ORIENTAÇÃO A OBJETOS

AULA 11

Programação em Ambiente Gráfico

Vandor Roberto Vilardi Rissoli

APRESENTAÇÃO

- Ambiente Gráfico
- Aplicação Gráfica
  - Componentes Gráficos
  - Layouts padrões
  - Eventos
- Menus
- Referências

Ambiente Gráfico

As aplicações gráficas, ou elaboradas com uma interface gráfica, normalmente são mais agradáveis e intuitivas aos seus usuários, porém devem ser bem estudadas e projetadas adequadamente para atingir seus objetivos.

Este tipo de aplicação possibilita a criação de uma interface gráfica (GUI-*Graphical User Interface*) de interação entre o sistema e seus usuários.

Ambiente Gráfico

O desenvolvimento deste tipo de aplicação necessita:

- Definir todos os **componentes** que farão parte desta interface de interação;
- Identificar quais são seus **objetivos** nesta interface;
- Distribuir cada componente nas diversas **posições** possíveis em uma janela de interação.

Ambiente Gráfico

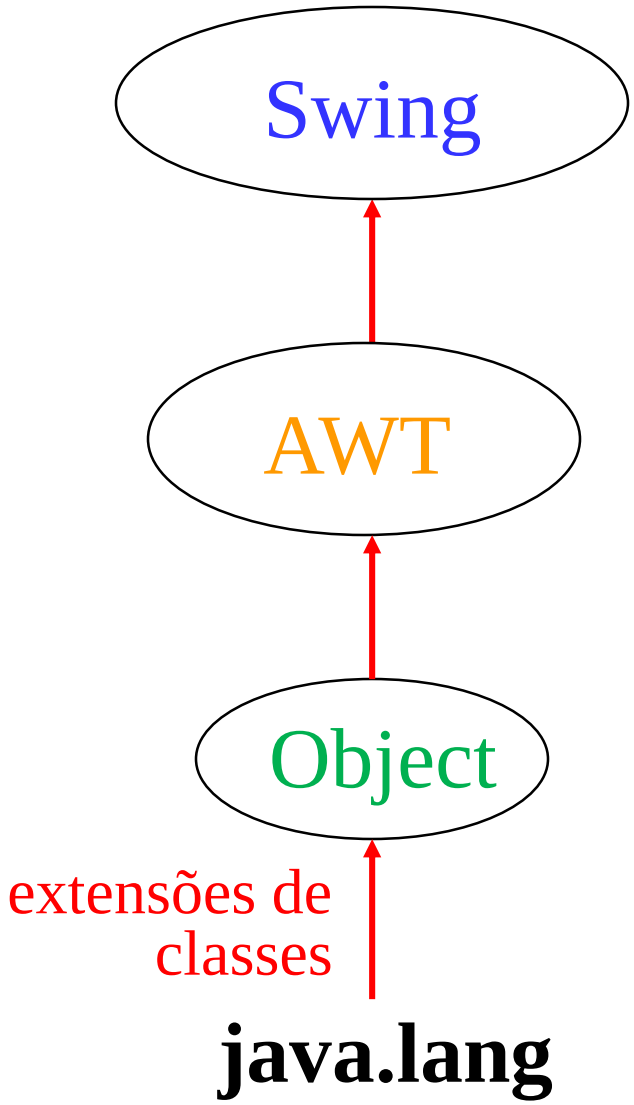
Histórico Evolutivo de Pacotes Gráficos

O pacote disponível em Java para criação de aplicações gráficas é conhecido como AWT (*Abstract Window Toolkit*).

A evolução dos recursos disponíveis neste pacote integrou uma nova classe no Java 2 denominada **Swing**:

- Totalmente desenvolvida em Java;
- Pertencente a *Java Foundation Classes* (JFC);
- Identificadores das classes Swing recebem o **J** (jota) ao nome original na AWT (Button => JButton);
- Interface implementada na Swing é mais independente do Sistema Operacional do que a AWT

Ambiente Gráfico



Estas extensões indicam que algumas importações serão necessárias ao desenvolvimento de aplicações gráficas "executáveis" (não é estudado ***applet*** nesta disciplina).

Geralmente, se farão presentes:

import java.awt.*;

import java.awt.event.*;

import javax.Swing.*;

Aplicação Gráfica

A elaboração de aplicações gráficas em Java consistem, basicamente, de 5 APIs da JFC que simplificam o desenvolvimento de aplicações com GUI, sendo elas:

- AWT;
- Swing;
- Java2D;
- Accessibility;
- Drag and Drop.

Estes recursos Java fornecem suporte ao desenvolvimento de aplicações visualmente interessantes e intuitivas, facilitando as atividades de seus usuários e podendo ser usados na criação de aplicações Java e *Applets* (conteúdo a ser estudado na próxima disciplina).

Aplicação Gráfica

O desenvolvimento de aplicações gráficas emprega recursos que possuem funcionalidades de contêineres, ou seja, que podem conter outros componentes destinados a geração de uma interface com o usuário.

Entre algumas das principais classes contêineres disponíveis na AWT têm-se:

- ***Component***: classe abstrata para objetos que podem ser exibidos ao usuário e raiz de todas as outras classes AWT
- ***Container***: subclasse abstrata de *Component* (componente que pode conter outros componentes)
- ***Panel***: herda a classe *Container*, correspondendo a uma área que pode ser inserida no *Frame*
- ***Frame***: Quadro ou janela completa que possui barra de menu, título, borda e cantos que podem redimensioná-la

Aplicação Gráfica

Componentes

Um componente (*Component*) da GUI corresponde a um objeto visual, criado por classes Java, que possibilita ao usuário realizar interação direta com a aplicação por meio de seu mouse e/ou teclado.

Cada um destes vários componentes são agrupados em contêineres (*Container*) e apresentados, geralmente, através de painéis (*Panel*) que compõem quadros (*Frame*) adequados para a interação almejada.

Estes componentes possuem propriedades próprias (tamanho, cor, fonte, etc.) que podem ser alteradas em tempo de desenvolvimento e/ou execução, conforme seja sua coerência com as expectativas de interação.

Aplicação Gráfica

Janela da Aplicação (JFrame)

O desenvolvimento de aplicações GUI utilizará a extensão da classe AWT na maioria dos recursos otimizados na Swing.

A criação de janelas gráficas, disponíveis no Windows, Solaris, Mac e outras plataformas, será realizada pela **JFrame** (extensão da Frame na AWT).

Este componente cria uma janela que possui os itens indicados anteriormente (título, barra de menu, etc.), podendo ainda possuir outros componentes em seu interior, similar a uma folha de projeto com vários objetos disponíveis para interação adequada do usuário.

