
Java Starter
www.t2ti.com

Apresentação

O Curso Java Starter foi projetado com o objetivo de ajudar àquelas pessoas que têm uma base de lógica de programação e desejam entrar no mercado de trabalho sabendo Java,

A estrutura do curso é formada por módulos em PDF e por mini-cursos em vídeo. O aluno deve baixar esse material e estudá-lo. Deve realizar os exercícios propostos. Todas as dúvidas devem ser enviadas para a lista de discussão que está disponível para inscrição na página do Curso Java Starter no site www.t2ti.com. As dúvidas serão respondidas pelos instrutores Albert Eije, Cláudio de Barros e Miguel Kojiio, além dos demais participantes da lista.

Nosso objetivo é que após o estudo do Curso Java Starter o aluno não tenha dificuldades para acompanhar um curso avançado onde poderá aprender a desenvolver aplicativos para Web, utilizando tecnologias como Servlets e JSP e frameworks como Struts e JSF, além do desenvolvimento para dispositivos móveis.

Albert Eije trabalha com informática desde 1993. Durante esse período já trabalhou com várias linguagens de programação: Clipper, PHP, Delphi, C, Java, etc. Atualmente mantém o site www.alberteije.com.

Cláudio de Barros é Tecnólogo em Processamento de Dados.

Miguel Kojiio é bacharel em Sistemas de Informação, profissional certificado Java (SCJP 1.5).

O curso Java Starter surgiu da idéia dos três amigos que trabalham juntos em uma instituição financeira de grande porte.

Módulo 10

Aplicações gráficas SWING

Interface gráfica em Java

As aplicações gráficas são aquelas que possibilitam a criação de uma GUI (Graphical User Interface – Interface Gráfica do Usuário), onde definimos os componentes que serão utilizados e suas disposições na tela, permitindo também uma interação com o usuário por meio do mouse e teclado.

O Java possui duas bibliotecas gráficas: AWT (Abstract Window Toolkit) e Swing. A primeira a ser criada foi a AWT e esta foi substituída pelo **swing** a partir do Java 1.2. Neste módulo, falaremos somente do **swing**, por ter melhor aparência, melhor tratamento de eventos, recursos estendidos, além de todas as classes deste pacote serem extensões do pacote AWT.

As classes do **swing** são encontradas no pacote **javax.swing**. Usando **swing**, sua aplicação terá a mesmo forma, aparência e comportamento independente de sistema operacional.

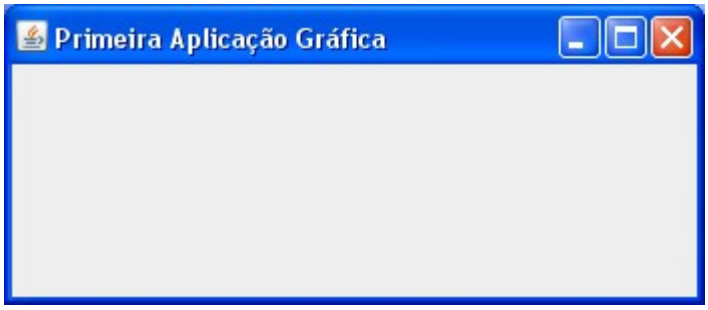
Primeira Aplicação Gráfica

Todo componente **swing** deve ser adicionado a um contêiner. O contêiner essencial para uma aplicação gráfica é o **JFrame**.

```
import javax.swing.JFrame; //importação da classe JFrame

public class PrimeiroSwing {
      //criamos um JFrame chamado 'janela'
      JFrame janela = new JFrame();

      public static void main (String args[]){
            new PrimeiroSwing();
      }
      private PrimeiroSwing(){
```

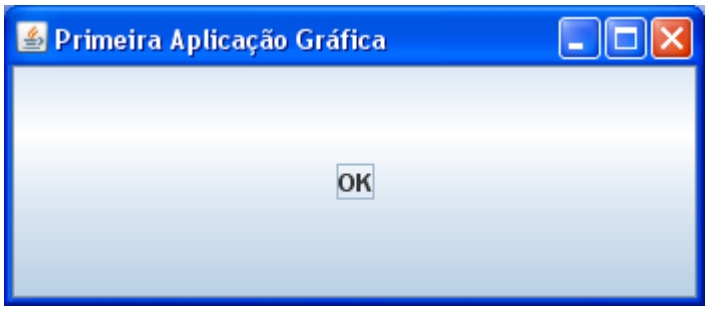
```
            //definimos o título da janela
            janela.setTitle("Primeira Aplicação Gráfica");
            //definimos a largura e a altura da janela
            janela.setSize(350, 150);
            //define a posição da janela na tela
            janela.setLocation(50, 50);
            //define que ao fechar a janela, encerre a aplicação
            janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            //mostramos a janela
            janela.setVisible(true);
      }
}
```

Depois de compilarmos e executarmos a classe, temos o resultado:



Vamos adicionar um botão a esta janela:

```
import javax.swing.JButton;
import javax.swing.JFrame;

public class PrimeiroSwing {

      //criamos e instaciamos um JFrame chamado 'janela'
      JFrame janela = new JFrame();
      //criamos e instaciamos um JButton chamado 'botao' e com o texto "OK"
      JButton botao = new JButton("OK");

      public static void main (String args[]){
            new PrimeiroSwing();
      }

      private PrimeiroSwing(){
            //definimos o título da janela
            janela.setTitle("Primeira Aplicação Gráfica");
            //definimos a largura e a altura da janela
            janela.setSize(350, 150);
            //define que ao fechar a janela, encerre a aplicação
            janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            //adicionamos o botao à janela
            janela.add(botao);
            //mostramos a janela
            janela.setVisible(true);
      }
}
```



Agora temos uma janela com um botão. Perceba que o botão ocupou toda a área da janela, e ao clicar no botão nada acontece. Isto acontece porque não foi definido nenhum layout para a janela e nenhum evento para o botão.

