

FCT/Unesp – Presidente Prudente
Departamento de Matemática e Computação

**Programação Orientada
a Objetos**
Interface Gráfica
Parte IV

Prof. Dr. Danilo Medeiros Eler
danilo.eler@unesp.br

Aula de Hoje

- Interface de Usuário (IU)
 - *User Interface* (UI)
 - Interface para Vendas
 - Interface para Localização de Clientes
 - Outros Componentes
 - JTable

Cadastro de Vendas

Cadastro de Vendas

The image shows a screenshot of an IDE (likely NetBeans) with a Java Swing window titled "Cadastro de Vendas". The window contains a form with two sections: "Cabecalho" and "Item".

Cabecalho Section:

- Fields: "Codigo Venda" and "Data" (top row); "CPF" and "Nome" (bottom row).

Item Section:

- Fields: "Codigo Produto" and "Descricao" (top row); "Valor" and "Quantidade" (bottom row).
- Button: "Adicionar" (bottom right).

Two red arrows point from the "Controles Swing" list in the palette to the "Cabecalho" and "Item" sections of the form.

Component Palette (Controles Swing):

- Panel
- Dividir painel
- Barra de ferramentas
- Quadro interno
- Controles Swing**
- label Rótulo
- Botão Alternar
- Botão de opção
- Caixa de combinação
- Campo de texto
- Barra de rolagem
- Barra de progresso

Properties Window (jPanel1 [JPanel] - Propriedad):

- Propriedades Vinculação
- Propriedades
- background
- border
- foreground
- toolTipText
- Outras propriedades
- UIClassID
- alignmentY
- jPanel1 [JPanel]

Cadastro de Vendas

Propriedades Vinculação Eventos Código

Propriedades		
background	<input type="checkbox"/> [240,240,240]	...
border	(Sem borda)	...
foreground	<input checked="" type="checkbox"/> [0,0,0]	...
toolTipText	null	...
Outras propriedades		
UIClassID	PanelUI	...
alignmentX	0.5	...
alignmentY	0.5	...
autoscrolls	<input type="checkbox"/>	...
baselineResizeBehavior	OTHER	...
componentPopupMenu	<nenhum>	...
cursor	Default Cursor	...
debugGraphicsOptions	NO_CHANGES	...
doubleBuffered	<input type="checkbox"/>	...
enabled	<input checked="" type="checkbox"/>	...
focusCycleRoot	<input type="checkbox"/>	...
focusTraversalPolicy	<nenhum>	...
focusTraversalPolicyProvider	<input type="checkbox"/>	...
focusable	<input checked="" type="checkbox"/>	...
font	Tahoma 11 Plano	...
inheritsPopupMenu	<input type="checkbox"/>	...

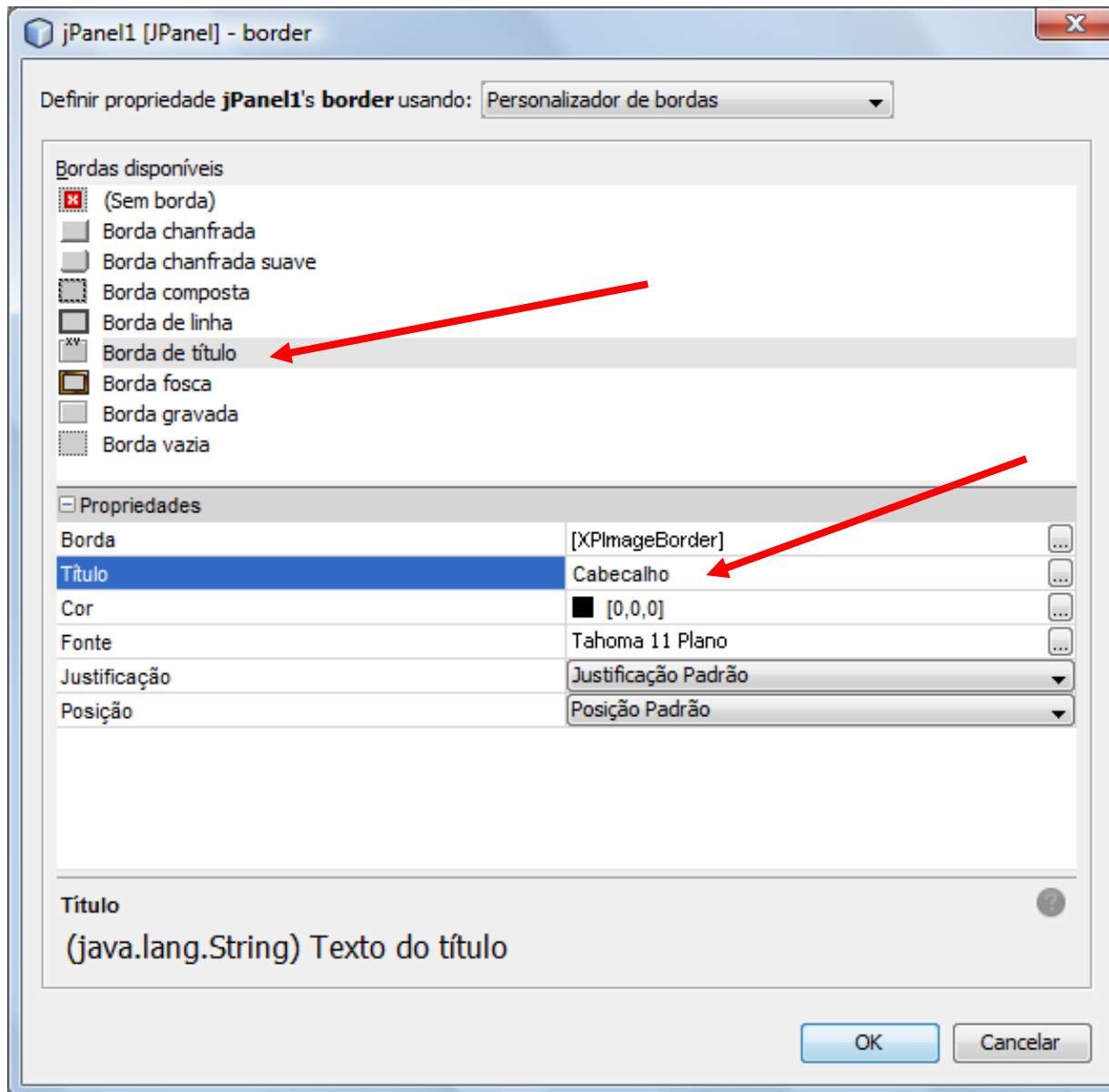
border
(javax.swing.border.Border) The component's border.