Aplicação Gráfica

Principais Características (JFrame)

- Exige importação da javax.Swing para seu uso
- Janelas criadas pela JFrame são configuradas para não serem visíveis por padrão
- Altura e largura destas janelas são sempre zero, após sua criação, sendo necessário definir seu real tamanho
- JFrame são compostos por painéis de conteúdos que permitem uma interação organizada com os usuários, sendo os quadros contêineres de componentes como barra de menu e outros destacados diretamente nas janelas de interação
- Existem vários objetos que podem ser componentes internos desta janela, sendo neste material apresentados somente os comumente usados em aplicações gráficas

Aplicação Gráfica

Métodos Importantes no Uso da JFrame

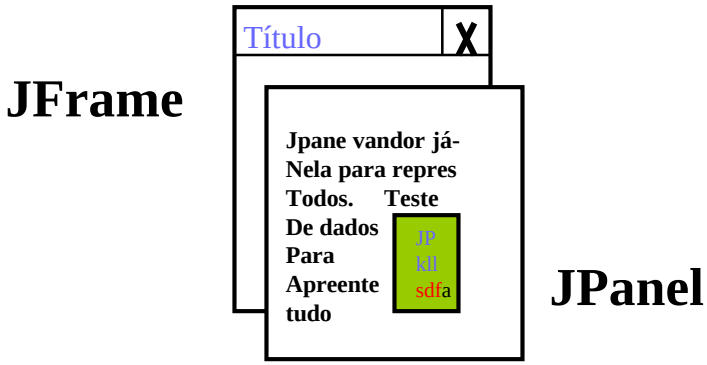
- **JFrame**: cria uma janela vazia
- **getTitle**: obtém o título da janela
- **setTitle(String)**: configura o título da janela
- **isResizable**: verifica se a janela é ou não dimensionável
- **setResizable(boolean)**: define janela dimensionável ou não
- **setIconeImage(Image)**: define ícone quando minimizado
- **setSize(int, int)**: define o tamanho da janela
- **setLocation(int,int)**: posiciona o componente na janela
- **setVisible(boolean)**: define se a janela é visível ou não
- **setBounds(int,int,int,int)**: posiciona e dimensiona a janela
- **show()**: método da classe Window que é herdado pela JFrame para apresentação da janela na aplicação (método *deprecated*)

Aplicação Gráfica

Painel (JPanel)

A elaboração de uma janela que possa apresentar texto e imagem, entre outros possíveis componentes, exige, para programação adequada, a criação de um ou vários painéis que também podem receber outros componentes (recipiente), sendo possível, inclusive, desenhar na superfície do mesmo.

Nesta camada, que sobrepõe o JFrame, é que esta o interesse dos programadores de aplicações gráficas em Java.



Aplicação Gráfica

O uso destes painéis permitem um posicionamento mais preciso dos componentes em uma janela gráfica, podendo os mesmos serem aninhados em uma mesma janela.

```
JPanel painel = new JPanel();  // cria um painel vazio

// add acrescenta um outro componente no painel
 painel.add(<outro componente>);
```

- O painel é o mecanismo de organização na interface do projeto gráfico
- Ele também possui características recipientes (recebem outros componentes), inclusive outros painéis (aninhar)
- As bordas dos painéis não são visíveis aos usuários

→ Método Importante: **JPanel** (**<layout principal no painel>**);

Aplicação Gráfica

Classe Font

A exibição de texto é possível no JPanel, sendo necessário primeiro a definição da fonte a ser usada por meio da classe Font e de métodos da classe Graphics.

```
Font texto = new Font("SansSerif",Font.PLAIN,12);
```

Nome da família da fonte

Tamanho da fonte (ponto)

Estilo indicado por:

normal => **Font.PLAIN**
negrito => **Font.BOLD**
itálico => **Font.ITALIC**
negrito e itálico => **Font.BOLD + Font.ITALIC**

Aplicação Gráfica

Classe Color

A configuração de cores no ambiente gráfico é possível com o uso da classe Color e Graphics, onde alguns de seus métodos permitem uma variação de cores de fundo (segundo plano) e do objeto (primeiro plano).

Várias opções podem ser usados para alterar as cores, sendo duas apresentadas a seguir:

- 13 constantes definidas para cores básicas
- 1 objeto Color com definição de intensidades que variam de 0-255 (1 byte) para as cores de vermelho, verde, azul (RGB)

Aplicação Gráfica

13 Constantes de cores

- black
- blue
- cyan
- darkGray
- gray
- green
- lightGray
- magenta
- orange
- pink
- red
- white
- yellow

Novo Objeto de Cor

```
Color(0,128,128);    // intensidades para azul esverdeado
```

vermelho

verde

azul

Aplicação Gráfica

Alguns Métodos da Classe Graphics

Para visualização das alterações de fonte e cores serão usados alguns métodos da classe Graphics, tais como:

- setFont(Font): define fonte para o contexto gráfico

  objetoGrafico.setFont(fonte);

- drawString (String, int, int): desenha String com fonte e cor atual objetoGrafico.drawString(string, x, y);

- setColor(Color): altera a cor atual para todas operações gráficas objetoGrafico.setColor(Color.green);

  ou objetoGrafico.setColor(new Color(0,0,128));

Classe Component objetoPainel.setBackground(Color)

- setBackground(Color): define cor do segundo plano
- setForeground(Color): define cor do primeiro plano

Aplicação Gráfica

```
/** Síntese
 *    Objetivo: saudar usuário
 *    Entrada:  nenhuma
 *    Saída:    saudação de bom dia
 */
import java.awt.*;
import javax.swing.*;
public class Janelas extends JFrame {
  public Janelas() {
    setTitle("Aplicação Gráfica");
    setSize(300,200);
    setBackground(Color.yellow);
    Container container = getContentPane();
    container.add(new PainelTexto());
    // Inseri imagem na barra de título
    Toolkit kit = Toolkit.getDefaultToolkit();
    Image imagem = kit.getImage("figura.jpg");
    setIconImage(imagem);
  }

  public static void main(String[] args) {
    JFrame janela = new Janelas();
    janela.setVisible(true);
  }
}
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    class PainelTexto extends JPanel {
      public void paintComponent(Graphics gra      {
          setForeground(Color.blue);
          Font fonte = new Font("SansSeri      nt.BOLD,14);
          grafico.setFont(fonte);
          grafico.drawString("Bom dia      0, 50);
    }
}
```

Aplicação não encerra.

- O **getContentPane** é um método que obtém as características de um painel e o **add** acrescenta um componente no objeto
- A classe Toolkit permite uma interação com o sistema operacional, enquanto o método **getDefaultToolkit** obtém seu estado atual e **getImage** retorna uma imagem que será lida de um arquivo
- A **Image** é a classe que permite a manipulação de imagens
- Método **setIconImage** pertence a JFrame e permite a inserção de uma imagem (ícone) no título da janela
- O método **paintComponent** permite o desenho na janela

Aplicação Gráfica

Observe no exemplo anterior que ao terminar a aplicação (opção fechar da janela) a mesma não é **encerrada** realmente, ficando em execução no computador (veja a situação na console).

A elaboração de uma aplicação gráfica necessita do acompanhamento das interações que o usuário, e o próprio processamento da aplicação, possam efetuar sobre os recursos disponível neste tipo de aplicação.

Este acompanhamento é realizado por meio de **eventos** que são assistidos pelos recursos implementados na própria aplicação (no programa).

