **Borda** conforme segue abaixo, e nos próximos passos devemos criar classes concretas e específicas para cada classe.

package roteiro7.parte4;

public interface Borda {

public void gerarBorda();

}

4 – Crie as classes **BordaSolida** e **BordaPontilhada** de forma que implemente a interface Borda, e o método **gerarBorda()** deve apenas exibir a mensagem abaixo.

OU

System.out.println("Borda Pontilhada");

System.out.println("Borda Solida");

5 – Agora façamos refatoração para que um container contenha ou não uma borda, caracterizando assim uma associação entre as classes Container e Borda.

package roteiro7.parte4;

public abstract class Container {

protected Borda borda;

public Container() {

}

public Container(Borda borda) {

this.borda = borda;

}

public abstract void addComponent(Component c);

public abstract void removeComponent(Component c);

public abstract void doLayout();

public void dispose() {

System.out.println("Fechando o Container");

}

}

Observe que foi criado o atributo **borda** indicando a associação entre as classes. Ou seja, um container contém uma borda.

Vejam também que o atributo está com a cláusula **protected** indicando que o atributo está protegido, mas também acessível as subclasses.

Além disso devem ser criados 2 construtores. Um com passagem de parâmetros e o outro não, como opção para criação de um container com ou sem borda.

6 – Outra refatoração deve ser feita nas classes **FlowContainer** e **GridContainer**. Estas classes agora já poderão indicar se tem bordas ou não. Elas poderão também indicar qual o tipo de borda, uma vez que são subclasses de **Container,**  que por sua vez tem o atributo borda (item 5).

public class FlowContainer extends Container {

protected List<Component> elements;

public FlowContainer() {

this.elements = new ArrayList<Component>();

}

public FlowContainer(Borda borda) {

super(borda);

this.elements = new ArrayList<Component>();

}

…….

…….

…….

…….

}

}

public class GridContainer extends Container {

protected Component[][] elements;

private int lineCounter = 0;

private int columnCounter = 0;

private int ColumnMax;

private int lineMax;

public GridContainer(int lineMax, int ColumnMax) {

this.lineMax = lineMax;

this.ColumnMax = ColumnMax;

this.elements = new Component[lineMax][ColumnMax];

}

public GridContainer(int lineMax, int ColumnMax, Borda borda) {

this.lineMax = lineMax;

this.ColumnMax = ColumnMax;

super.borda = borda;

this.elements = new Component[lineMax][ColumnMax];

}

…….

…….

…….

…….

}

}

7 – Ainda nas classes **FlowContainer** e **GridContainer** é preciso adaptar a o método doLayout para indicar a existência e o tipo de borda, pois as classes **FlowContainerBordaSolida**, **FlowContainerBordaPontilhada**, **GridContainerBordaSolida**, e **GridContainerBordaPontilhada** serão eliminadas.

public void doLayout() {

System.out.println("O Container utilizado é o GridContainer");

if (this.borda != null)

this.borda.gerarBorda();

System.out.println("Estes são os elementos presentes no container");

System.out.println(Arrays.deepToString(elements));

System.out.println("Usando o método dispose como herança para fechar o container");

this.dispose();

System.out.println("--------------------------------");

}

public void doLayout() {

System.out.println("O Container utilizado é o FlowContainer");

if (this.borda != null)

this.borda.gerarBorda();

System.out.println("Estes são os elementos presentes no container");

System.out.println(elements);

System.out.println("Usando o método dispose como herança para fechar o container");

this.dispose();

System.out.println("--------------------------------");

}

8 – Depois de realizadas as refatorações não precisamos das classes **FlowContainerBordaSolida**, **FlowContainerBordaPontilhada**, **GridContainerBordaSolida**, e **GridContainerBordaPontilhada**. Estas classes já podem ser eliminadas.

9 – Para realizar os devidos testes é necessário um pequeno ajuste na classe TesteHeranca. Segue o trecho de código na criação do container c3 e c4.

// Criando um FlowContainer com Pordas Pontilhadas para adicionar os elementos

//Container c3 = new FlowContainerBordaPontilhada();

Container c3 = new FlowContainer(new BordaPontilhada());

c3.addComponent(radiobutton); c3.addComponent(label);

c3.addComponent(combobox);

c3.doLayout();

System.out.println(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

// Criando um FlowContainer com Bordas Solidas para adicionar os elementos

//Container c4 = new FlowContainerBordaSolida();

Container c4 = new FlowContainer(new BordaSolida());

c4.addComponent(radiobutton); c4.addComponent(label);

c4.addComponent(combobox);

c4.doLayout();

10 – Utilize uma ferramenta de software qualquer para geração do diagrama de classes para esta etapa do projeto (Sugestão : Astah Community). Obs.: Adicione aqui o diagrama para que seja disponibilizado no teams

11 – Quando se fala em **favorecer o uso de composição sobre herança**, também estamos falando intrinsecamente do SOLID. Avalie as técnicas aplicadas neste roteiro e identifique os princípios e boas práticas do SOLID. Justifique a sua resposta.

**OBS**.: Procure fazer a análise das 3 partes deste Roteiro