

- UESC - Universidade Estadual de Santa Cruz



Cap 5 – Interface Gráfica e Collections

Parte 1 – Interface Gráfica II

Disciplina: Linguagem de Programação III
Professor: Otacílio José Pereira

Plano de Aula

- **Objetivos**

- O foco agora é ver alguns passos e alguns conceitos relacionados com a manipulação de uma simples lista na interface

- **Tópicos**

- Passo 7: Criando um objeto
- Passo 8: Trabalhando com Containers
- Passo 9: Manipulando Lista de Clientes

A screenshot of a software window titled "Primeira Janela". It contains a form with two input fields labeled "Código" and "Nome". Below these fields are two buttons: "Salvar" and "Cancelar". Under the buttons, it says "Total clientes: 0". At the bottom, there is a section labeled "Lista de Clientes" which is currently empty.

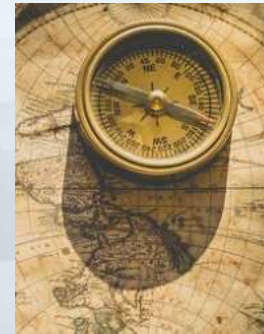


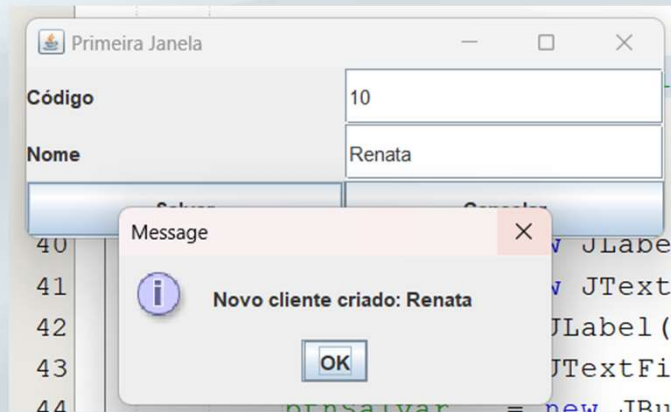
Contexto

- Onde estamos?
- Foco agora!
- Cenário de exemplo

Onde estamos?

- *Considerando nosso planejamento inicial*
- Capítulo 1 – Introdução
- Capítulo 2 - Conceitos básicos de Orientação a Objetos
- Capítulo 3 – Herança
 - Parte 1 – Herança
 - Parte 2 - Polimorfismo
- Capítulo 4 – Classes abstratas e interfaces
- **Foco** Capítulo 5 – Interface Gráfica e Collections
- Capítulo 6 – Desenvolvimento de um projeto em OO



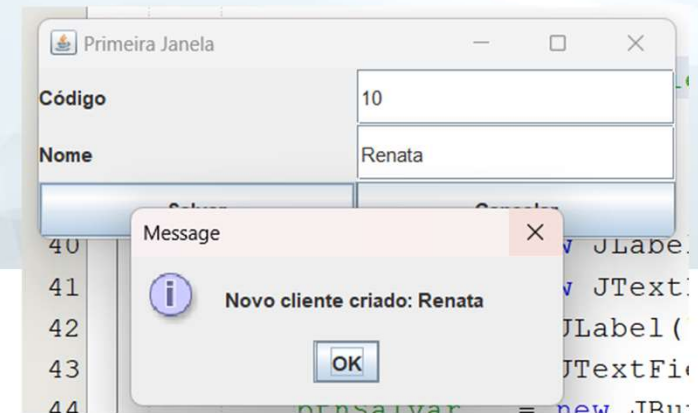


Passo 7: Manipulando Objeto Cliente

- Observe e execute o código
- Classe Cliente
- Método NovoCliente
- Evento do Botão Salvar
- Exercício

Observe e execute!

- Observe o exemplo ao lado
 - Qual a funcionalidade de nossa tela?
 - Como um novo cliente pode ser manipulado?