Aplicação Gráfica

Um exemplo deste acompanhamento é o evento de pressionar o botão fechar da janela (**x**), onde a aplicação deverá realizar as operações necessárias para realmente encerrar sua execução e não somente ocultar a janela.

```
/** Síntese
 *    Objetivo: Mostrar uma janela e encerrar aplicação
 *    Entrada:  nenhuma
 *    Saída:    janela com opção de fechar funcionando
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Janela extends JFrame {
  Janela (String titulo) {   // construtor de Janela
    super(titulo);  // título na superclasse
    this.setBounds(100,100,250,250); // própria janela
    // cria objeto de uma classe interna anônima
    this.addWindowListener(new WindowListener() {
     // método anterior inclui acompanhamento na janela
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    public void windowActivated(WindowEvent evJanela) {
    }
    public void windowClosed(WindowEvent evJanela) {
    }
    public void windowClosing(WindowEvent evJanela) {
      fechaJanela();
    }
    public void windowDeactivated(WindowEvent evJanela){
    }
    public void windowDeiconified(WindowEvent evJanela){
    }
    public void windowIconified(WindowEvent evJanela) {
    }
    public void windowOpened(WindowEvent evJanela) {
    }
    // implementação de todos métodos da interface

  });  // termina classe anônima
  this.setVisible(true);
}
```

Aplicação Gráfica

```
// continuação do exemplo anterior
   public void fechaJanela() {
     System.exit(0);  // método que encerra o programa
   }
   public static void main(String args[]) {
     new Janela("Aplicação Gráfica");
   }
}
```

Observe na console que esta aplicação é encerrada realmente quando a janela é fechada. Para que isso acontecesse corretamente foi necessário acompanhar a ocorrência do evento e sobre qual objeto.

No entanto, o uso da interface WindowListener obriga a implementação de seus **7 métodos abstratos**, mas somente **1** é usado na aplicação (windowClosing).

Aplicação Gráfica

Classes Adaptadoras (Adapter)

Frequentemente é necessário implementar somente alguns métodos das interfaces que tem vários outros métodos a serem implementados pelas classes que as utilizam.

Esta situação é bastante trabalhosa e torna seu código extenso para leitura e manutenção. Mas Java disponibiliza uma classe (***Adapter***) que implementa os métodos que não serão usados nas interfaces que acompanham os eventos (*listener*).

Para isso, cada interface implementada deve possuir mais que um método, onde aqueles que não receberem sua nova codificação nesta classe serão implementados como vazios (similar aos outros 6 do exemplo anterior).

Aplicação Gráfica

Aplicando o conteúdo de estudo até o momento (classe interna anônima e adaptadoras) a codificação para manipulação deste evento (encerrar aplicação) pode ser simplificado.

```
public class Janela2 extends JFrame {
  Janela2 (String titulo) {   // método construtor
    super(titulo);
    setBounds(100,100,250,250);
    setVisible(true);
    // add inclui + <tipo do objeto> + acompanhamento
     addWindowListener(new WindowAdapter() {
      // usa classe interna anônima para criar 1 objeto
      // com a classe adaptadora só codifica o que é usado
      public void windowClosing(WindowEvent evJanela){
         System.exit(0); // encerra o programa
      }
     });
  }
  public static void main(String args[]) {
     new Janela2("Aplicação Gráfica");
  }
}
```

Aplicação Gráfica

- O método **add** inclui objetos a serem acompanhados quanto as suas ações (programa o estará ouvindo – *listener*)
- A sintaxe para esta inclusão é:

add <**tipo de objeto**> **Listener**

- Uma classe interna sem nome (anônima) gera um objeto que é passado como parâmetro para ser usado somente uma vez neste método
- Uma classe adaptadora é usada no controle desta janela gráfica (**WindowAdapter**), sendo somente o método que interessa implementado
- Os eventos desta janela são acompanhados (**WindowEvent**)

Aplicação Gráfica

Uma diversidade cada vez maior de classes e objetos são disponibilizados para o uso de componentes interessantes na interação com o usuário em uma aplicação gráfica. Alguns dos mais comuns e importantes serão abordados neste material.

Rótulo (JLabel)

A inclusão de rótulos ou etiquetas orientadoras ao apoio interativo da aplicação gráfica e seu usuário é realizada por meio do componente JLabel, que possui sobrecarga em seu método construtor, podendo envolver texto, imagem e alinhamento.

```
JLabel identificador = new JLabel(texto, JLabel.alinhar);
```

String do rótulo

Alinhamento (LEFT, RIGHT, CENTER)

Aplicação Gráfica

Uma outra situação para criação do JLabel pode apresentar uma imagem ou somente o texto do rótulo.

```
rotulo1 = new JLabel("Nome:");    // só o rótulo
rotulo2 = new JLabel("Endereço:",JLabel.RIGHT);
rotulo3 = new JLabel("Diagrama:",imagem,JLabel.CENTER);
```

A terceira instrução apresenta uma imagem definida anteriormente em uma declaração do tipo ImageIcon:

```
ImageIcon imagem = new ImageIcon("C:/temp/foto.gif");
```

Uma imagem, foto ou diagrama poderia ser apresentada junto ao rótulo para facilitar a compreensão do usuário do programa.

Aplicação Gráfica

```
/** Síntese
 *   Objetivo: Mostrar rótulos (etiquetas) na janela gráfica
 *   Entrada:  sem entrada
 *   Saída:    texto e imagem como rótulo da janela gráfica
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Rotulos extends JFrame {
  JLabel rotulo1, rotulo2, rotulo3;
  Container contem = getContentPane();
  Rotulos() {    // construtor
    setTitle("Aplicação Gráfica");
    setLocation(150,150);
    setSize(200,200);
    // objeto de manipulação de imagem
    ImageIcon figura = new ImageIcon(
                                   "C:/temp/diagrama.gif");
    contem.setBackground(Color.gray);
    rotulo1 = new JLabel("Etiqueta 1:");
    rotulo2 = new JLabel("Rótulo 2:",JLabel.RIGHT);
    rotulo2.setForeground(Color.red);
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    rotulo3 = new JLabel("Etiqueta 3:",
                                        figura,JLabel.CENTER);
    // configura gerenciador de layout
    contem.setLayout(new GridLayout(4,1));
    contem.add(rotulo1);
    contem.add(rotulo2);
    contem.add(rotulo3);
  }

  public static void main(String[] args) {
    JFrame janela = new Rotulos();
    janela.setVisible(true);
    // Método que encerra aplicação gráfica
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

Aplicação Gráfica

- **ImageIcon** permite a manipulação de imagens na aplicação gráfica
- **setLayout** corresponde a um método que permite a definição do layout de apresentação da janela gráfica
- **setDefaultCloseOperation** é um método que permite o acompanhamento sobre a janela gráfica usando algumas constantes:
  - **DO_NOTHING_ON_CLOSE** → não realiza nada
  - **HIDE_ON_CLOSE** → apenas esconde a janela
  - **DISPOSE_ON_CLOSE** → esconde e libera a janela
  - **EXIT_ON_CLOSE** → fecha a aplicação

Aplicação Gráfica

Caixa de Texto (JTextField)

Um outro componente importante a ser incluído em aplicações gráficas é o campo ou caixa de texto. Este componente coleta somente dados do tipo caracter na interação com o usuário, possuindo várias propriedades que podem ser configuradas para sua apresentação.

- Apesar da coleta ser somente de caracter, seu conteúdo pode ser transformado e armazenado em outros tipos de variáveis (int, double, boolean, Integer, etc.)
- A propriedade Editable pode tornar este componente somente de leitura para apresentação de dados somente
- Permite iniciar uma caixa de texto com um valor inicial
- Conforme o Layout adotado, ainda permite restringir a quantidade de caracteres a serem inseridos na caixa

Aplicação Gráfica

Métodos Importantes no Uso da JTextField

- **JTextField**: cria uma caixa de texto vazia
- **JTextField(String)**: cria caixa de texto iniciada com String
- **JTextField(String, int)**: cria caixa de texto com String e tamanho definido (quantidade de colunas)
- **JTextField(int)**: cria caixa de texto com tamanho específico
- **getText**: obtém o texto do objeto
- **getSelectedText**: obtém o texto selecionado do objeto
- **isEditable**: verifica se o componente é editável ou não
- **setEditable(boolean)**: define se caixa é editável ou não
- **setText**: especifica o texto a ser contido no componente

Aplicação Gráfica

Geralmente, a coleta de alguns dados pela caixa de texto exige alguma formatação. Por exemplo, a leitura de valores numéricos. Para realizar esta formatação (máscara) podem ser usadas diversas classes Java, tais como NumberFormat, DecimalFormat, entre outras.