Fechar

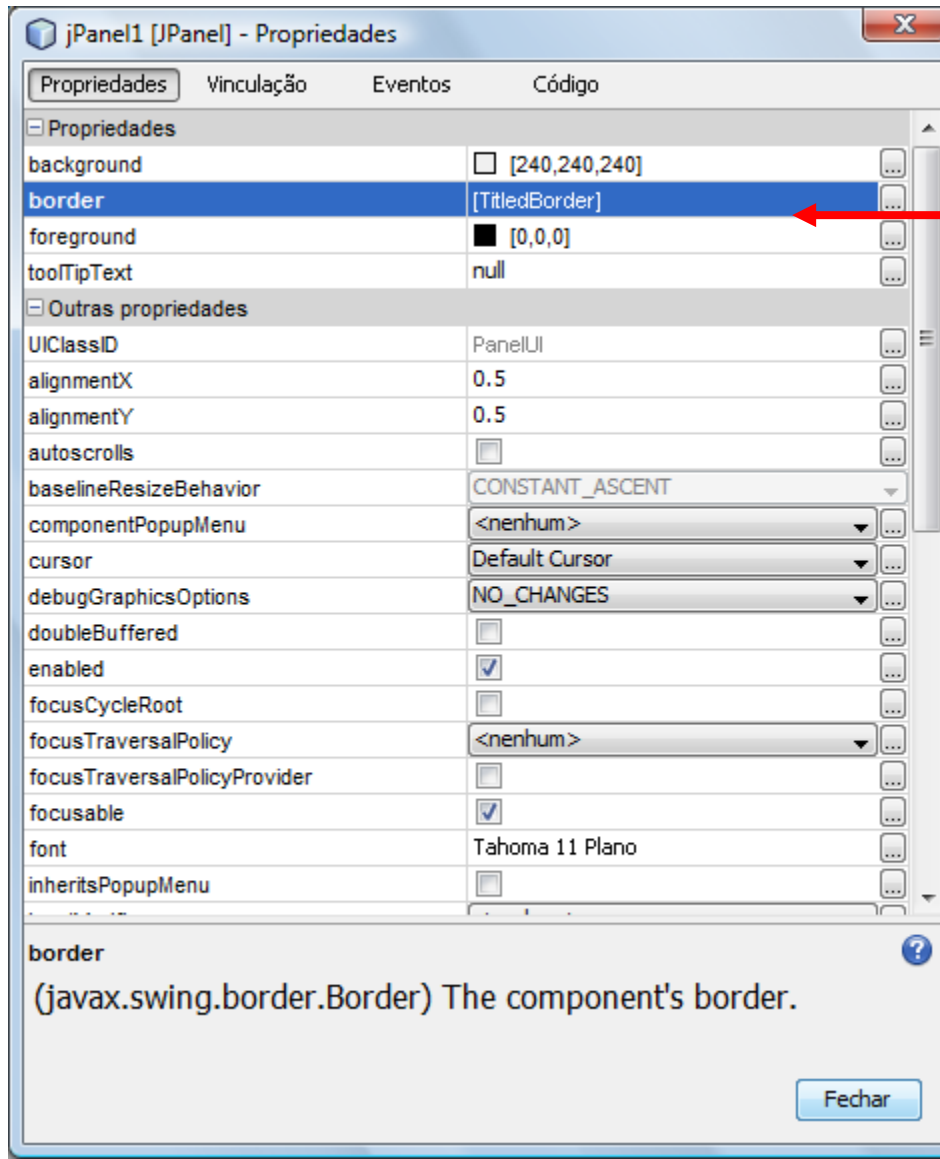
run:
CONSTRUÍDO COM SUCESSO (tempo total: 8 segundos)

Modificar o estilo de borda

Cadastro de Vendas



Cadastro de Vendas



Note que mudou o tipo de borda

Cadastro de Venda

Cabecalho

Codigo Venda

Data

CPF

Nome

Item

Codigo Produto

Descricao

Valor

Quantidade

Adicionar

Cadastro de Venda

- Adicionaremos mais um panel (JPanel) para organizar a exibição dos itens
 - A configuração da borda é igual ao que foi efetuado anteriormente