- C é uma variável local ou atributo?
- Onde ela deve estar declarada no código?
- Como os valores dos controles de interface chegaram até o objeto?

```
public void NovoCliente ()
{
    String strCodigo, strNome;
    int iCodigo;
    strCodigo = txtCodigo.getText();
    strNome    = txtNome.getText();
    iCodigo = Integer.parseInt(strCodigo);

    c = new Cliente(iCodigo, strNome);

    JOptionPane.showMessageDialog(rootPane, "Novo cliente criado: " +
                                    c.getNome());
}
```

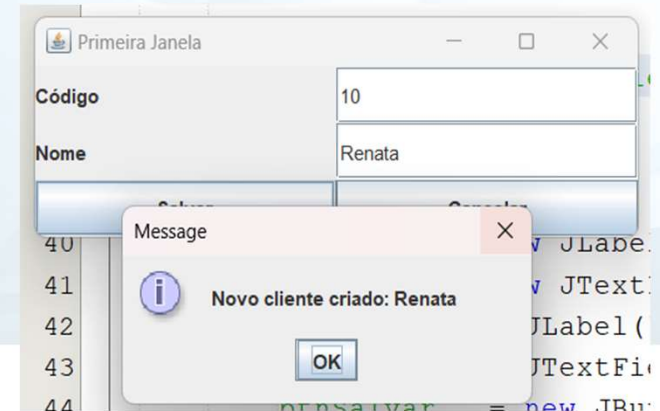
Tópicos

- As alterações a partir de agora não serão tão pontuais como anteriormente, algumas mudanças maiores serão necessárias e é importante organizar as partes
- Ajuste 1
 - Precisamos de ter uma classe Cliente, base dos objetos
- Ajuste 2
 - A janela precisa ter agora o atributo Cliente
- Ajuste 3
 - Um método na janela cria um novo cliente
- Ajuste 4: Precisamos chamar o método no evento do botão

```
public void NovoCliente ()
{
    String strCodigo, strNome;
    int iCodigo;
    strCodigo = txtCodigo.getText();
    strNome    = txtNome.getText();
    iCodigo = Integer.parseInt(strCodigo);

    c = new Cliente(iCodigo, strNome);

    JOptionPane.showMessageDialog(rootPane, "Novo cliente criado: " +
                                      c.getNome());
}
```



Classe Cliente

- Este é um dos elos com os conteúdos anteriores de Orientação a Objetos
- Aquelas classes e objetos que manipulávamos via console, agora pode ser manipulada pela janela
- Classe Cliente
 - Nos mesmos moldes que usávamos para nossas classes anteriores
- Curiosidade
 - Em um projeto normal ela seria uma classe a parte no projeto
 - No nosso caso ela foi criada internamente, uma classe abrigando outra, facilita eventual dinâmica da aula

```
public class Cliente
{
    private int codigo;
    private String nome;

    Cliente(int pCodigo, String pNome)
    {
        this.codigo = pCodigo;
        this.nome = pNome;
    }

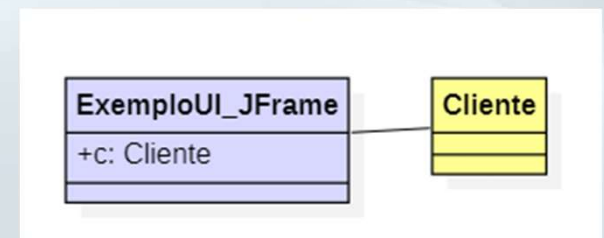
    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
}
```


Cliente: Atributo na Classe Janela

- Para que a nossa Janela possa manipular o cliente, ele passa a ser um atributo da Janela
- Isto é, uma Janela possui internamente um objeto que é um “espelho”, isto é, guarda as informações que são mostradas em tela
- Em alguma aula veremos um pouco de UML, ela pode nos ser útil neste momento, perceba temos duas classes e uma como atributo de outra

```
public class ExemploUI extends JFrame{  
  
    JLabel        lblCodigo, lblNome;  
    JTextField    txtCodigo, txtNome;  
    JButton        btnSalvar, btnCancelar;  
  
    Cliente c;  
  
    ExemploUI ()  
    {  
        lblCodigo = new JLabel("Código");  
        txtCodigo = new JTextField();  
        lblNome = new JLabel("Nome");  
        txtNome = new JTextField();  
    }  
}
```