```
import java.text.NumberFormat;
public class Pagamento {
    :
  double salario;
  NumberFormat valor1;     // define objeto formatado
      :
    valor1 = NumberFormat.getNumberInstance();
    valor1.setMinimumFractionDigits(3);
      :
    caixaTexto.setText("" + valor1.format(salario));
      :
}
```

Aplicação Gráfica

Caixa de Senha (JPasswordField)

As mesmas funcionalidades e propriedades da JTextField estão disponíveis na caixa de senha (JPasswordField), que inclui o recurso de ocultar os caracteres digitados por meio da apresentação de um outro caracter fixo. Este caracter que oculta a real digitação é definido por padrão como * (asterisco), mas pode ser alterado.

Métodos Importantes no Uso da JPasswordField

- **JPasswordField**: cria uma caixa de senha vazia
- **setEchoChar(char)**: define um caracter que ocultará os valores realmente digitados

Aplicação Gráfica

Área de Texto (JTextArea)

Um componente semelhante a caixa de texto é a Área de Texto (JTextArea) que permite o tratamento de várias linhas na informação a ser solicitada ao usuário.

Alguns métodos são comuns a estes dois componentes, como getText e setText, além de um texto inicial dentro do mesmo. Porém algumas funcionalidades, como salto de linha na própria caixa (**\n**) é possível somente no JTextArea.

Forma Geral:

```
JTextArea (string, número de linhas, caracter por linha);
```

opcional um texto inicial

número de linha mostrada na tela

dimensão em número de caracteres

Aplicação Gráfica

Métodos Importantes no Uso da JTextArea

- **JTextArea(String, int, int)**: cria uma área de texto
- **getSelectedText**: obtém o texto selecionado na área de texto
- **insert(String, int)**: inseri a string na posição inteira indicada
- **replaceRange(String, int, int)**: métodos JTextField que funcionam em JTextArea
- **append(String)**: inseri um texto no final da área de texto
- **getColumns**: obtém comprimento da área de texto
- **getRows**: obtém largura da área de texto
- **setColumns**: define colunas da área de texto
- **setRows**: define linhas na área de texto
- **setLineWrap(boolean)**: define quebra de linha automática

Aplicação Gráfica

```
/** Síntese
 *    Objetivo: analisa a senha
 *    Entrada:  senha
 *    Saída:    situação de acesso, senha
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class AveriguaSenha extends JFrame
                                  implements ActionListener {
  JLabel etiq1,etiq2,etiq3;
  JTextArea area;
  JPasswordField senha;
  JTextField texto;
  // Obtém aspectos do painel padrão
  Container conteiner = getContentPane();

  AveriguaSenha ()    // construtor
  {
    setBounds(100,100,250,250);
    setTitle("Analisa Senha");
    conteiner.setBackground(new Color(150,150,150));
    conteiner.setLayout(new GridLayout(3,2));
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    // Cria primeira etiqueta
    etiq1 = new JLabel("Digite a senha: ");
    etiq1.setForeground(Color.black);
    etiq1.setFont(new Font("SansSerif",Font.BOLD,14));
    // Cria segunda etiqueta
    etiq2 = new JLabel("Situação: ");
    etiq2.setForeground(Color.black);
    etiq2.setFont(new Font("",Font.BOLD,14));
    // Cria terceira etiqueta
    etiq3 = new JLabel(" => Senha");
    etiq3.setForeground(Color.CYAN);
    etiq3.setFont(new Font("",Font.ITALIC,14));
    // Cria campos de texto
    area = new JTextArea();
    texto = new JTextField();
    texto.setEnabled(false);
    texto.setFont(new Font("Arial",Font.ITALIC,18));
    senha = new JPasswordField();
    senha.setEchoChar('?');
    senha.addActionListener(this);
```

Aplicação Gráfica

```
// continuação do exemplo anterior

    // Inclui no painel padrão do conteiner
    conteiner.add(etiq1);
    conteiner.add(senha);
    conteiner.add(etiq2);
    conteiner.add(area);
    conteiner.add(texto);
    conteiner.add(etiq3);
  }

  public static void main(String[] args) {
    JFrame janela = new AveriguaSenha();
    janela.setVisible(true);
    janela.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent evJanela) {
         System.exit(0); //método que encerra o programa
      }
    });     // termina a classe interna anônima
  }
```

Aplicação Gráfica