Cadastro de Vendas

Cabecalho

Codigo Venda

Data

CPF

Nome

Item

Codigo Produto

Descricao

Valor

Quantidade

Adicionar

Itens da Venda

Codigo Produto	Descricao	Valor	Quantidade

Palette x

Swing Containers

Panel

Tabbed Pane

Split Pane

ToolBar

Desktop Pane

Internal Frame

Swing Controls

Label

Button

Toggle Button

Radio Button

Button Group

Combo Box

Text Field

Text Area

Scroll Bar

Progress Bar

Formatted Field

Password Field

Separator

Text Pane

Editor Pane

Table

Swing Menus

Swing Windows

Swing Fillers

AWT

Beans

Beans Properties x

Properties

Events

Code

Properties

defaultCloseOperation

DISPOSE

title

Other Properties

alwaysOnTop

alwaysOnTopSupported

autoRequestFocus

background

[240,240,240]

bounds

<Not Set>

cursor

Cursor Padrão

enabled

focusCycleRoot

[JDialog]

Cadastro de Vendas

- Colocaremos uma tabela (JTable) dentro do panel (JPanel) e adicionaremos itens à tabela
 - Clicando com o botão direito sobre o panel e adicionando um componente da paleta, no caso, uma Tabela (JTable)

Cadastro de Vendas

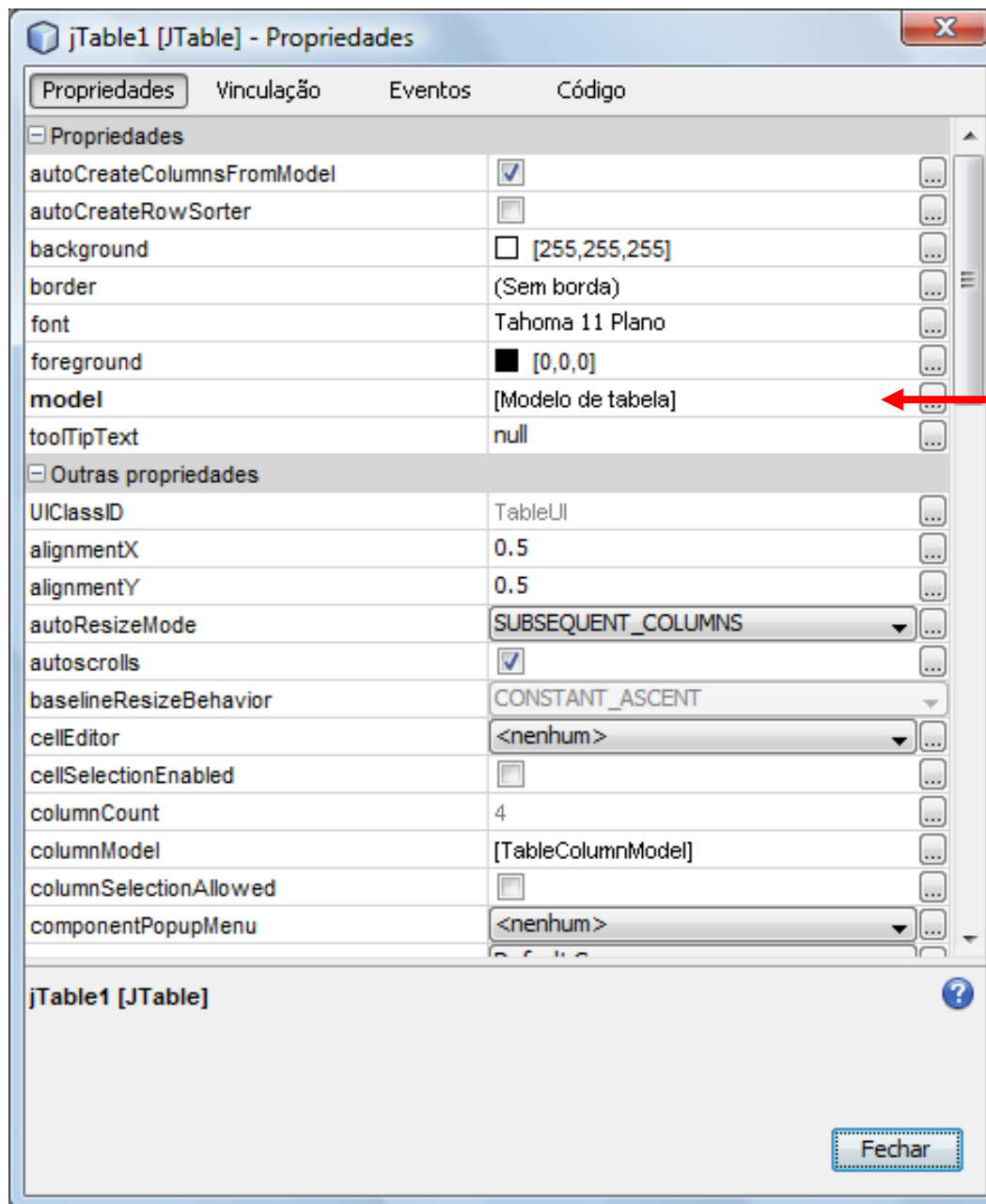
The image shows a Java Swing IDE interface. On the left, there is a form titled 'Item' with four input fields: 'Codigo Produto', 'Descricao', 'Valor', and 'Quantidade'. Below this is a section titled 'Itens da Venda' which is currently empty. A context menu is open over the 'Itens da Venda' area, listing various actions like 'Adicionar da paleta', 'Desenhar este recipiente', and 'Alinhar'. The 'Controles Swing' option is selected, and a sub-menu is visible showing 'Tabela' as the chosen component. A red arrow points from the 'Tabela' option in the sub-menu to the 'Itens da Venda' area. On the right side, a properties window is partially visible, showing attributes like 'border', 'foreground', and 'horizontalScrollBarPolicy' for a 'JScrollPane1 [JScrollPane]' component.