O método "`setDefaultCloseOperation()`" foi usado para definir que ao fecharmos a janela, encerra-se a aplicação (`JFrame.EXIT_ON_CLOSE`). Normalmente definimos este valor na janela principal da aplicação. Quando instanciamos um **JFrame**, o valor padrão é definido como (`JFrame.HIDE_ON_CLOSE`). Se deixarmos com o valor padrão, ao fecharmos a janela, o processo continuará rodando, apenas será "escondida a janela". Temos também a opção (`JFrame.DISPOSE_ON_CLOSE`) que destrói a janela, sem encerrar a aplicação caso esta não seja a janela principal do sistema.

No decorrer do estudo dos gerenciadores de layout e de tratamento de eventos, veremos novos componentes do **swing** e no final do módulo seus principais métodos.

Gerenciadores de Layout

Uma GUI é construída a partir de componentes chamados contêineres que contém outros componentes. Os mais usados são o **JFrame** e o **JPanel**. Caso nenhum gerenciador de layout seja especificado, será considerado que toda a extensão do contêiner é formada por uma única célula, possibilitando assim apenas a inclusão de um objeto.

Os principais gerenciadores de layout são: **FlowLayout**, **GridLayout**, **BorderLayout** *e* **CardLayout**.

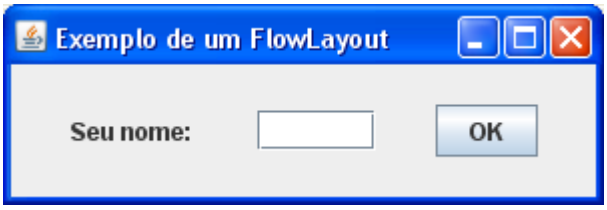
FlowLayout

O **FlowLayout** é o gerenciador mais simples. Os componentes são dispostos da esquerda para a direita na ordem em que aparecem, isto é, na ordem em que são adicionados. Quando não existe mais espaço em uma linha, é criada uma outra linha, se assemelhando a um editor de texto. Este processo é feito automaticamente de acordo com o tamanho do contêiner.

Vamos incluir em uma janela, um **JPanel** (contêiner), um **JText** (caixa de texto) , um **JLabel** (rótulo) e um **JButton** (botão).

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class ExemploFlowLayout {

	//criamos e instaciamos um JFrame chamado 'janela'
	JFrame janela = new JFrame();
	//criamos um JPanel chamado painel
	JPanel painel = new JPanel();
	//criamos um JLabel chamado rotulo e com o texto "Seu nome: "
	JLabel rotulo = new JLabel("Seu nome: ");
	//criamos um JTextField chamado texto com o tamanho 5
	JTextField texto = new JtextField(5);
	//criamos e instanciamos um JButton chamado 'botao' e com o texto "OK"
	JButton botao = new JButton("OK");

	public static void main (String args[]){
		new ExemploFlowLayout();
	}

	private ExemploFlowLayout(){
		//definimos o título da janela
		janela.setTitle("Exemplo de um FlowLayout");
		//definimos a largura e a altura da janela
		janela.setSize(300, 100);
		//define a posição da janela na tela
		janela.setLocation(50, 50);
		//define que ao fechar a janela, encerre a aplicação
		janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		//definimos o layout do painel
		painel.setLayout(new FlowLayout(FlowLayout.CENTER, 30, 20));
		//adicionamos o rotulo ao painel
		painel.add(rotulo);
		//adicionamos o texto ao painel
		painel.add(texto);
		//adicionamos o botao ao painel
		painel.add(botao);
		//adicionamos o painel à janela
		janela.add(painel);
		//mostramos a janela
```

```
            janela.setVisible(true);
        }
}
```



Perceba que os componentes foram posicionados na janela de acordo com a sequência em que os adicionamos no código (primeiro o rótulo, depois o texto, e por último o botão). A sintaxe do construtor do **FlowLayout** é a seguinte:

FlowLayout(alinhamento, espaçamento-horizontal, espaçamento-vertical)

O nosso painel foi definido com o alinhamento centralizado, 30 de espaçamento horizontal e 20 de espaçamento vertical.

GridLayout

O **GridLayout** é um gerenciador que divide um contêiner em um conjunto de células espalhadas numa grade retangular, de maneira que todas elas possuam a mesma dimensão. Pode-se dividir um contêiner em linhas e colunas de acordo com sua necessidade. Os componentes são dispostos na ordem em que aparecem, sendo inseridos na grade da esquerda para a direita e de cima para baixo.

Qualquer modificação no tamanho do contêiner será automaticamente alterado o tamanho dos componentes adicionados a ele, ou seja, os componentes são redimensionados em função da nova dimensão do contêiner.

Para definirmos um layout na forma de "grade" utilizamos a seguinte sintaxe:

```
nome-do-conteiner.setLayout(new GridLayout(linhas, colunas, espaçamento-horizontal, espaçamento-vertical)
```

linhas --> número de linhas que terá o contêiner

colunas --> número de colunas que terá o contêiner

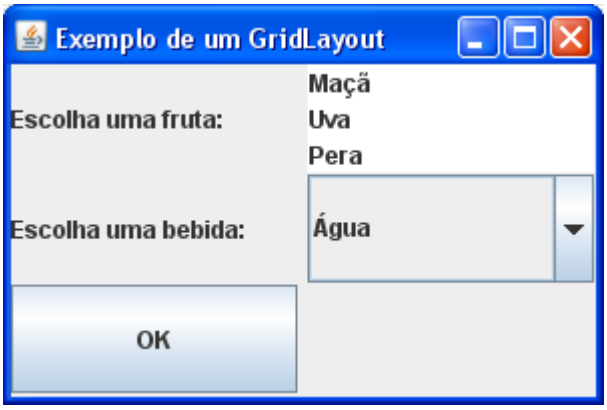
espaçamento-horizontal --> distância horizontal entre os componentes

espaçamento-vertical --> distância vertical entre os componentes

Vamos criar uma janela contendo os componentes **JPanel** (contêiner), **JList** (lista), um **JComboBox** (combo), dois **JLabel** (rótulo) e um **JButton** (botão):