```
// continuação do exemplo anterior
  public void actionPerformed(ActionEvent evento)
  {
    if(senha.getText().equals("Java")) {    // senha
      area.setForeground(Color.blue);
      area.setText("Senha válida\n");
      texto.setText(senha.getText());
    }
    else {
      area.setForeground(Color.red);
      area.append("Senha inválida\n");
      senha.setText("");     // limpa entrada da senha
    }
  }
}
```

Aplicação Gráfica

Gerenciadores de Layout

Em alguns programas anteriores pode ser observado o uso do método **setLayout**. Por meio dele é possível definir layouts de apresentação das aplicações gráficas em Java.

Estas apresentações são elaboradas através da definição de Container (JFrame e JPanel por exemplo). Para cada um deles deve ser **definido um layout** ou os mesmos possuirão somente uma célula com capacidade de armazenar somente um objeto.

Portanto, os objetos não são inseridos nestes componentes com coordenadas *X* e *Y*, mas inseridos **coerentemente nos gerenciadores de layouts** definidos por meio deste método.

Aplicação Gráfica

Os principais gerenciadores de layout em Java são:

FlowLayout:

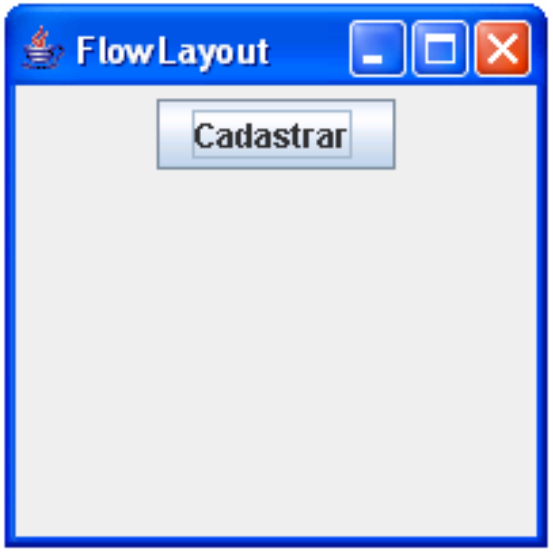
- Componentes inseridos da esquerda para direita na ordem em que são adicionados pelo método **add**
- Quando não existe espaço na linha atual, é automático o salto para próxima linha disponível na janela definida
- Possui variação em seu método de criação para definir
  - Alinhamento dos componentes no layout (0-esquerda, 1-central, 2-direita)
  - Espaçamento horizontal entre componentes em uma mesma linha é 5 por padrão, mas pode ser alterado
  - Espaçamento vertical é a distância entre as linhas no contêiner e por padrão também corresponde a 5 unidades

Aplicação Gráfica

Exemplo:

```
<Container>.setLayout(new FlowLayout(1,20,40));
```

- alinhamento será centralizado;
- espaçamento horizontal (largura) de 20 unidades
- espaçamento vertical (altura) de 40 unidades



Aplicação Gráfica

GridLayout:

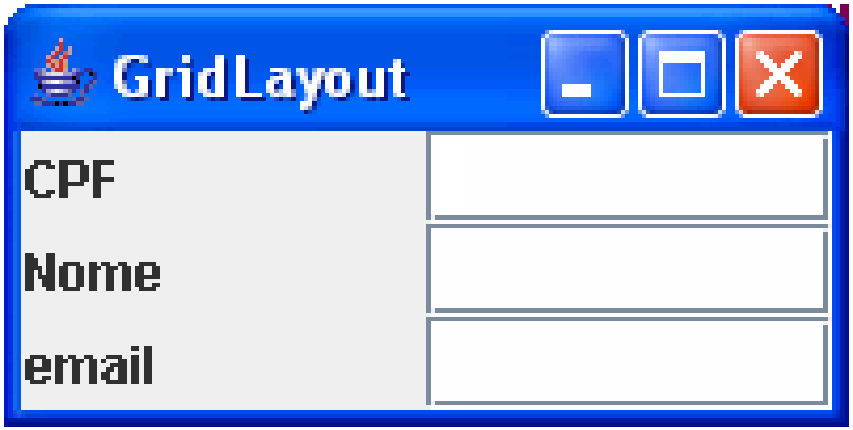
- Divide um layout em várias células (linha, coluna) de mesma dimensão para todos seus componentes
- Distribui os componentes conforme são inseridos, iniciando da esquerda para direita e de cima para baixo
- Componentes são dimensionados de acordo com o tamanho da célula e todos eles sofrem interferência do tamanho da janela, podendo ser redimensionados
- Cuida do alinhamento de seus componentes inseridos nas células
- Possui variação em seu método de criação para definir
  - Número de linhas – padrão é 1 se não for definido
  - Número de colunas – o padrão também é 1
  - Espaçamento horizontal e vertical – opcional e exatamente igual a definição do **FlowLayout**

Aplicação Gráfica

Exemplo:

```
<Container>.setLayout(new GridLayout(3,2,20,40));
```

- número de linha: 3
- número de coluna: 2
- espaçamento horizontal (largura) de 20 unidades
- espaçamento vertical (altura) de 40 unidades



Aplicação Gráfica

BorderLayout:

- Divide o layout em 5 regiões (**North, South, West, East, Center**) que só recebem um único componente cada uma, não sendo a ordem de inserção relevante nesta layout
- Um painel pode ser formado por vários componentes e ser inserido em um região com todos seus componentes
- A inserção de mais que um componente na mesma região sobrepõe os componentes
- Similar ao GridLayout é redimensionável pela janela
- Método **add** diferente, pois exige também qual a região
- Todas as regiões não precisam ser preenchidas, podendo as regiões vazias serem preenchidas com um componente qualquer (sem indicação da região no **add**)
- Possui variação em seu método de criação para definir somente o Espaçamento horizontal e vertical, como as definições do FlowLayout e GridLayout

Aplicação Gráfica

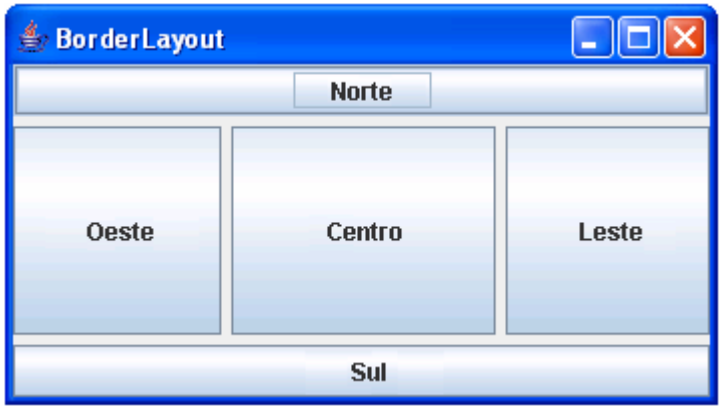
Exemplo:

```
<Container>.setLayout(new BorderLayout(20,40));
```

- espaçamento horizontal (largura) de 20 unidades
- espaçamento vertical (altura) de 40 unidades

Variação do método add com a indicação da região:

```
<Container>.add("North" , etiq1);
<Container>.add("East" , campo1);
// inseri campo ocupando as outras 3 regiões
<Container>.add(campo2);
```



Aplicação Gráfica

CardLayout:

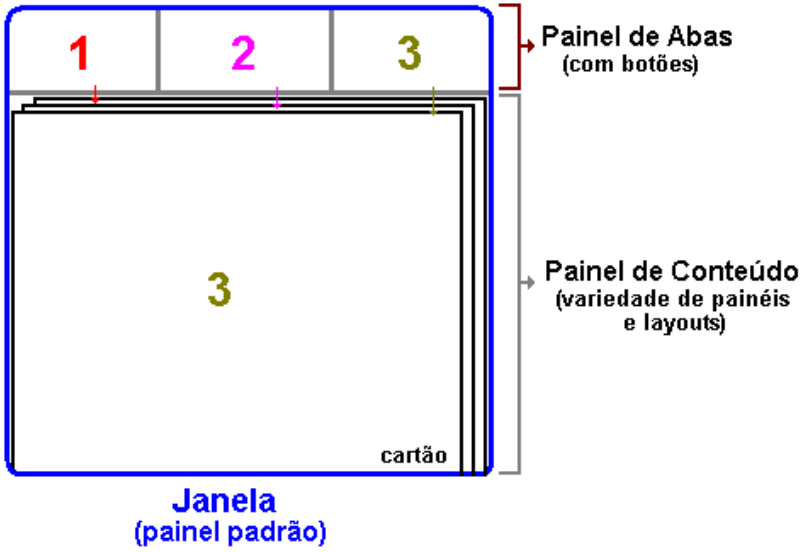
- Pode agrupar vários contêineres na forma de cartões, mostrando um de cada vez conforme seja selecionado
- Cada contêiner pode possuir seu layout específico
- Permite que diversos layouts sejam usados em um mesmo espaço da janela, similar as abas de pastas
- Apresenta vários tipos de painéis em uma mesma janela e navega entre eles através dos métodos do CardLayout
- A ordem dos painéis depende da ordem em que forem inseridos

Aplicação Gráfica

Exemplo:

```
<Container>.setLayout(new CardLayout(20,40));
```

- espaçamento horizontal (largura) de 20 unidades
- espaçamento vertical (altura) de 40 unidades
- normalmente o Container é um painel definido
- O painel CardLayout usa o método **add** para inserir painéis em seus cartões




```
<Panel>.add(painel, "P1");
```

painel criado na aplicação

nome qualquer vinculado ao painel

Aplicação Gráfica

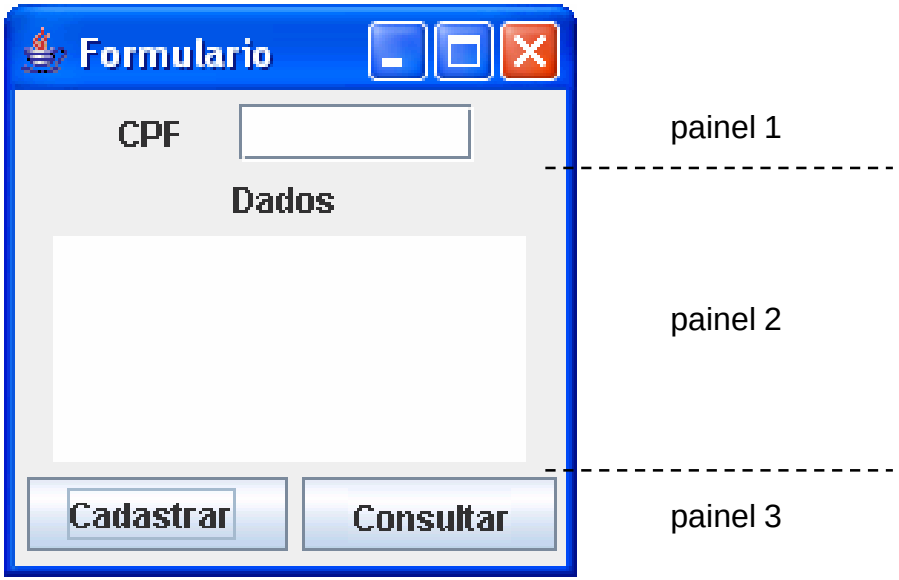
Métodos Importantes no Uso do CardLayout

- **CardLayoyt**: cria layout sem espaçamento entre os painéis
- **first(container)**: mostra o primeiro painel inserido ao CardLayout
- **last(container)**: mostra o último painel inserido ao CardLayout
- **previous(container)**: mostra componente anterior inserido ao CardLayout
- **next(container)**: exibe o próximo componente
- **show(container,"string")**: exibe o componente especificado pela string fornecida

Aplicação Gráfica

A análise da interação e do seu conteúdo a ser tratado com o usuário é relevante na identificação de qual layout seria mais adequado na aplicação gráfica desejada.

No entanto, a elaboração de um layout adequado, geralmente, faz uso da **combinação** entre os mesmos, **utilizando painéis diferentes** sobre uma mesma janela.



Aplicação Gráfica

Botão (JButton)

Um componente interessante são os botões nas janelas gráficas, sendo os mesmos implementados pela JButton. Uma variedade de propriedades podem ser manipulados por este tipo de componente.