Cadastro de Vendas

- A tabela vem pré-configurada
- Teremos que adicionar os campos que desejamos
 - Uma maneira é por meio das propriedades

Itens da Venda

Title 1	Title 2	Title 3	Title 4



Definir propriedade jTable1's model usando: Personalizador de modelo de tabela

Modelo de tabela

Configurações da tabela | Valores padrão

Especificar tipos de título e coluna aqui:

Coluna	Título	Tipo	Editável
1	Codigo	String	<input type="checkbox"/>
2	Descricao	String	<input type="checkbox"/>
3	Valor Unitario	Float	<input type="checkbox"/>
4	Quantidade	Float	<input type="checkbox"/>
5	Subtotal	Float	<input type="checkbox"/>

Inserir

Excluir

Mover para cima

Mover para baixo

Linhas: 4 + - Colunas: 5 + -

OK

Redefinir para padrão

Cancelar

Definir propriedade jTable1's model usando: Personalizador de modelo de tabela

Modelo de tabela

Configurações da tabela | **Valores padrão**

Especificar tipos de título e coluna aqui:

Coluna	Título	Tipo	Editável
1	Codigo	String	<input type="checkbox"/>
2	Descricao	String	<input type="checkbox"/>
3	Valor Unitario	Float	<input type="checkbox"/>
4	Quantidade	Float	<input type="checkbox"/>
5	Subtotal	Float	<input type="checkbox"/>

Inserir

Excluir

Mover para cima

Mover para baixo

Linhas: 4 + - Colunas: 5 + -

OK

Redefinir para padrão

Cancelar

Definir propriedade **jTable1's model** usando: Personalizador de modelo de tabela

Modelo de tabela

Configurações da tabela | Valores padrão

Valores padrão da tabela:

Codigo	Descricao	Valor Unitario	Quantidade	Subtotal

Colunas:

Inserir

Excluir

Mover à esquerda

Mover à direita

Linhas:

Inserir

Excluir

Mover para cima

Mover para baixo

Linhas: 4 + - Colunas: 5 + -

OK

Redefinir para padrão

Cancelar

Definir propriedade **jTable1's model** usando: Personalizador de modelo de tabela

Modelo de tabela

Configurações da tabela | Valores padrão

Valores padrão da tabela:

Codigo	Descricao	Valor Unitario	Quantidade	Subtotal
--------	-----------	----------------	------------	----------

Colunas:

Inserir

Excluir

Mover à esquerda

Mover à direita

Linhas:

Inserir

Excluir

Mover para cima

Mover para baixo

Linhas: 0 | + - Colunas: 5 | + -

OK

Redefinir para padrão

Cancelar

Cadastro de Vendas

- Outra maneira de alterar os campos da tabela é via código
 - Para isso, criaremos um modelo de tabela do tipo `DefaultTableModel`, para termos outras funcionalidades de manipulação da tabela
 - Esse modelo será usado em outras partes do código. Por isso, é um atributo da classe

```
public class IUVenda extends javax.swing.JDialog {  
    private DefaultTableModel model;  
    private Controlador control;  
    private float total;  
    public IUVenda(java.awt.Frame parent, boolean modal) {  
        super(parent, modal);  
        initComponents();  
        control = new Controlador();  
  
        String colunas[] = {"Cod", "Nome", "Valor", "Quantidade", "SubTotal"};  
        model = new DefaultTableModel(colunas, 0);  
        tabela.setModel(model);  
  
        textDescricao.setEditable(false);  
        textValor.setEditable(false);  
        textTotal.setEditable(false);  
  
        total = 0;  
    }  
}
```

Cadastro de Vendas

- Os itens são adicionados na tabela por meio de instruções
- Para isso, uma linha é criada (um vetor) e é adicionada à tabela
- Todas as instruções são executadas no clique do botão Adicionar

Cadastro de Vendas

Cabecalho

Codigo Venda: Data:

CPF: Nome:

Item

Codigo Produto: Descricao:

Valor: Quantidade:

Itens da Venda

Codigo	Descricao	Valor Unitario	Quantidade	Subtotal

Total:

```
private void add_ButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    String codigo = codProduto_Text.getText();  
    String descricao = descProduto_Text.getText();  
    float valor = Float.parseFloat(valorProduto_Text.getText());  
    float qtd = Float.parseFloat(qtd_Text.getText());  
    float subTotal = valor * qtd;  
  
    Object[] linha = new Object[5];  
    linha[0] = codigo;  
    linha[1] = descricao;  
    linha[2] = valor;  
    linha[3] = qtd;  
    linha[4] = subTotal;  
  
    model.addRow(linha);  
    float total = Float.parseFloat(total_Text.getText());  
    total = total + subTotal;  
    total_Text.setText(Float.toString(total));  
}
```

```
private void add_ButtonActionPerformed(java.awt.event.ActionEvent evt) {
```

```
String codigo = codProduto_Text.getText();  
String descricao = descProduto_Text.getText();  
float valor = Float.parseFloat(valorProduto_Text.getText());  
float qtd = Float.parseFloat(qtd_Text.getText());  
float subTotal = valor * qtd;
```

```
Object[] linha = new Object[5];  
linha[0] = codigo;  
linha[1] = descricao;  
linha[2] = valor;  
linha[3] = qtd;  
linha[4] = subTotal;
```

```
model.addRow(linha);  
float total = Float.parseFloat(total_Text.getText());  
total = total + subTotal;  
total_Text.setText(Float.toString(total));
```

```
}
```