```
import java.awt.GridLayout;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;

public class ExemploGridLayout {
	//criamos um JFrame chamado 'janela'
	JFrame janela = new JFrame();
	//criamos um JPanel chamado painel
	JPanel painel = new JPanel();
	//criamos um JLabel chamado rotulo1 e com o texto "Escolha uma fruta: "
	JLabel rotulo1 = new JLabel("Escolha uma fruta: ");
	//criamos um JLabel chamado rotulo2 e com o texto "Escolha uma bebida: "
	JLabel rotulo2 = new JLabel("Escolha uma bebida: ");
	/*  criamos um ListModel que é o objeto que contém as opções da lista
	 *  não se preocupem com isso agora, pois veremos com mais detalhes
	 */
	DefaultListModel listModel = new DefaultListModel();
	//criamos um JList chamado lista e definimos onde estão os objetos da
lista
	JList lista = new JList(listModel);
	//criamos um JComboBox chamado combo
	JComboBox combo = new JComboBox();
	//criamos um JButton chamado 'botao' e com o texto "OK"
	JButton botao = new JButton("OK");

	public static void main (String args[]){
		new ExemploGridLayout();
	}

	private ExemploGridLayout(){
		//definimos o título da janela
		janela.setTitle("Exemplo de um FlowLayout");
		//definimos a largura e a altura da janela
		janela.setSize(300, 200);
		//define a posição da janela na tela
		janela.setLocation(50, 50);
		//define que ao fechar a janela, encerre a aplicação
		janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		//definimos o layout do painel
		painel.setLayout(new GridLayout(3, 2, 5, 1));
		//adicionamos o rotulo1 ao painel
		painel.add(rotulo1);
		//acrescentamos algumas frutas a lista
		listModel.addElement("Maçã");
		listModel.addElement("Uva");
		listModel.addElement("Pera");
		//adicionamos a lista ao painel
		painel.add(lista);
		//adicionamos o rotulo2 ao painel
		painel.add(rotulo2);
		//acrescentamos algumas bebidas ao combo
		combo.addItem("Água");
		combo.addItem("Refrigerante");
		combo.addItem("Suco");
		//adicionamos o combo ao painel
		painel.add(combo);
		//adicionamos o botao ao painel
```

```
            painel.add(botao);
            //adicionamos o painel à janela
            janela.add(painel);
            //mostramos a janela
            janela.setVisible(true);
        }
}
```



BorderLayout

O **BorderLayout** é um gerenciador que divide um contêiner em cinco regiões distintas: **north** (região superior), **south** (região inferior), **west** (região à esquerda), **east** (região à direita) e **center** (região central). Diferentemente dos gerenciadores vistos anteriormente, a ordem em que os componentes são adicionados é irrelevante, pois no momento em que adicionamos o componente, definimos em qual região o mesmo irá ficar. Em cada região, conseguimos colocar apenas um componente, ou seja, somente 5 (cinco) componentes podem ser inseridos neste layout. Caso um componente seja inserido em uma região que já contenha outro, este será sobreposto. Da mesma maneira que o **GridLayout**, os componentes são redimensionados de acordo com as dimensões do contêiner.

Pode ser que seja uma desvantagem podermos adicionar somente 5 componentes, mas se adicionarmos um painel em cada região, poderemos ter 5 painéis que terão seus próprios componentes e respectivos gerenciadores de layout.

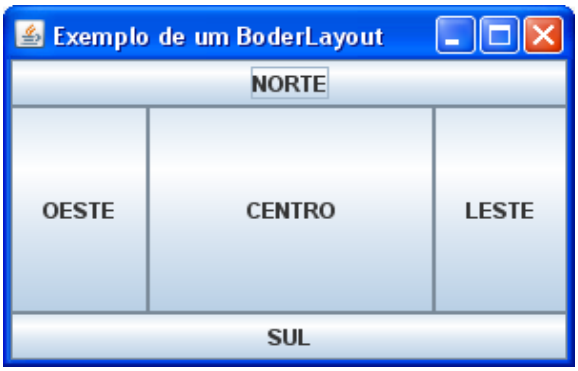
Para a definição do **BorderLayout**, utilizamos a seguinte sintaxe:

```
nome-do-conteiner.setLayout(new BorderLayout(espaçamento-horizontal, espaçamento-vertical)
```

espaçamento-horizontal e espaçamento-vertical --> parâmetros opcionais que definem o espaço entre os objetos.

Vamos definir um **JPanel** com o layout **BorderLayout**, e incluir um **JButton** em cada região do contêiner:

```
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class ExemploBorderLayout {
	//criamos um JFrame chamado 'janela'
	JFrame janela = new JFrame();
	//criamos um JPanel chamado painel
	JPanel painel = new JPanel();
	//criamos um JButton chamado sul
	JButton sul = new JButton("SUL");
	//criamos um JButton chamado norte
	JButton norte = new JButton("NORTE");
	//criamos um JButton chamado leste
	JButton leste = new JButton("LESTE");
	//criamos um JButton chamado oeste
	JButton oeste = new JButton("OESTE");
	//criamos um JButton chamado centro
	JButton centro = new JButton("CENTRO");