Método NovoCliente

- Na tentativa de modularizar as ações na janela, foi criado um método novo cliente
- Ele é responsável por capturar os campos de tela e instanciar o novo cliente que está sendo manipulado
- Este método será invocado quando houver um clique do botão

```
public void NovoCliente ()  
{  
    String strCodigo, strNome;  
    int iCodigo;  
    strCodigo = txtCodigo.getText();  
    strNome    = txtNome.getText();  
    iCodigo = Integer.parseInt(strCodigo);  
  
    c = new Cliente(iCodigo, strNome);  
  
    JOptionPane.showMessageDialog(rootPane,  
        "Novo cliente criado: " +  
        c.getNome());  
}
```

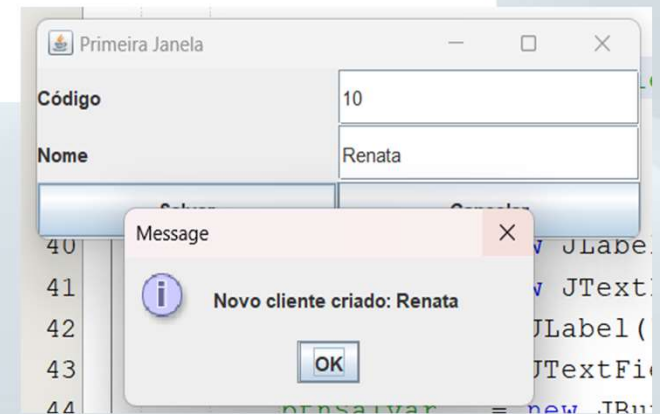
Evento do botão Novo

- Quando um evento é acionado, existe um objeto Evento que é criado e disparado para o Listener, no nosso caso a classe Handler
- Neste evento é possível capturar quem foi a fonte do evento, isto é, quem disparou
- Estamos usando esta forma de identificar o clique do botão Novo e chamada do método NovoCliente

```
private class EventoHandler implements ActionListener
{
    @Override
    public void actionPerformed(ActionEvent event)
    {
        String string = "";

        if (event.getSource() == btnSalvar)
        {
            NovoCliente();
        }

        else if (event.getSource() == btnCancelar)
        {
            JOptionPane.showMessageDialog(null, "Operação cancelada");
        }
    }
}
```



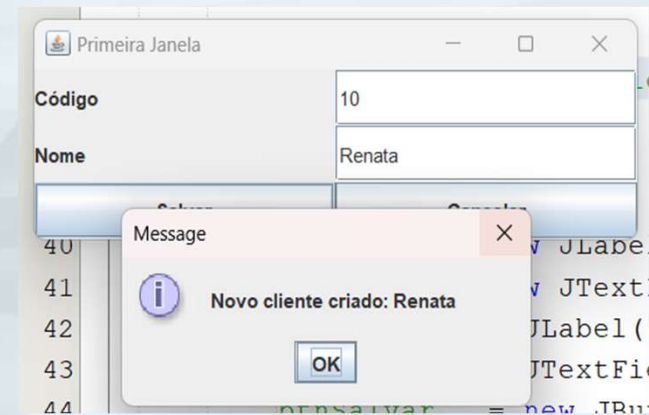
Exercício

- Exercício 1

- Experimente criar um método Cancelar na Janela que limpa os campos da Interface Gráfica quando o botão Cancelar é clicado
- Faça a chamada do método no clique do botão

- Exercício 2

- Faça uma pesquisa ou relembre uma breve explicação sobre tratamento de exceção e use para criticar quando o usuário tentar salvar um Cliente com código inválido.



getContentPane()