Obtem os dados dos JText e
calcula o subtotal

Efetuar essa etapa se o modelo não for criado por linhas de comando ou se ele não for obtido no construtor (armazenado em um atributo da classe)

```
float subTotal = valor * qtd;
```

```
DefaultTableModel model = (DefaultTableModel) itens_Table.getModel();
```

```
Object[] linha = new Object[5];
```

```
linha[0] = codigo;
```

```
linha[1] = descricao;
```

```
linha[2] = valor;
```

```
linha[3] = qtd;
```

```
linha[4] = subTotal;
```

Obtem o modelo de dados da JTable para poder adicionar uma linha nele

```
model.addRow(linha);
```

```
float total = Float.parseFloat(total_Text.getText());
```

```
total = total + subTotal;
```

```
total_Text.setText(Float.toString(total));
```

```
}
```

```
private void add_ButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    String codigo = codProduto_Text.getText();  
    String descricao = de  
    float valor = Float.  
    float qtd = Float.pa  
    float subTotal = val
```

Cria um vetor com o número de colunas da tabela.
Cada coluna representa uma célula da tabela.
O vetor é do tipo Object para poder receber qualquer tipo de objeto, pois Object é a classe base do java.

```
Object[] linha = new Object[5];  
linha[0] = codigo;  
linha[1] = descricao;  
linha[2] = valor;  
linha[3] = qtd;  
linha[4] = subTotal;
```

```
model.addRow(linha);  
float total = Float.parseFloat(total_Text.getText());  
total = total + subTotal;  
total_Text.setText(Float.toString(total));
```

```
}
```

```
private void add_ButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    String codigo = codProduto_Text.getText();  
    String descricao = descProduto_Text.getText();  
    float valor = Float.parseFloat(valorProduto_Text.getText());  
    float qtd = Float.parseFloat(qtd_Text.getText());  
    float subTotal = valor * qtd;  
  
    DefaultTableModel model = (DefaultTableModel) itens_Table.getModel();  
    Object[] linha = new Object[5];  
    linha[0] = codigo;  
    linha[1] = descricao;  
    linha[2] = valor;  
    linha[3] = qtd;  
    linha[4] = subTotal;  
    model.addRow(linha);  
    float total = Float.parseFloat(total_Text.getText());  
    total = total + subTotal;  
    total_Text.setText(Float.toString(total));  
}
```