	public static void main (String args[]){
		new ExemploBorderLayout();
	}

	private ExemploBorderLayout(){
		//definimos o título da janela
		janela.setTitle("Exemplo de um BoderLayout");
		//definimos a largura e a altura da janela
		janela.setSize(300, 200);
		//define a posição da janela na tela
		janela.setLocation(50, 50);
		//define que ao fechar a janela, encerre a aplicação
		janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		//definimos o layout do painel
		painel.setLayout(new BorderLayout());
		//adicionamos os botões ao painel
		painel.add("North", norte);
		painel.add("South", sul);
		painel.add("East", leste);
		painel.add("West", oeste);
		painel.add("Center", centro);
		//adicionamos o painel à janela
		janela.add(painel);
		//mostramos a janela
		janela.setVisible(true);
	}
}
```



Para a utilização do **BorderLayout** existe uma pequena diferença em relação aos gerenciadores anteriores: é necessário infomar no método **add()** em qual região o objeto será inserido.

Neste layout, não é obrigatório preencher todas as regiões. É possível utilizar apenas as regiões desejadas. Na classe anterior, vamos retirar os botões do "Leste" e "Sul" e ver o resultado:

Como não existiam componentes nestas regiões, os botões "oeste" e "centro" utilizam o espaço que ficou livre.

CardLayout

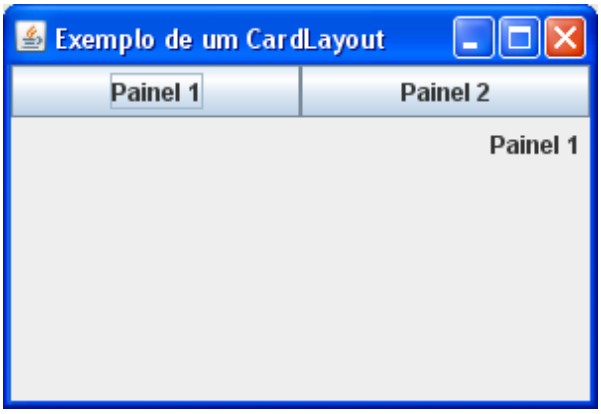
O **CardLayout** se constitui em um gerenciador mais aprimorado que pode agrupar diversos contêineres na forma de cartões, mostrando um de cada vez, ou seja, apenas um contêiner é visível por vez. Cada contêiner pode possuir seu layout específico, permitindo que diversos gerenciadores de layout sejam usados em um mesmo espaço da janela.

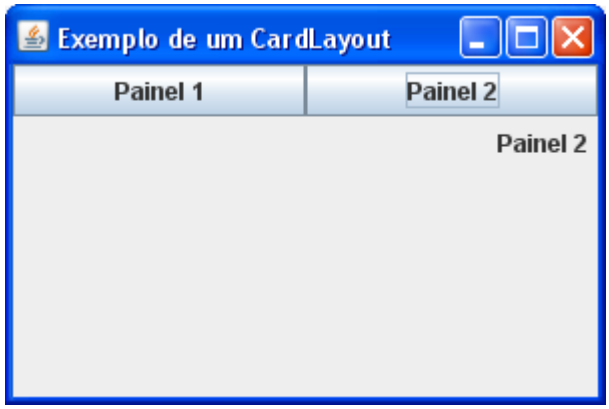
Vamos criar uma janela sendo gerenciada pelo **BorderLayout**, contendo três

JPanel (painéis), sendo que um deles será gerenciado pelo **CardLayout** e os outros sem gerenciador contendo um **JLabel** (rótulo). Para alternar entre os painéis, vamos incluir dois **JButton** (botões) em um **JPanel**:

```
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class ExemploCardLayout implements ActionListener{
	//criamos um JFrame chamado 'janela'
	JFrame janela = new JFrame();
	//criamos um JPanel chamado painel1
	JPanel painel1 = new JPanel();
	//criamos um JPanel chamado painel2
	JPanel painel2 = new JPanel();
	//criamos um JLabel chamado rotulo1
	JLabel rotulo1 = new JLabel("Painel 1", JLabel.CENTER);
	//criamos um JLabel chamado rotulo2
	JLabel rotulo2 = new JLabel("Painel 2", JLabel.CENTER);
	//criamos dois botoes e um painel para alternar entre os paineis
	JPanel painelSelecao = new JPanel();
	JButton botao1 = new JButton("Painel 1");
	JButton botao2 = new JButton("Painel 2");
	//criamos o painelCard que será gerenciado pelo CardLayout
	JPanel painelCard = new JPanel();

	public static void main (String args[]){
		new ExemploCardLayout();
	}

	private ExemploCardLayout(){
		//definimos o título da janela
		janela.setTitle("Exemplo de um CardLayout");
		//definimos a largura e a altura da janela
		janela.setSize(300, 200);
		//define a posição da janela na tela
		janela.setLocation(50, 50);
		//define que ao fechar a janela, encerre a aplicação
		janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		//definimos o layout da janela
		janela.setLayout(new BorderLayout());
		//definimos o painelselecao com o GridLayout e incluimos os botoes
		painelSelecao.setLayout(new GridLayout(1,2));
		painelSelecao.add(botao1);
		painelSelecao.add(botao2);
		//adicionamos os rotulos a seus respectivos paineis
		painel1.add(rotulo1);
		painel2.add(rotulo2);
		//definimos o layout do painelCard
		painelCard.setLayout(new CardLayout());
		//adicionamos os paineis quem contém os rotulos ao painelCard
		painelCard.add(painel1, "p1");
		painelCard.add(painel2, "p2");
```

```
            //adicionamos os paineis à janela
            janela.add("North", painelSelecao);
            janela.add("East", painelCard);
            //registra os botoes para tratarmos os eventos gerados por eles
            botao1.addActionListener(this);
            botao2.addActionListener(this);
            //mostramos a janela
            janela.setVisible(true);
        }