```
JButton botao = new JButton("Cancelar"); // cria botão
JButton outroBotao = new JButton("Confirmar", imagem);
```

No botão **outroBotao** são apresentados um rótulo de orientação da funcionalidade do botão e uma imagem definida, como está apresentado no exemplo do **JLabel** anteriormente.

Aplicação Gráfica

Controlador de Evento

A utilização da maioria dos componentes de interação gráfica precisam estar atrelados ao controle de eventos que podem ocorrer sobre eles.

A compreensão desta necessidade pode ser alcançada facilmente sobre o componente botão, pois o mesmo, estando disponível, possibilitará que o usuário da aplicação o pressione, efetuando assim um evento sobre este componente (o botão).

É importante a compreensão dos **eventos** que podem ser realizados sobre os componentes de uma aplicação gráfica, pois serão por meio deles que esta aplicação realizará processamentos desejados pelo usuário.

Aplicação Gráfica

As aplicações que precisam controlar eventos em Java necessitam implementar suas correspondentes **interfaces** que efetuam a recepção do evento (acompanhamento).

Os objetos que poderão gerar certos eventos também precisam registrar tais eventos para que possam ser acompanhados e controlados.

Cada objeto ou classe pode implementar quantos receptores de evento forem necessários. Por exemplo: num objeto JButton pode ser acompanhado quando:

- mouse entra ou sai de sua área gráfica;
- mouse clica sobre o botão;
- mouse solta seu botão que foi clicado

$\Rightarrow$ entre outros eventos possíveis de serem acompanhados.

Aplicação Gráfica

Cada classe acompanhadora de evento trata de um evento específico, sendo os mais comuns da Swing apresentadas a seguir:

- **ActionListener**: evento de ação como pressionar um botão existente na janela gráfica
- **AdjustmentListener**: evento de ajuste gerado por um componente
- **FocusListener**: componente recebe ou perde o foco
- **ItemListener**: quando um item de uma lista é alterado
- **KeyListener**: evento sobre o teclado (tecla pressionada, solta e outros sobre o teclado)
- **MouseListener**: clique, entrada ou saída do mouse na área do componente gráfico
- **MouseMotionListener**: movimentação do mouse sobre um componente gráfico disponível na janela

Aplicação Gráfica

- **WindowListener**: manipular aspectos da janela
- **TextListener**: alterar o conteúdo de um campo texto
- **ComponentListener**: movimento, tornar visível, redimensionar qualquer componente de uma janela

As classes receptoras de evento trabalham junto com os objetos que devem ser registrados como geradores de eventos, sendo seus métodos de registro :

- **addActionListener**
- **addItemListener**
- **addFocusListener**
- **addAdjustmentListener**
- **addMouseListener**
- **addMouseMotionListener**
- **addKeyListener**
- **addWindowListener**
- **addTextListener**
- **addComponentListener**

Aplicação Gráfica

A implementação das interfaces adequadas aos respectivos eventos devem implementar os métodos definidos na mesma para trabalharem corretamente.

ActionListener

- Evento: ActionEvent
- Método: **actionPerformed**
  - executado com clique do mouse sobre o componente gráfico ou pressionar a tecla ENTER quando estiver nele
  - JButton, JCheckBox, JComboBox, JTextField, JRadio-Button

AdjustmentListener

- Evento: AdjustmentEvent
- Método: **adjustmentValueChanged**
  - executado quando valor de um componente é alterado
  - JScrollBar

Aplicação Gráfica

ComponentListener

- Evento: ComponentEvent
- Métodos
  - **componentHidden**: executado quando componente torna-se oculto (escondido)
  - **componentMoved**: executado quando componente é movido
  - **componentResized**: executado quando componente é redimensionado
  - **componentShown**: executado quando componente torna-se visível

  - Empregado por todos os componentes Swing

Aplicação Gráfica

FocusListener

- Evento: FocusEvent
- Métodos
  - **focusGained**: executado quando se recebe o foco
  - **focusLost**: executado quando se perde o foco
  - Empregado por todos os componentes Swing

KeyListener

- Evento: KeyEvent
- Métodos: **keyPressed**: executado quando uma tecla é pressionada e o foco esta no componente
  - **keyReleased**: executado quando uma tecla é solta sobre o componente que está em foco (selecionado)
  - **keyTyped**: executado quando uma tecla *Unicode* é pressionada sobre o componente, não sendo válidas *Shift*, *Alt*, *Ctrl*, *Insert*, setas e outras que não sejam codificadas
  - Empregado por todos os componentes Swing

Aplicação Gráfica

MouseListener

- Evento: MouseEvent
- Métodos
  - **mousePressed**: executado quando botão do mouse é pressionado sobre o componente selecionado (em foco)
  - **mouseClicked**: executado quando botão do mouse é solto sobre o componente selecionado
  - **mouseEntered**: executado quando apontador do mouse entra na área do componente gráfico
  - **mouseExited**: executado quando apontador do mouse sai da área do componente gráfico
  - **mouseReleased**: executado quando mouse é arrastado sobre um componente gráfico

  - Empregado por todos os componentes Swing

Aplicação Gráfica

MouseMotionListener

- Evento: MouseEvent
- Métodos
  - **mouseMoved**: executado quando apontador do mouse se move sobre um componente gráfico
  - **mouseDragged**: executado enquanto o apontador do mouse é arrastado sobre um componente gráfico
  - Empregado em todos os componentes Swing

TextListener

- Evento: TextEvent
- Método: **textValueChanged**
  - executado quando valor do componente de texto é alterado (modificado)
  - TextComponent (inválido para Swing – JTextField, ...)

Aplicação Gráfica

WindowListener

- Evento: WindowEvent
- Métodos
  - **windowClosing**: executado enquanto a janela está fechando
  - **windowClosed**: executado após janela ter fechado
  - **windowActivated**: executado quando janela é ativada
  - **windowDeactivated**: executado quando janela é desativada
  - **windowIconified**: executado quando janela é minimizada
  - **windowDeiconified**: executado quando janela é restaurada
  - **windowOpened**: executado quando janela é aberta pelo método *show()* ou *setVisible()*
  - JWindow, JFrame

Aplicação Gráfica

Os métodos anteriores executam quando um evento ocorre, porém outros métodos permitem averiguar outras características na ocorrência de certos eventos. Entre estes observe os eventos relacionados ao mouse e ao teclado.

Mouse

- Evento: MouseEvent
  - **int getClickCount**: retorna o número de cliques do mouse
  - **int getX**: retorna posição X do apontador do mouse
  - **int getY**: retorna posição Y do apontador do mouse
  - **boolean isPopupTrigger**: retorna verdadeiro se evento gerado resultará na abertura de menu Popup
  - **boolean isAltDown**: retorna verdadeiro se a tecla **Alt** estava pressionada quando mouse foi clicado
  - **boolean isControlDown**: retorna verdadeiro se a tecla **Crtl** estava pressionada quando mouse foi clicado
  - **boolean isShiftDown**: retorna verdadeiro se a tecla **Shift** estava pressionada quando mouse foi clicado

Aplicação Gráfica

Teclado

- Evento: KeyEvent
  - **int getKeyChar**: retorna o caracter Unicode relacionado ao evento
  - **setKeyChar**: gera o caracter Unicode relacionado ao evento
  - **int getKeyCode**: retorna o código do caracter relacionado ao evento
  - **setKeyCode**: gera o código do caracter relacionado ao evento, por exemplo: simular o teclar da ESC pode ser usado *setKeyCode(e.VK_ESCAPE)* (código 27)
  - **boolean isAltDown**: funciona exatamente como abordado no mouse (tecla **Alt**)
  - **boolean isControlDown**: funciona como no mouse (**Ctrl**)
  - **boolean isShiftDown**: funciona como no mouse (**Shift**)

Aplicação Gráfica

Métodos Importantes no Uso da JButton

- **JButton**: cria um botão na janela gráfica
- **getLabel**: obtém o rótulo do botão
- **setLabel(String)**: define rótulo do botão
- **setEnabled(boolean)**: define se botão esta habilitado ou não
- **setHorizontalTextPosition**: define alinhamento horizontal do rótulo em relação a sua imagem (LEFT ou RIGTH)
- **setVerticalTextPosition**: define alinhamento vertical do rótulo em relação a sua imagem (TOP ou BOTTOM)
- **setToolTipText**: define uma mensagem a ser mostrada quando mouse se posicionar sobre o botão
- **setMnemonic**: define uma letra como atalho para o botão

Aplicação Gráfica

```
/** Síntese
 *    Objetivo: mostra o código de caracteres
 *    Entrada:  caracteres
 *    Saída:    códigos
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ExerBotao extends JFrame implements
 ActionListener, KeyListener {
   JTextField letra, codigo, mostra;
   JButton botaoFim;
   Container contem;
   ExerBotao() {   // construtor
   super("Conversor");
   setBounds(300,100,200,200);
   setResizable(false);
   Font fonte = new Font("SansSerif",Font.BOLD,16);
   contem = getContentPane();
   contem.setLayout(new GridLayout(4,1));
   letra = new JTextField();
   letra.addKeyListener(this);
```

Aplicação Gráfica

```
// continuação do exemplo anterior
   codigo = new JTextField();
   codigo.setEditable(false);    // não permite escrita
   mostra = new JTextField();
   mostra.setEnabled(false);     // desabilita o campo
   mostra.setFont(fonte);

   botaoFim = new JButton("Sair");
   botaoFim.setEnabled(true);
   botaoFim.addActionListener(this);
   botaoFim.setToolTipText("Pressione botão para sair.");
   botaoFim.setMnemonic(KeyEvent.VK_S);

   // sequencia de inserção no layout
   contem.add(letra);
   contem.add(codigo);
   contem.add(mostra);
   contem.add(botaoFim);
 }
```

Aplicação Gráfica