Primeira Janela

Código **formularioPane**

Nome

Salvar Cancelar

Total clientes: 0

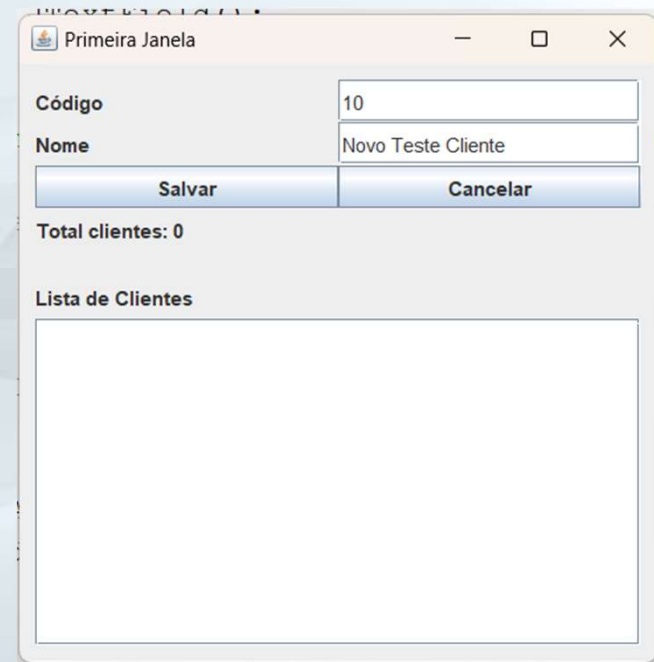
Lista de Clientes **listPane**

Passo 8: Containers

- Observe e execute o código
- Containers
- Estruturação de Containers
- Exercícios

Observe e execute 1!

- Observe o código a seguir!
 - O que há de novo em nossa janela?
 - É possível separar regiões em nossa tela?
 - Quais delas?
 - Dê uma olhada na paleta de controles do Netbeans, que controles poderiam servir para mostra a lista de clientes?



The screenshot shows a Java Swing window titled "Primeira Janela". It contains a form with two text input fields: "Código" with the value "10" and "Nome" with the value "Novo Teste Cliente". Below these fields are two buttons: "Salvar" and "Cancelar". Under the buttons, it says "Total clientes: 0". At the bottom, there is a section labeled "Lista de Clientes" followed by a large, empty rectangular area, likely intended for a list or table of client data.

Observe e execute 2!

- Observe agora este código!
 - Existe alguma correlação da variável/objeto formularioPane com as partes anteriores?
 - Agora, os controles lblCodigo, txtCodigo e outros estão sendo adicionados em que objeto?
 - O gerenciamento de layout está sendo feito em que controle?
 - O que você acha que é o contentPane?

```
JPanel formularioPane = new JPanel();  
// Gerenciador de Layout  
formularioPane.setLayout(new GridLayout(4, 2));  
// Uma borda em volta  
formularioPane.setBorder(BorderFactory.createEmbossedBorder());  
// Adicionando os controles  
formularioPane.add(lblCodigo);  
formularioPane.add(txtCodigo);  
formularioPane.add(lblNome);  
formularioPane.add(txtNome);  
formularioPane.add(btnSalvar);  
formularioPane.add(btnCancelar);  
formularioPane.add(lblNumeroClientes);
```

```
Container contentPane = getContentPane();  
contentPane.add(formularioPane, BorderLayout.NORTH);  
contentPane.add(listPane, BorderLayout.CENTER);
```

```
JPanel listPane = new JPanel();
```