        //veremos este metodo com mais detalhe no proximo topico
        public void actionPerformed(ActionEvent e){
            CardLayout cl = (CardLayout) painelCard.getLayout();
            if (e.getSource() == botao1){
                cl.show(painelCard, "p1");
            }
            if (e.getSource() == botao2){
                cl.show(painelCard, "p2");
            }
        }
}
```





No trecho de código mostrado com a seta verde, definimos o layout do painel "painelCard" como sendo do tipo **CardLayout** e adicionamos os paineis que contém os rótulos. Perceba que no método **add()** os parâmetros são: componente a ser adicionado, e uma **String** como identificador do componente.

No trecho marcado com a seta azul, criamos um objeto do tipo **CardLayout** e definimos qual painel seria visível no momento. Caso o usuário clique no botão 1, o painel 1 será mostrado, e caso clique no botão 2, o painel 2 será mostrado. Para que mostremos o painel desejado, usamos a seguinte sintaxe:

*nome-do-cardlayout.**show**(nome-do-conteiner, identificador)*

Tratamento de Eventos

Nas aplicações em que desejamos interpretar as ações do usuário (ao clicar em um botão, por exemplo), necessitamos implementar uma ou mais interfaces receptoras de eventos. Cada classe receptora de eventos trata de um evento diferente. Alguns exemplos:

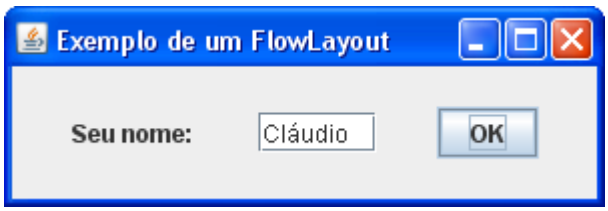
- **ActionListener** – Eventos de ação como o clique do mouse sobre um botão, ao pressionar "ENTER" em um campo texto, etc.
- **FocusListener** – Eventos de foco, gerados quando um componente recebe ou perde o foco.
- **MouseListener –** eventos gerados pelo mouse quando é clicado, entra na área de um componente, etc.
- **WindowListener** – eventos de janelas, gerados quando uma janela é maximizada, minimizada, etc.

Neste módulo vamos aprender a utilizar a interface **ActionListener**. Esta classe exige a declaração do método **actionPerformed**, que é onde vamos definir o tratamento do evento gerado. Através do método **addActionListener** registramos os componentes que serão "observados".

Vamos utilizar a classe de exemplo do gerenciador **FlowLayout** para registrar o botão e definir que ao ser clicado mostre o conteúdo do campo texto.

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

//implementamos a classe ActionListener para controlar os eventos
public class ExemploFlowLayout implements ActionListener{

	//criamos um JFrame chamado 'janela'
	JFrame janela = new JFrame();
	//criamos um JPanel chamado painel
	JPanel painel = new JPanel();
	//criamos um JLabel chamado rotulo e com o texto "Seu nome: "
	JLabel rotulo = new JLabel("Seu nome: ");
	//criamos um JTextField chamado texto com o tamanho 5
	JTextField texto = new JTextField(5);
	//criamos e instaciamos um JButton chamado 'botao' e com o texto "OK"
```

```
	JButton botao = new JButton("OK");

	public static void main (String arg[]){
		new ExemploFlowLayout();
	}

	private ExemploFlowLayout(){
		//definimos o título da janela
		janela.setTitle("Exemplo de um FlowLayout");
		//definimos a largura e a altura da janela
		janela.setSize(300, 100);
		//define a posição da janela na tela
		janela.setLocation(50, 50);
		//define que ao fechar a janela, encerre a aplicação
		//janela.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
		//definimos o layout do painel
		painel.setLayout(new FlowLayout(FlowLayout.CENTER, 30, 20));
		//adicionamos o rotulo ao painel
		painel.add(rotulo);
		//adicionamos o texto ao painel
		painel.add(texto);
		//adicionamos o botao ao painel
		painel.add(botao);
		//adicionamos o painel à janela
		janela.add(painel);
		//mostramos a janela
		janela.setVisible(true);
		//registramos o botao ao Listener
		botao.addActionListener(this);
	}

	//construimos o metodo exigido pela interface
	public void actionPerformed(ActionEvent e){
		if (e.getSource() == botao){
			JOptionPane.showMessageDialog(null, texto.getText());
		}
	}
}
```



Na linha mostrada pela seta azul, registramos o botão para que os eventos gerados por ele sejam tratados pela interface.

O bloco representado pela seta verde, é onde definimos o que será feito ao ser gerado um evento. Para sabermos qual componente gerou o evento, utilizamos o método **getSource()** da classe **ActionEvent.** Neste exemplo, se o componente que gerou o evento foi o botão, então mostramos uma mensagem com o conteúdo do campo texto.

Componentes Swing

Neste tópico veremos alguns componentes do **swing**, entre eles:

- **JButton**
- **JOptionPane**
- **JLabel**
- **JTextField**
- **JComboBox**
- **JCheckBox**
- **JRadioButton**

JButton

Utilizando a classe **JButton** podemos incluir botões em um contêiner.

A tabela seguinte mostra alguns métodos da classe **Jbutton**:

Método	Descrição
JButton()	Constrói um botão sem texto *
JButton(String)	Constrói um botão com o texto informado *
JButton(String, Icon)	Constrói um botão com o texto e imagem informados *
getText()	Obtém o texto do botão
setText(String)	Define o texto do botão
setEnabled(boolean)	Define se o botão está habilitado (true) ou desabilitado (false)

* construtores

Exemplo:

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;


public class Botoes implements ActionListener{

      JFrame janela = new JFrame();
      //criamos dois botões
      JButton botao1 = new JButton();
      JButton botao2 = new JButton();
```

```
	public static void main (String args[]){
		new Botoes();
	}

	public Botoes(){
		janela.setSize(240,70);
		janela.setTitle("Exemplo de JButton");
		janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		janela.setLayout(new GridLayout(1,2));
		//definimos o texto dos botões
		botao1.setText("Fechar");
		botao2.setText("OK");
		//registramos os botoes no ActionListener
		botao1.addActionListener(this);
		botao2.addActionListener(this);
		//adicionamos os botões a janela
		janela.add(botao1);
		janela.add(botao2);
		janela.setVisible(true);
	}

	public void actionPerformed(ActionEvent e){
		if (e.getSource() == botao1){
			//se pressionado o botao1, fechamos a aplicação
			System.exit(0);
		}
		if (e.getSource() == botao2){
			//se pressionado o botao2, mostramos uma mensagem
			JOptionPane.showMessageDialog(null, "Janela de botões");
		}
	}
}
```

JOptionPane

Utilizamos a classe **JOptionPane** para a inclusão de mensagens. Esta classe é especializada na geração de caixas de diálogo, isto é, pequenas janelas que fazem uma pergunta, avisam o usuário ou fornecem uma mensagem qualquer. Existem quatro tipos padrão de caixas de diálogo, sendo eles:

- **MessageDialog –** caixa de diálogo que apresenta uma mensagem, possibilitando acrescentar ícones de alerta ao usuário.
- **ConfirmDialog** – caixa de diálogo que além de emitir uma mensagem,

possibilita ao usuário responder a uma pergunta.

- **InputDialog** – caixa de diálogo que além de emitir uma mensagem, permite a entrada de um texto.
- **OptionDialog** – caixa de diálogo que abrange os três tipos anteriores.

Vamos incrementar a classe "Botoes" e antes de fechar a janela ao ser pressionado o botão "Fechar", confirmar a ação do usuário. No botão "OK", o texto a ser apresentado será o que o usuário digitar no **InputDialog**:

```
public void actionPerformed(ActionEvent e){
  if (e.getSource() == botao1){
     //se pressionado o botao1, pedimos confirmação do usuário
     int resposta = JOptionPane.showConfirmDialog(null, "Deseja encerrar a aplicação?",
                                  "Confirmação", JOptionPane.OK_CANCEL_OPTION);
     //se a resposta for sim, encerra a aplicação
     if (resposta == JOptionPane.OK_OPTION){
         System.exit(0);
     }
  }

  if (e.getSource() == botao2){
     //se pressionado o botao2, solicitamos ao usuário digitar uma mensagem
     String mensagem = JOptionPane.showInputDialog("Informe a mensagem:");
     //mostramos a mensagem informada pelo usuário
     JOptionPane.showMessageDialog(null, mensagem);
  }
}
```

No bloco indicado pela seta azul, declaramos um objeto do tipo **int** que irá receber o valor da resposta na caixa de confirmação. A sintaxe utilizada para o método `showConfirmDialog()` é:

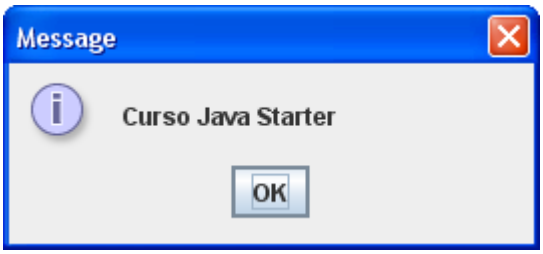
JOptionPane`.showConfirmDialog`*(Component, mensagem, título, botões)*

- **Component**– refere-se a um objeto do tipo contêiner que permite definir a posição da tela em que a caixa de mensagem aparecerá. Quando é utilizado **null**, a mensagem é centralizada na tela.
- **Mensagem** – mensagem do tipo **String** que será informada ao usuário
- **Título –** texto que aparece na barra de título. Também do tipo **String**
- **Tipo –** os botões que estarão presentes na caixa de diálogo. No nosso exemplo, estão os botões "OK" e "Cancel".

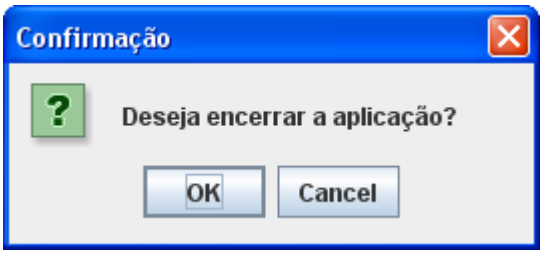
No bloco indicado pela seta amarela, declaramos um objeto do tipo **String** que irá receber a mensagem informada pelo usuário no **InputDialog**. O parâmetro utilizado é a mensagem que será informada ao usuário.

Executando a aplicação, ao pressionar o botão "OK" temos:

Ao pressionarmos "OK" aparece a mensagem:



Quando pressionamos o botão "Fechar", aparece a tela de confirmação:



Ao pressionarmos "OK" a aplicação é encerrada.

JLabel

A classe **JLabel** é utilizada para a inclusão de texto (rótulos) e/ou imagens. Esta classe não gera eventos, e também não permanece com o foco.

A tabela seguinte mostra alguns métodos desta classe:

Método	Descrição
JLabel()	Constrói um label sem texto
JLabel(String)	Constrói um label com o texto informado
JLabel(Icon)	Constrói um label contendo a imagem especificada
setText(String)	Define o texto a ser mostrado

getText()	Retorna o texto que o label está mostrando

JTextField

A classe **JTextField** é utilizada para a inclusão de campos textos editáveis de uma única linha. A tabela seguinte mostra alguns métodos desta classe:

Método	Descrição
JTextField()	Constrói um novo campo texto
JTextField(int)	Constrói um campo texto com o número de colunas (tamanho) informado
JTextField(String)	Constrói um campo texto com o texto especificado
setText(String)	Define o texto do campo texto para o informado
getText()	Retorna o texto do campo texto
setEnabled(boolean)	Define se o campo texto está habilitado (true) ou desabilitado (false)

JComboBox

Utilizamos a classe **JComboBox** para criarmos caixas de seleção. Estas caixas permitem que o usuário selecione apenas um item de sua lista.

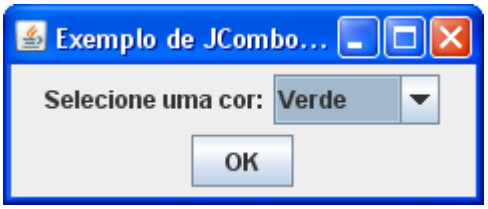
A tabela seguinte mostra alguns métodos da classe **JcomboBox**:

Método	Descrição
JComboBox()	Cria uma caixa de seleção
addItem(Object)	Adiciona o objeto como um novo item
getSelectedItem()	Retorna o objeto que está selecionado
getSelectedIndex()	Retorna um valor do tipo int que representa o índice selecionado
removeItemAt(int)	Remove o item com o índice especificado
removeAllItems()	Remove todos os itens da lista

Exemplo:

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
```

```
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;


public class ExemploCombo implements ActionListener{

	JFrame janela = new JFrame();
	JLabel label = new JLabel("Selecione uma cor:");
	JButton botao = new JButton("OK");
	//criamos um novo combo
	JComboBox combo = new JComboBox();

	public static void main (String args[]){
		new ExemploCombo();
	}

	public ExemploCombo(){
		janela.setSize(240,100);
		janela.setTitle("Exemplo de JComboBox");
		janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		janela.setLayout(new FlowLayout());
		janela.add(label);
		janela.add(combo);
		janela.add(botao);
		janela.setVisible(true);
		botao.addActionListener(this);
		//adicionamos as cores ao combo
		combo.addItem("Vermelho");
		combo.addItem("Verde");
		combo.addItem("Azul");
	}

	public void actionPerformed(ActionEvent e){
		if (e.getSource() == botao){
		//mostramos uma mensagem informando a cor selecionada pelo usuário
			JOptionPane.showMessageDialog(null, combo.getSelectedItem());
		}
	}
}
```



JCheckBox

Para incluirmos caixas de opção utilizamos a classe **JCheckBox** que permite

representar uma opção que está ativada (true) ou desativada(false). As caixas de opção são utilizadas para exibir várias opções dentre as quais o usuário pode optar por selecionar nenhuma, uma ou várias delas.