A linha é adicionada à tabela

model.addRow(linha);

```

private void add_ButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String codigo = codProduto_Text.getText();
    String descricao = descProduto_Text.getText();
    float valor = Float.parseFloat(valorProduto_Text.getText());
    float qtd = Float.parseFloat(qtd_Text.getText());
    float subTotal = valor * qtd;

    DefaultTableModel model = (DefaultTableModel) itens_Table.getModel();
    Object[] linha = new Object[5];
    linha[0] = codigo;
    linha[1] = descricao;
    linha[2] = valor;
    linha[3] = qtd;
    linha[4] = subTotal;

    model.addRow(linha);

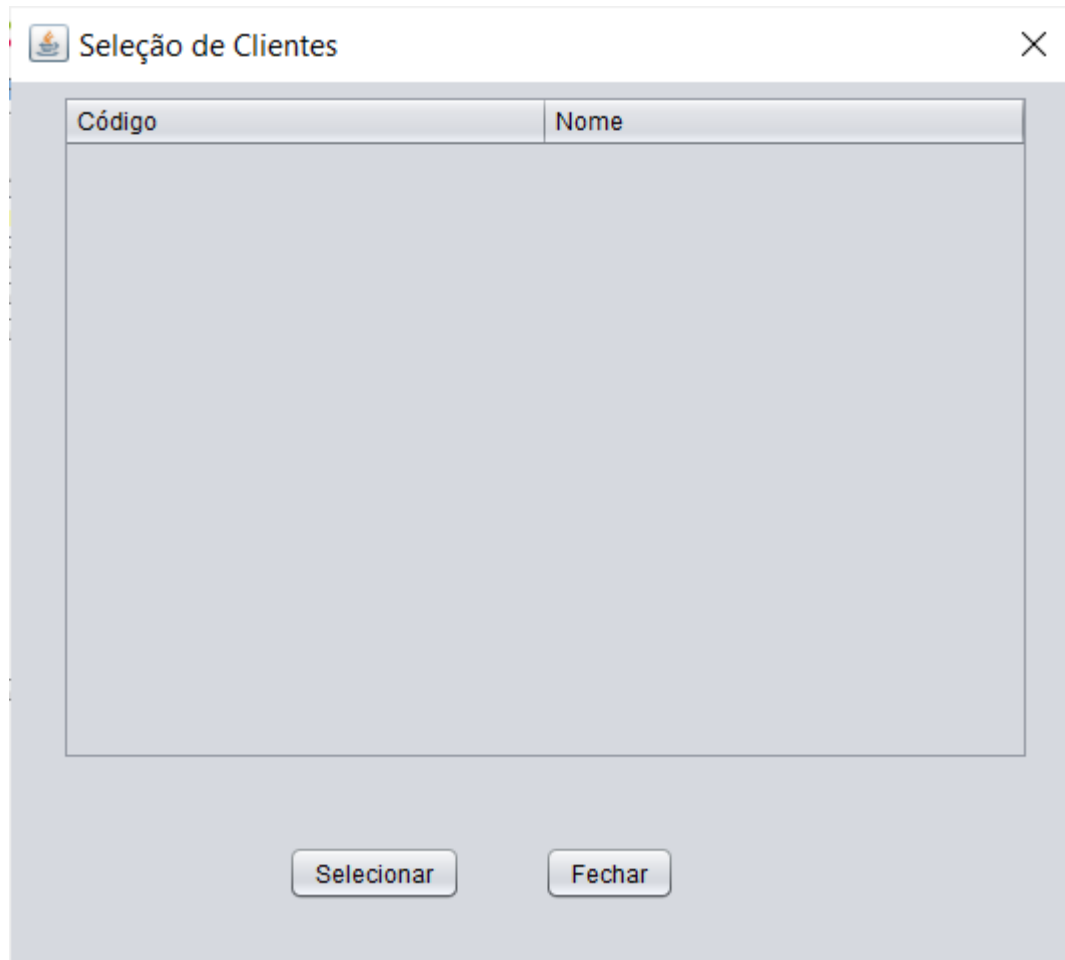
    float total = Float.parseFloat(total_Text.getText());
    total = total + subTotal;
    total_Text.setText(Float.toString(total));
}

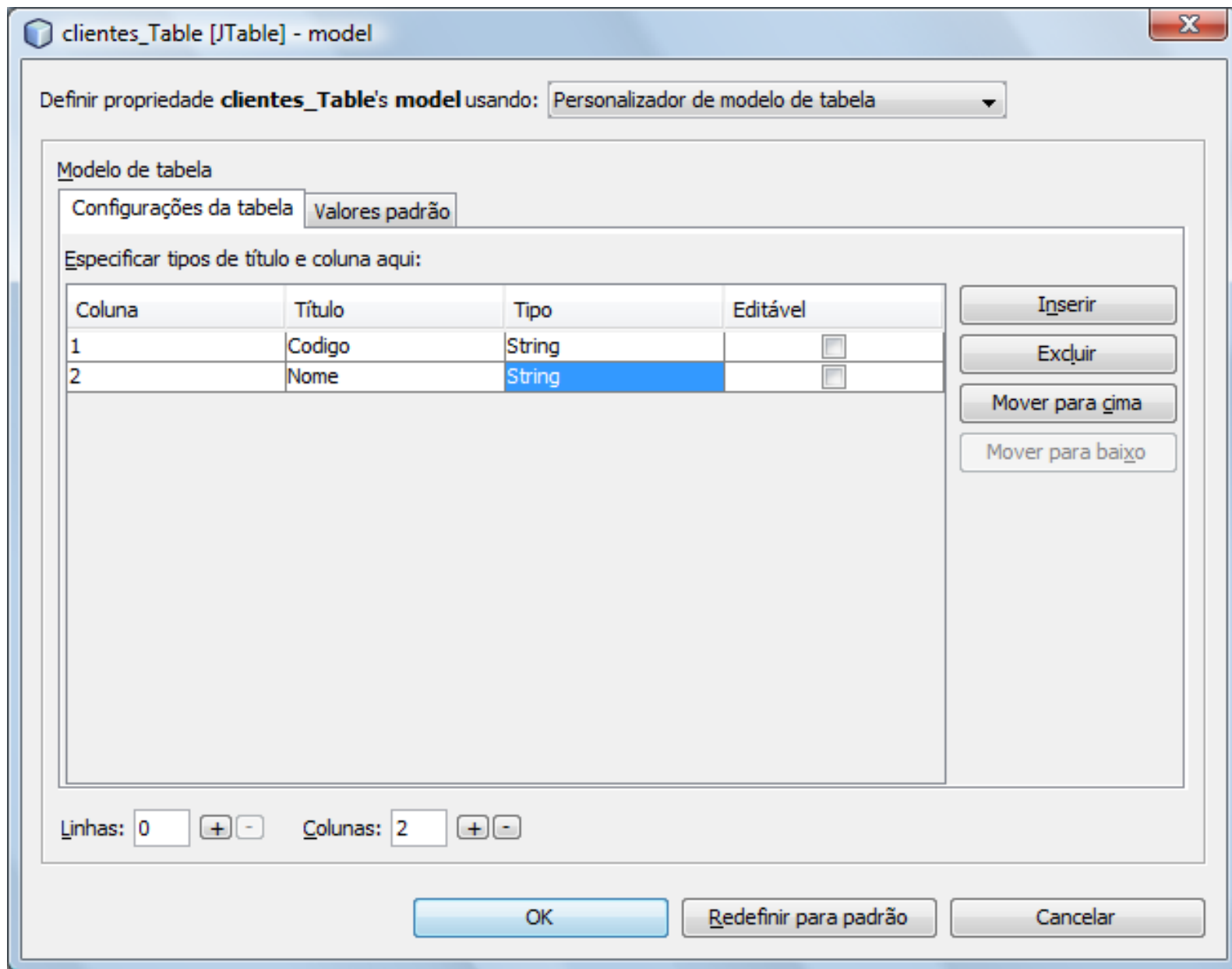
```