Containers

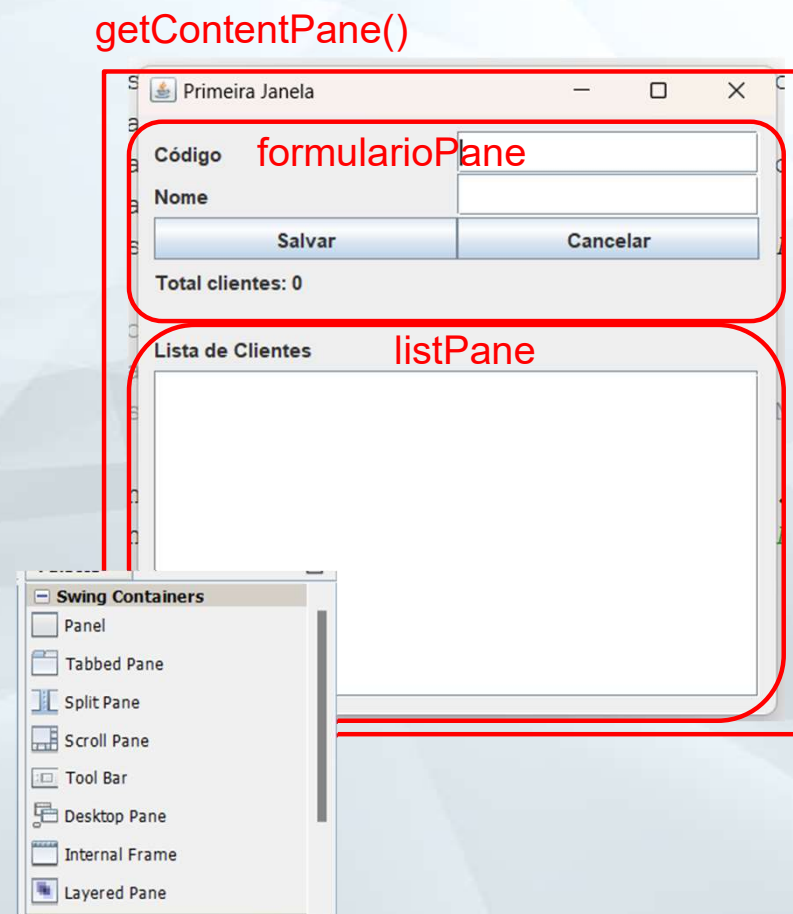
- Container
 - São tipos especiais de controles que podem conter outros controles
 - **JPanel** (Painel) é um exemplo
 - Eles ajudam a formar estruturas maiores nas interfaces
- **getContentPane** (de uma Janela)
 - Recupera o container, o “painel principal” da Janela onde são adicionados os elementos

```
// Estruturando a interface:  
// Parte 1 (Formulario) Parte 2 Lista de contas  
// **** Parte 1: Painel com o Formulário  
JPanel formularioPane = new JPanel();  
// Gerenciador de Layout  
formularioPane.setLayout(new GridLayout(6, 2));  
// ****  
formularioPane.add(lblNumero);  
formularioPane.add(lblAgencia);  
formularioPane.add(lblTitular);  
formularioPane.add(lblSaldo);  
formularioPane.add(btnNovo);
```

```
// **** Montando as partes maiores  
// O container da Janela (ContentPane) e  
// Os dois painéis com Formulário (superior-Norte) e  
Container contentPane = getContentPane();  
contentPane.add(formularioPane, BorderLayout.NORTH);  
contentPane.add(listPane, BorderLayout.CENTER);
```

Estruturação com os Containers

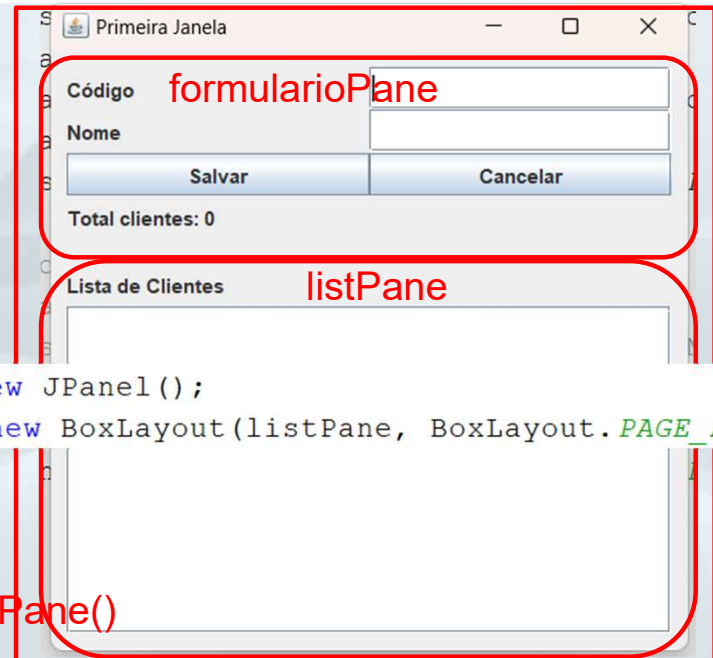
- Container
 - Reforçando, os containers ajudam a estruturar interfaces, organizando as várias partes.
 - Existem uma série de outros tipos de containers úteis, vale dar uma olha na paleta do Netbeans para observar as possibilidades
- Curiosidades
 - Você já trabalhou com interfaces Web? A analogia é com o conceito de div lá!
 - Em User Experience (UX) umas das tarefas é encontrar o melhor agrupamento de informações em interfaces, que tem a ver com esta estruturação de que estamos falando!