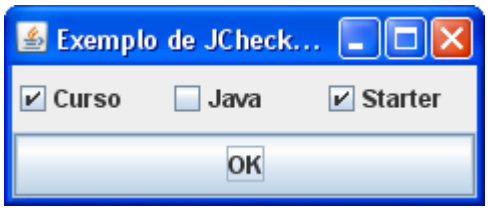
A tabela seguinte mostra alguns métodos da classe **JcheckBox**:

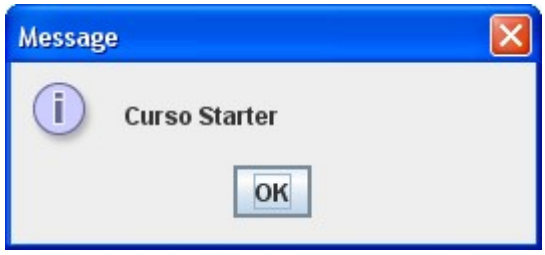
Método	Descrição
JCheckBox()	Cria um checkbox sem texto e não selecionado
JCheckBox(String, boolean)	Cria um checkbox com o texto e estado especificados
isSelected()	Retorna se o checkbox está selecionado (true) ou não (false)

Exemplo:

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;


public class ExemploCheckBox implements ActionListener{
	JFrame janela = new JFrame();
	JPanel painel = new JPanel();
	JButton botao = new JButton("OK");
	//criamos três checkbox
	JCheckBox chk1 = new JCheckBox("Curso");
	JCheckBox chk2 = new JCheckBox("Java");
	JCheckBox chk3 = new JCheckBox("Starter");

	public static void main (String args[]){
		new ExemploCheckBox();
	}

	public ExemploCheckBox(){
		janela.setSize(240,100);
		janela.setTitle("Exemplo de JCheckBox");
		janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		janela.setLayout(new GridLayout(2,1));
		painel.setLayout(new GridLayout(1,1));
		painel.add(chk1);
		painel.add(chk2);
		painel.add(chk3);
		janela.add(painel);
		janela.add(botao);
		janela.setVisible(true);
		botao.addActionListener(this);
	}

	public void actionPerformed(ActionEvent e){
		if (e.getSource() == botao){
```

```
                String mensagem = "";
                //se chk1 selecionado acrescentamos o seu texto a mensagem
                if (chk1.isSelected()){
                      mensagem = chk1.getText() + " ";
                }
                //se chk2 selecionado acrescentamos o seu texto a mensagem
                if (chk2.isSelected()){
                      mensagem += chk2.getText() + " ";
                }
                //se chk3 selecionado acrescentamos o seu texto a mensagem
                if (chk3.isSelected()){
                      mensagem += chk3.getText();
                }
                JOptionPane.showMessageDialog(null, mensagem);
          }
     }
}
```





JRadioButton

Os "radio buttons" são criados a partir da classe **JRadioButton**, e diferentemente dos **JCheckBox**, permitem que apenas uma entre várias opções seja selecionada. Cada conjunto de "radio buttons" deve ser inserido em um **ButtonGroup**.

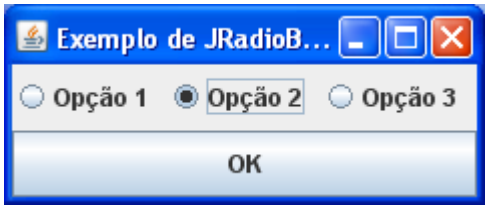
A tabela seguinte mostra alguns métodos da classe **JRadioButton**:

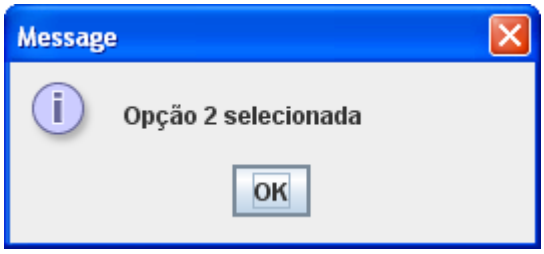
Método	Descrição
JRadioButton()	Cria um radio button sem texto e não selecionado
JRadioButton(String, boolean)	Cria um radio button com o texto e estado especificados
setSelected(boolean)	define se o radio button está ou não selecionado (true ou false)
ButtonGroup()	Cria um grupo para radio buttons
Nome-do-grupo.add(JRadioButton)	Adiciona um radio button a um grupo

Exemplo:

```
import java.awt.GridLayout;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;

public class ExemploJRadioButton implements ActionListener{

	JFrame janela = new JFrame();
	JPanel painel = new JPanel();
	JButton botao = new JButton("OK");
	//criamos três radio buttons
	JRadioButton rb1 = new JRadioButton("Opção 1");
	JRadioButton rb2 = new JRadioButton("Opção 2");
	JRadioButton rb3 = new JRadioButton("Opção 3");
	//criamos o grupo para os radio buttons
	ButtonGroup grupo = new ButtonGroup();