O total geral é calculado e atribuído
ao JText que exibe o total

Localizar Clientes

Localizar Clientes





Localizar Clientes

- Devemos adicionar os clientes à tabela sempre que o objeto dessa interface for criado
 - Podemos fazê-lo no construtor


```

public class IULocalizaCliente extends javax.swing.JDialog {
    private DefaultTableModel model;
    private Controlador control;
    private String cpf;
    public IULocalizaCliente(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
        cpf = null;
        String colunas[] = {"Código", "Nome"};
        model = new DefaultTableModel(colunas, 0);
        tabela.setModel(model);
        control = new Controlador();
        Cliente clientes[] = control.getClientes();
        for(int i=0; i<clientes.length; i++){
            if (clientes[i] != null){
                String linha[] = new String[2];
                linha[0] = clientes[i].getCPF();
                linha[1] = clientes[i].getNome();
                model.addRow(linha);
            }
        }
    }
}

```

```

public class IULocalizaCliente extends javax.swing.JDialog {
    private DefaultTableModel model;
    private Controlador control;
    private String cpf;
    public IULocalizaCliente(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
        cpf = null;
        String colunas[] = {"Código", "Nome"};
        model = new DefaultTableModel(colunas, new Object[0]);
        tabela.setModel(model);
        control = new Controlador();
        Cliente clientes[] = control.getClientes();
        for(int i=0; i<clientes.length; i++){
            if (clientes[i] != null){
                String linha[] = new String[2];
                linha[0] = clientes[i].getCPF();
                linha[1] = clientes[i].getNome();
                model.addRow(linha);
            }
        }
    }
}

```

Criamos um atributo para armazenar o CPF do cliente selecionado

```

public class IULocalizaCliente extends javax.swing.JDialog {
    private DefaultTableModel model;
    private Controlador control;
    private String cpf;
    public IULocalizaCliente(javax.swing.JFrame parent, boolean modal) {
        super(parent, modal);
        initComponents();
        cpf = null;
        String colunas[] = {"Código", "Nome"};
        model = new DefaultTableModel(colunas, 0);
        tabela.setModel(model);
        control = new Controlador();
        Cliente clientes[] = control.getClientes();
        for(int i=0; i<clientes.length; i++){
            if (clientes[i] != null){
                String linha[] = new String[2];
                linha[0] = clientes[i].getCPF();
                linha[1] = clientes[i].getNome();
                model.addRow(linha);
            }
        }
    }
}

```

Devemos percorrer o vetor de clientes e exibir os dados na tabela criada

Localizar Clientes

Cadastro de Vendas

Cabecalho

Codigo Venda: Data:

CPF: Nome:

Item


Codigo Produto: Descricao:

Valor: Quantidade:

Itens da Venda

Codigo	Descricao	Valor Unitario	Quantidade	Subtotal
--------	-----------	----------------	------------	----------

Total:





Cabecalho

Codigo Venda

Data



Seleção de Clientes



Código	Nome
11111	Cliente 1
22222	Cliente 2
33333	Cliente 3
44444	Cliente 4
55555	Cliente 5

Localizar

Subtotal

Selecionar

Fechar

Total

0

Localizar Clientes

- Quando o usuário fizer uma seleção e clicar no botão Selecionar, devemos fechar a janela e armazenar o CPF, que será recuperado na interface de vendas

```
private void selecionarButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    int linha = tabela.getSelectedRow();  
    if (linha >=0){  
        cpf = (String) model.getValueAt(linha,0);  
        setVisible(false);  
    }else{  
        JOptionPane.showMessageDialog(null, "Selecione um Cliente");  
    }  
}
```

Localizar Clientes

- Quando o usuário fizer uma seleção e clicar no botão Selecionar, devemos fechar a janela e armazenar o CPF, que será recuperado na interface de vendas

```
private void selecionarButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    int linha = tabela.getSelectedRow();  
    if (linha >=0){  
        cpf = (String) model.getValueAt(linha, 0);  
        setVisible(false);  
    }else{  
        JOptionPane.showMessageDialog(null, "Selecione um Cliente");  
    }  
}
```

Retorna o índice da linha selecionada. Se for positivo, o usuário selecionou alguma linha; caso contrário, retornará -1 e exibe uma mensagem

Localizar Clientes

- Quando o usuário fizer uma seleção e clicar no botão Selecionar, devemos fechar a janela e armazenar o CPF, que será recuperado na interface de vendas

```
private void selecionarButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    int linha = tabela.getSelectedRow();  
    if (linha >=0){  
        cpf = (String) model.getValueAt(linha,0);  
        setVisible(false);  
    }else{  
        JOptionPane.showMessageDialog(null, "Selecione um Cliente");  
    }  
}
```

Recupera o CPF do cliente selecionado.
O método `getValueAt` indica a linha e a coluna, no caso, temos uma `String`, por isso fizemos o casting para o tipo `String`

Localizar Clientes

- Quando o usuário fizer uma seleção e clicar no botão Selecionar, devemos fechar a janela e armazenar o CPF, que será recuperado na interface de vendas

```
private void selecionarButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    int linha = tabela.getSelectedRow();  
    if (linha >=0){  
        cpf = (String) model.getValueAt(linha,0);  
        setVisible(false);  
    }else{  
        JOptionPane.showMessageDialog(null, "Selecione um Cliente");  
    }  
}
```

Esconde a janela, torna invisível. Não devemos utilizar o dispose() porque precisaremos recuperar o CPF na janela de Vendas

Cadastro de Vendas

Cabecalho

Codigo Venda: Data:

CPF: Nome:

Localizar

Item

Codigo Produto: Descricao:

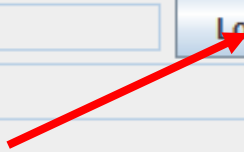
Valor: Quantidade:

Adicionar

Itens da Venda

Codigo	Descricao	Valor Unitario	Quantidade	Subtotal

Total:



Localizar Clientes

- Na Interface Vendas, no botão Localizar, implementaremos a recuperação do CPF do cliente
- Em seguida, localizamos o cliente por meio do controlador, obtendo os dados para exibição

```
private void buttonLocalizarClienteActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    IULocalizaCliente locCliente = new IULocalizaCliente(null, true);  
    locCliente.setVisible(true);  
  
    String cpf = locCliente.getCPF();  
    Cliente c = control.buscarClientePorCPF(cpf);  
    if (c != null) {  
        textCPF.setText(c.getCPF());  
        textNomeCliente.setText(c.getNome());  
    }  
}
```

Localizar Clientes

- Na Interface Vendas, no botão Localizar, implementaremos a recuperação do CPF do cliente
- Em seguida, localizamos o cliente por meio do controlador, obtendo os dados para exibição

```
private void buttonLocalizarClienteActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    IULocalizaCliente locCliente = new IULocalizaCliente(null, true);  
    locCliente.setVisible(true);  
  
    String cpf = locCliente.getCPF();  
    Cliente c = control.buscarClientePorCPF(cpf);  
    if (c != null) {  
        textCPF.setText(c.getCPF());  
        textNomeCliente.setText(c.getNome());  
    }  
}
```

Instância o objeto da janela de localização de clientes e o exibe

Localizar Clientes

- Na Interface Vendas, no botão Localizar, implementaremos a recuperação do CPF do cliente
- Em seguida, localizamos o cliente por meio do controlador, obtendo os dados para exibição

```
private void buttonLocalizarClienteActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    IULocalizaCliente locCliente = new IULocalizaCliente(null, true);  
    locCliente.setVisible(true);  
  
    String cpf = locCliente.getCPF();  
    Cliente c = control.buscarClientePorCPF(cpf);  
    if (c != null) {  
        textCPF.setText(c.getCPF());  
        textNomeCliente.setText(c.getNome());  
    }  
}
```

Recupera o conteúdo do atributo CPF da interface de localização de clientes

Localizar Clientes

- Na Interface Vendas, no botão Localizar, implementaremos a recuperação do CPF do cliente
- Em seguida, localizamos o cliente por meio do controlador, obtendo os dados para exibição

```
private void buttonLocalizarClienteActi
// TODO add your handling code here
IULocalizaCliente locCliente = new
locCliente.setVisible(true);

String cpf = locCliente.getCPF();
Cliente c = control.buscarClientePorCPF(cpf);
if (c != null) {
    textCPF.setText(c.getCPF());
    textNomeCliente.setText(c.getNome());
}
}
```

Recupera o cliente, se existir, exibe os dados nas caixas de texto da interface de vendas

Recupera o cliente, se existir, exibe os dados nas caixas de texto da interface de vendas

Sistema de Cadastro de Produtos

Cadastro Vendas Relatórios

Cadastro de Vendas

Cabecalho

Seleção de Clientes

Código	Nome
11111	Cliente 1
22222	Cliente 2
33333	Cliente 3
44444	Cliente 4
55555	Cliente 5
452342	Henrique
1234558	José

Localizar

Subtotal

Seletor

Fechar

Total 0

Cadastro de Vendas

Cabecalho

Codigo Venda:

Data:

CPF:

Nome: **Localizar**

Item

Codigo Produto:

Descricao:

Valor:

Quantidade: **Adicionar**

Itens da Venda

Codigo	Descricao	Valor Unitario	Quantidade	Subtotal

Total:

Referências

BIBLIOGRAFIA BÁSICA

1. SINTES, A., Aprenda programação orientada a objetos em 21 dias, Pearson Education do Brasil, 2002.
2. VAREJÃO, F., Linguagens de programação : Java, C e C++ e outras : conceitos e técnicas, Campus, 2004.
3. DEITEL, H. M., DEITEL, P. J., **Java:** como programar, São Paulo: Pearson Education do Brasil, 2010. 1144p.
4. DEITEL, H. M., DEITEL, P. J., **Java:** como programar, Porto Alegre: Bookman, 2003. 1386p.
5. SAVITCH, W. J., C++ absoluto, Pearson Education : Addison Wesley, 2004.

BIBLIOGRAFIA COMPLEMENTAR

1. BERMAN, A. M. *Data Structures via C++: Objects by Evolution*, Oxford University Press Inc., 1997.
2. BARNES, D.J. & KÖLLING, M., Programação orientada a objetos com Java, Pearson Education : Prentice Hall, 2004.
3. DEITEL, H. M. e DEITEL, P. J. *C++: Como Programar*, Bookman, 2001.
4. GILBERT, R. F. e FOROUZAN, B. A. *Data Structures: A Pseudo Approach with C++*, Brooks/Cole Thomson Learning, 2001.
5. MUSSER, D. R. e SAINI, A. *STL Tutorial and Reference Guide: Programming with the Standard Template Library*, Addison-Wesley, 1996.
6. SEBESTA, R. W. *Conceitos de Linguagem de Programação*, 4ª Ed., Bookman, 2003.
7. SEDGEWICK, R. *Algorithms in C++*, Addison-Wesley, 2002.
8. STROUSTRUP, B. *A Linguagem de Programação C++*, 3ª Ed., Bookman, 2000.