Containers e Gerenciamento de Layout

- Perceba que cada container, por possuir os seus controles precisa ter seu próprio Gerenciador de Layout
- Para o formulário, continuamos com o GridLayout para organizar os controles em grid
- Para a lista, usou-se o Box para controles na sequência
- E para tela como um todo, BorderLayout para usar o norte e sul.

```
JPanel formularioPane = new JPanel();  
// Gerenciador de Layout  
formularioPane.setLayout(new GridLayout(4, 2));
```

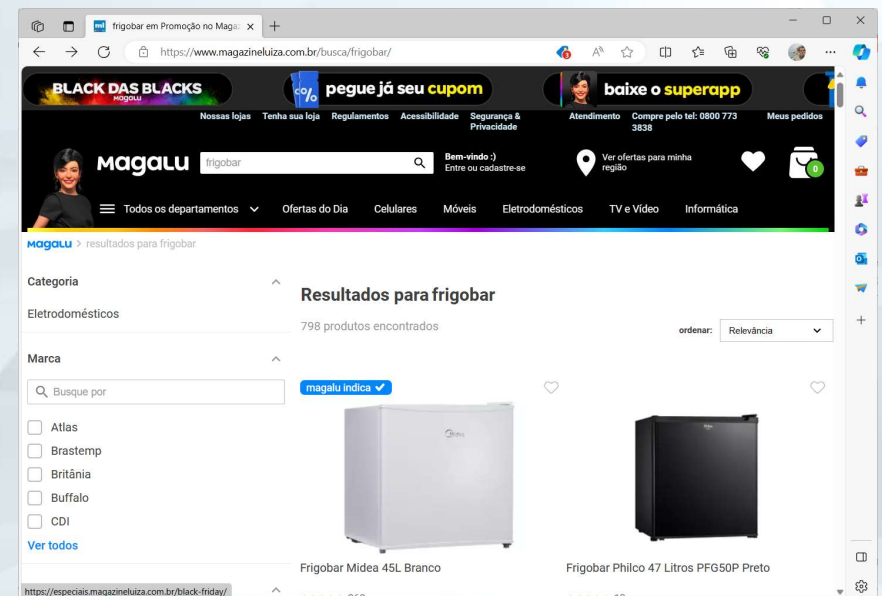


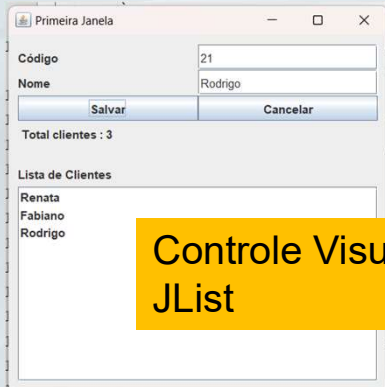
```
JPanel listPane = new JPanel();  
listPane.setLayout(new BoxLayout(listPane, BoxLayout.PAGE_AXIS));
```

```
Container contentPane = getContentPane();  
contentPane.add(formularioPane, BorderLayout.NORTH);  
contentPane.add(listPane, BorderLayout.CENTER);
```

Exercício

- Exercício 1:
 - Observe uma tela de um programa que você usa ou um site na Internet e identifique quais elementos você construiria com algo análogo a um container
- Exercício 2:
 - Experimente retirar um dos add do container Formulário ou Lista e verifique o que acontece com a interface.





Model

ArrayList:
IstClientes

Controle Visual:
JList

Passo 9: JList

- Observe e execute o código
- Jlist
- Conceito de Model e View
- Exercícios

Observe e execute!