	public static void main (String args[]){
		new ExemploJRadioButton();
	}
	public ExemploJRadioButton(){
		janela.setSize(240,100);
		janela.setTitle("Exemplo de JRadioButton");
		janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		janela.setLayout(new GridLayout(2,1));
		painel.setLayout(new GridLayout(1,1));
		painel.add(rb1);
		painel.add(rb2);
		painel.add(rb3);
		janela.add(painel);
		janela.add(botao);
		janela.setVisible(true);
		botao.addActionListener(this);
		//adicionamos os radio buttons ao grupo
		grupo.add(rb1);
		grupo.add(rb2);
		grupo.add(rb3);
	}
	public void actionPerformed(ActionEvent e){
		if (e.getSource() == botao){
			String mensagem = "Nenhuma opção selecionada";
			//verificamos qual radio button está selecionado
			if (rb1.isSelected()){
				mensagem = "Opção 1 selecionada";
			}else if (rb2.isSelected()){
				mensagem = "Opção 2 selecionada";
			}else if (rb3.isSelected()){
				mensagem = "Opção 3 selecionada";
			}
			JOptionPane.showMessageDialog(null, mensagem);
		}
	}
}
```





Agora você deve tentar resolver a lista de exercícios abaixo. Qualquer dificuldade envie para a lista de discussão:

Exercícios

Aprenda com quem também está aprendendo, veja e compartilhe as suas respostas no nosso Fórum:

Exercícios – Módulo 10 – Aplicações Gráficas – Swing

1. Inclua em um **JFrame** gerenciado pelo layout **GriLayout,** dois **JTextField** e um **JButton**.

2. Utilizando a classe feita no exercício anterior, acrescente a seguinte funcionalidade: quando o usuário clicar no botão, mostre a soma dos valores informados nos campos texto.

3. Crie um janela contendo 3 botões gerenciada pelo **BorderLayout**. Inicialmente, somente um deles estará habilitado. Ao clicar no botão que está habilitado, desabilita o mesmo e habilita somente um dos outros dois.

4. Crie uma aplicação que calcule o valor final de uma venda. Caso o pagamento seja realizado em dinheiro, não haverá acréscimo, caso seja em cartão, terá um acréscimo de 5% e caso seja em cheque, o acréscimo será de 10%. Utilize o **JRadioButton** para que o usuário escolha a forma de pagamento e o **JTextField** para o usuário informar o valor da venda e mostrar o resultado.

5. Refaça a aplicação do exercício anterior, substituindo o **JRadioButton** pelo **JComboBox** e o **JTextField** pelo **InputDialog** para receber o valor e **MessageDialog** para mostrar o resultado.

6. Crie uma aplicação que simule o cadastramento de pessoas. O usuário

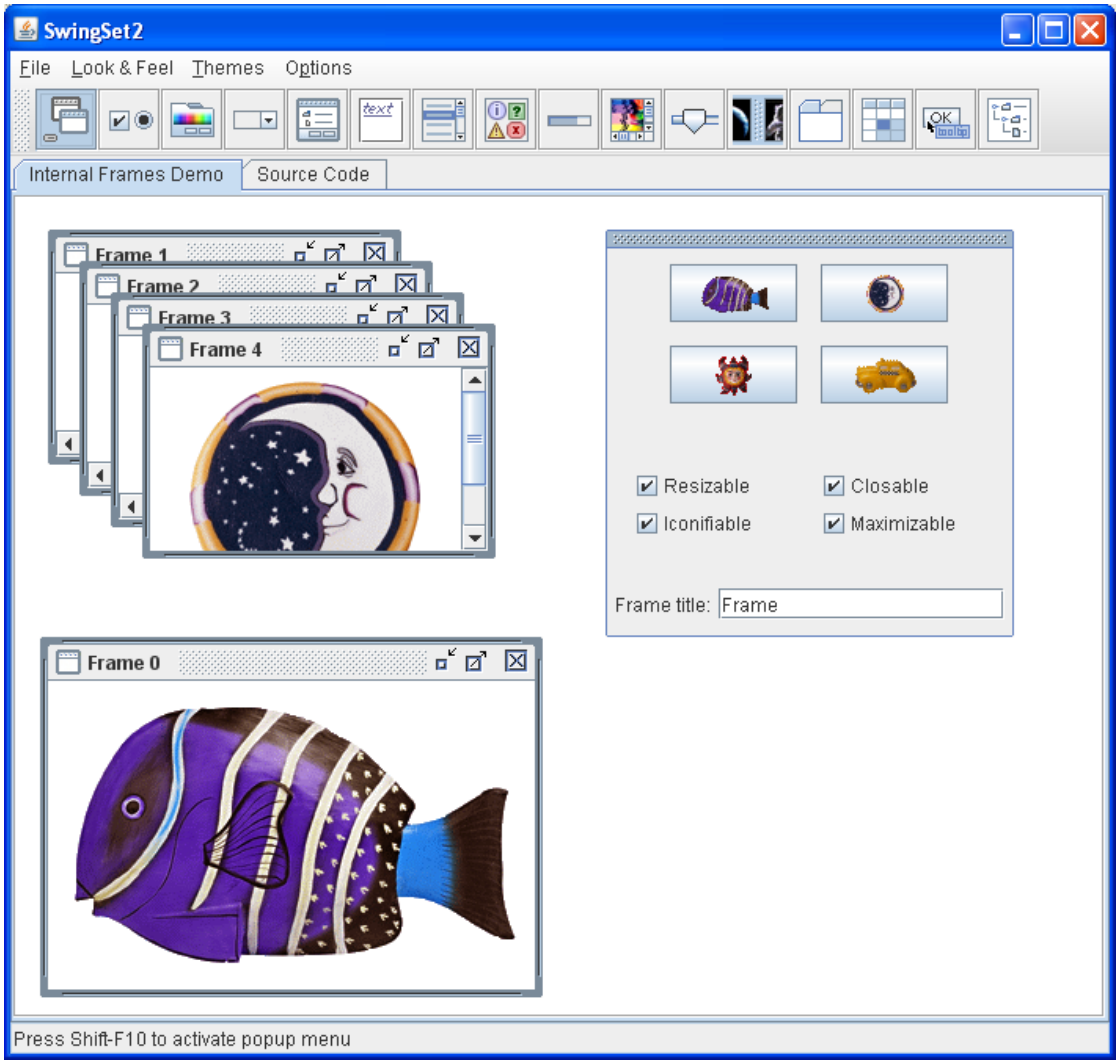
digita o nome e o endereço, escolhe o sexo e estado civil através do **ComboBox**. Ao pressionar um botão, apresenta uma mensagem com os dados informados.

7. Refaça o exercício anterior substituindo o **JComboBox** pelo **JRadioButton.**

8. Faça uma aplicação que apresente 5 itens de uma papelaria. O usuário irá escolher um ou mais itens. Ao clicar no botão "Comprar", o sistema emite uma mensagem com os itens escolhidos pelo usuário. Utilize o **JcheckBox.**

9. Altere a aplicação feita no exercício anterior, confirmando a ação do usuário antes de mostrar a mensagem.

10. Abra o promp de comando e acesse a pasta "diretorio-instalacao-JDK/demo/jfc/SwingSet2/". Digite o comando: java -jar SwingSet2.jar e veja uma aplicação de demonstração feita em **swing.** A tela inicial é a seguinte:



11. No pacote **swing** existem muitas classes. Neste módulo foram vistos alguns componentes básicos e seus principais métodos. Acesse a documentação deste pacote para aprender mais sobre essas classes e seus respectivos métodos: http://java.sun.com/javase/6/docs/api/