- Por que o objeto Cliente foi comentado e o que foi colocado no lugar?
- Qual controle visual está sendo usado para visualizar a lista de clientes?
- Em sua criação, que elemento é necessário?
- Como este elemento é mantido no atualizar da lista?

```
JLabel      lblListaClientes;  
JList       jlstClientes;  
DefaultListModel mdlListaClientes;  
  
//Cliente c;  
ArrayList<Cliente> lstClientes = new ArrayList<Cliente>();
```

```
lblListaClientes = new JLabel("Lista de Clientes");  
mdlListaClientes = new DefaultListModel();  
jlstClientes = new JList(mdlListaClientes);
```

```
public void AtualizarLista()  
{  
    int i;  
    mdlListaClientes.clear();  
    for(Cliente aux : lstClientes)  
    {  
        // DefaultListModel atribuído ao JList  
        mdlListaClientes.addElement(aux.getNome());  
    }  
}
```


Passo 1: Compreendendo conceitos (Model)

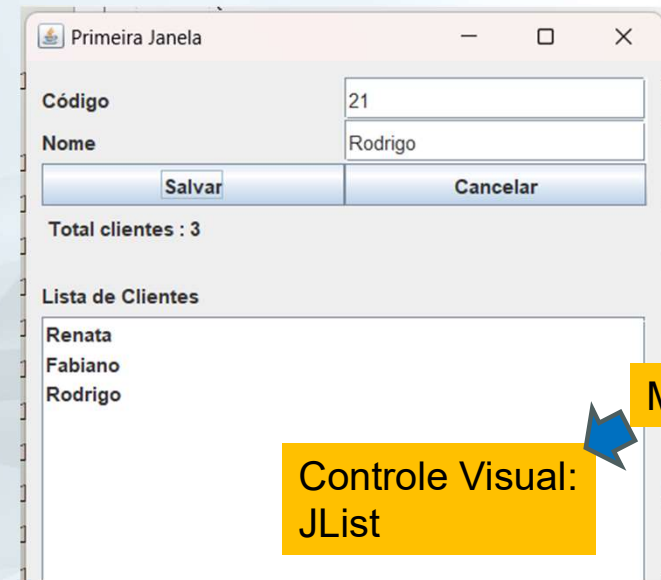
- Controles mais complexos
 - Controles mais complexos, como o JList, JTable e outros manipulam mais do que apenas um texto (como JTextField)
 - Eles possuem um elemento auxiliar chamado de Model
 - O model contém o “teor” (recheio), o conteúdo do que é mostrado visualmente
 - Modificando este Model o controle visual é atualizado

Método MontarInterface()

```
lblListaClientes = new JLabel("Lista de Clientes");  
mdlListaClientes = new DefaultListModel();  
jlstClientes = new JList(mdlListaClientes);
```


Passo 1: Compreendendo conceitos (Model)

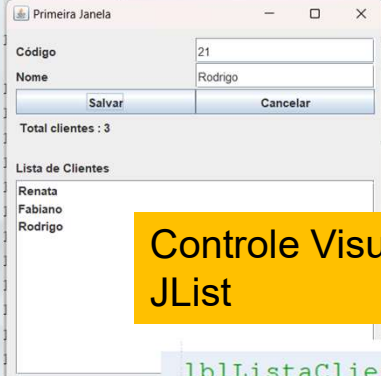
- Model em controles mais complexos
 - Controles mais complexos, como o JList, JTable e outros manipulam mais do que apenas um texto (como JTextField)
 - Eles possuem um elemento auxiliar chamado de Model
 - O model contém o “teor” (recheio), o conteúdo do que é mostrado visualmente
 - Modificando este Model o controle visual é atualizado



```
lblListaClientes = new JLabel("Lista de Clientes");  
mdlListaClientes = new DefaultListModel();  
jlstClientes = new JList(mdlListaClientes);
```

Passo 2: Aquecendo com collections (Model)

- Conteúdo do Model
 - Se o Model, no caso o nosso mdlListaClientes serve de conteúdo para o controle visual JList
 - O que serve de base para compor o conteúdo do Model?
 - Neste caso, criamos um ArrayList, que guarda uma lista de clientes
 - Ele que serve para adicionar os elementos no Model que por sua vez serão atualizados no JList



```
lblListaClientes = new JLabel("Lista de Clientes");
mdlListaClientes = new DefaultListModel();
jlstClientes = new JList(mdlListaClientes);
```

```
public void AtualizarLista()
{
    int i;
    mdlListaClientes.clear();
    for(Cliente aux : lstClientes)
    {
        // DefaultListModel atribuído ao JList
        mdlListaClientes.addElement(aux.getNome());
    }
}
```

Exercícios

- Exercício 1
 - Observe na paleta de controles ou pesquise sobre quais controles possuem o elemento Model
 - Pesquise trechos de códigos deste componente e como o Model é preenchido e acoplado ao controle visual
- Exercício 2
 - O conceito de Model vem de uma estrutura MVC (Model-View-Controller) com ênfase para o View e o Model
 - Cuidado, MVC é também usado para arquiteturas de software mais complexas, aqui estamos falando apenas de controles visuais

Primeira Janela

Código: 21

Nome: Rodrigo

Salvar Cancelar

Total clientes : 3

Lista de Clientes

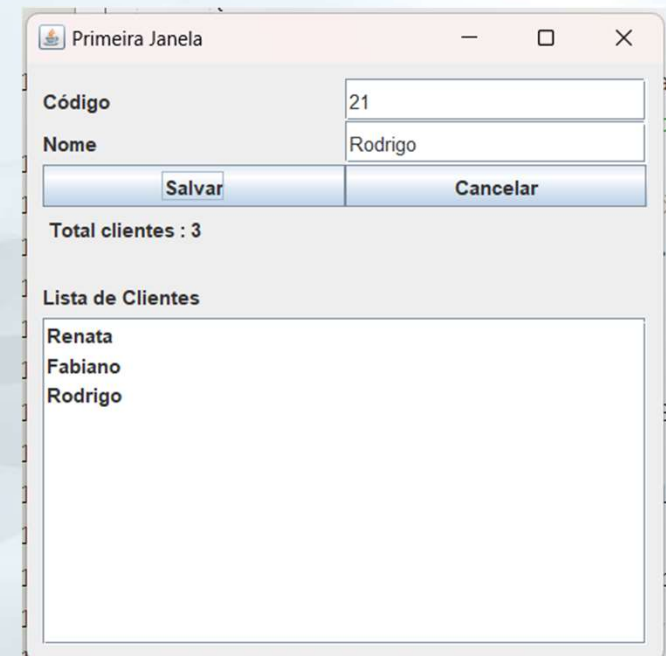
- Renata
- Fabiano
- Rodrigo

Considerações Finais

- Assunto mais extenso
- Outras bibliotecas
- Orientação a Objetos

Observações: Assunto amplo

- O assunto de interface gráfica é bem mais extenso, o foco deste tópico dentro desta disciplina é de apresentar alguns conceitos de interface
 - IU, Janelas, Controles, Eventos, Listener, Containers além de outros
- Há muito mais detalhes e macetes de como implementar
- E com este assunto perceber como o conceito de orientação a objetos é aplicado
 - Perceba que uma Janela herda de JFrame e por si é um objeto
 - Por sua vez, ela contém uma série de controles (objetos)
 - Com seus atributos e métodos
 - E tudo isso fica empacotado e disponível em uma biblioteca



Observações: Outras bibliotecas

- Usamos como base a biblioteca Swing, que é uma biblioteca tradicional e explorada em diversas fontes como o livro de Deitel e outros
- Ela nos foi útil para entendermos os conceitos comentados
- Existem outros meios de se produzir a interface, conforme nosso tempo na disciplina podemos explorar

Primeira Janela

Código: 21

Nome: Rodrigo

Salvar Cancelar

Total clientes : 3

Lista de Clientes

- Renata
- Fabiano
- Rodrigo



Conclusões

- Visão Geral dos Passos
- Para saber mais

Visão Geral

- *Passo 1: Aquecendo com JOptionPane*
- *Passo 2: Primeira Janela*
- *Passo 3: Controles Visuais*
- *Passo 4: Gerenciamento de Layout*
- *Passo 5: Outros tipos de controles*
- *Passo 6: Eventos, Listeners e outros*
- *Passo 7: Integrando com um Objeto*
- *Passo 8: Estruturando a interface com Containers*
- *Passo 9: Trabalhando com JList e o conceito de Model*