```
// continuação do exemplo anterior
 public void actionPerformed (ActionEvent acao) {
   if(acao.getSource() == botaoFim) {
     System.out.println("Pressionado botão Fim.");
     System.exit(0);  // encerra a aplicação gráfica
   }
 }
 public void keyPressed(KeyEvent evento) {
   codigo.setText("" + evento.getKeyCode());
 }
 public void keyReleased(KeyEvent evento) {
   mostra.setText(letra.getText());
   letra.setText("");
 }
 public void keyTyped(KeyEvent evento) {
    // método sem implementação
 }
 public static void main(String[] args) {
   JFrame janela = new ExerBotao();
   janela.setVisible(true);
   janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
}
```

Exercício de Fixação

1) Analisando o exemplo anterior você deverá criar um novo botão neste mesmo tipo de layout com a funcionalidade de limpar os três campos existentes. A aplicação continuará funcionando, mas sem valores nenhum.

Após corrigir a aplicação para aceitar mais um botão neste formato de apresentação do programa anterior, deverá ser elaborado um método que será acionado quando este novo botão for pressionado e os três campos de texto deverão ser limpos.

Similar ao registro do botão Sair este novo botão (Limpar) deverá registrar seu pressionamento na console Java todas as vezes que o mesmo for pressionado. Inclua também uma mensagem instrutiva quando o mouse estiver sobre ele.

Aplicação Gráfica

Caixa de Seleção (JCheckBox)

Este componente é usado para facilitar o fornecimento de dados pelo usuário, onde o mesmo pode selecionar ou não uma sugestão da informação desejada.

Forma Geral:

```
JCheckBox (string, situação da seleção);
```

texto da opção a ser selecionada

true indica seleção e **false** não selecionado

- A interface ItemListener deve ser implementada para acompanhamento dos eventos deste objeto
- **itemStateChanged** é o método que será executado para alterar a situação do objeto
- **getStateChange** método que verifica o estado do objeto

Aplicação Gráfica

Métodos Importantes no Uso da JCheckBox

- **JCheckBox**: cria uma caixa de seleção na janela gráfica
- **getSource**: obtém qual objeto gerou o evento
- **itemStateChanged(ItemEvent)**: método executado quando a situação do objeto é modificada
- **getStateChange**: obtém estado atual do objeto, com retorno do tipo boolean (ItemEvent.SELECTED ou ItemEvent.DESELECTED
- **setSelected**: define o estado do JChexkBox, sendo o padrão DESELECTED

Aplicação Gráfica

Painél e Botões de Rádio (JRadioButton)

Os botões de rádio devem estar sempre agrupados para que seja escolhido somente um entre os possíveis, estando este objeto em um painel adequado.

Forma Geral:

```
JRadioButton(string, seleção);
```

texto específico de cada botão

cria um botão de rádio com **true** ou **false** para seleção

```
ButtonGroup(string);
```

nome do grupo dos botões de rádio

- Cada botão de rádio pode ser criado com texto, uma imagem, com ambos e ser indicado se esta selecionado
- O grupo criado deve ser iniciado para conter um conjunto de botões de rádio (operador new ButtonGroup)

Aplicação Gráfica

Métodos Importantes no Uso da JRadioButton

- **JRadioButton**: cria botão de rádio com seu texto
- **ButtonGroup**: cria um grupo para botões de rádio
- **setSelected(boolean)**: define a seleção do botão ou não
- **JPanel**: cria um painel para janela gráfica
- **<nome do painel>.add(<nome do objeto>)**: adiciona objetos em um determinado painel
- **<nome do grupo>.add(<nome do botão de rádio>)**: adiciona cada botão de rádio em um grupo específico

Aplicação Gráfica

Lista de Seleção (JList)

Uma lista de seleção permite a escolha de um ou vários valores armazenados em uma lista de opções. Sua implementação é possível por meio de algumas definições de recursos e componentes na aplicação, além da importação **javax.swing.event.***.

Para funcionalidade correta da JList é importante:

- Definir um objeto **DefaultListModel** que conterá todas as opções da lista
- Implementar a interface **ListSelectionListener** para que o usuário possa selecionar itens, sendo obrigatória a definição do método **valueChanged** para isso
- Para navegar entre suas várias opções, esta lista deve ser adicionada em um painel de rolagem (com **JScrollBar**)

Aplicação Gráfica

A múltipla seleção é possível neste componente, podendo ser realizada de duas formas:

- contínua: seleção em sequência (com tecla **Shift**)
- alternada: seleção aleatória (com tecla **Ctrl**)

Para implementar a múltipla seleção o método **setSelectionMode**(<modo>) deve ser utilizado, sendo:

- **1** para seleção em seqüência
- **2** para seleção alternada

É importante destacar que a múltipla seleção exige o uso de array, onde o retorno de **setSelectIndex(int [ ])** será um array de índices selecionados.

Forma Geral:

```
JList(modelo da lista);
```

DefaultListModel → classe que define um modelo de lista

Aplicação Gráfica

Métodos Importantes no Uso da JList

- **JList**: cria uma lsita de seleção
- **getSelectedValue**: obtém o texto do item selecionado
- **getSelectedIndex**: obtém o índice do item selecionado
- **setSelectedIndex(int)**: seleciona o índice indicado
- **setSelectedInterval(int, int)**: seleciona índices no intervalo
- **isSelectionEmpty**: verifica item selecionado (retorno lógico)
- **isSelectedIndex(int)**: confirma se índice indicado está selecionado na lista, retornando um valor lógico (boolean)
- **addElement(String)**: adiciona texto como novo item da lista
- **getSize**: obtém número total de opções existentes na lista
- **remove(int)**: remove item da lista indicado pelo índice
- **valueChanged**: método acionado quando item é selecionado
- **JScrollPane**: objeto que agrega um painel com barra de rolagem

Aplicação Gráfica

Caixa de Seleção (JComboBox)

A caixa de seleção tem funcionalidade similar a lista de seleção, mas se apresenta de outra forma interativa na janela gráfica e permite só uma seleção de item.

Outro aspecto importante são os métodos diferentes usados na **JComboBox** em relação a **JList**.

- Implementa a interface **ItemListener** para reconhecer a mudança da opção selecionada neste componente
- O método **itemStateChanged**, implementado em **ItemListener**, será acionado durante o evento de escolha de um item neste componente

Forma Geral:

```
JComboBox(opções de seleção);
```

`opções de seleção` → corresponde a um **array** do tipo String

Aplicação Gráfica

Métodos Importantes no Uso da JComboBox

- **JComboBox(String [ ])**: cria uma caixa de seleção com array do tipo String
- **getSelectedItem**: obtém o texto do item selecionado
- **getItemCount**: obtém o número total de item disponíveis
- **getSelectedIndex**: obtém o índice do item selecionado
- **removeItemAt(int)**: remove o item com o índice indicado
- **removeAllItems**: remove todos os itens da lista
- **addItem(String)**: adiciona texto como novo item disponível
- **itemStateChanged**: método acionado quando item é selecionado

Aplicação Gráfica

```
/* Síntese
 *   Objetivo: exemplo de Card Layout e componentes
 *   Entrada:  diversos, conforme Card apresentado
 *   Saída:    dados do contexto de cada Card Layout
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class PainelCard extends JFrame implements
 ActionListener, ItemListener, ListSelectionListener {
  float valor=0, total=0;   // usadas no painel 1
  JPanel painelBotoes,painelCard, P1,P11,P12,P2,P22,P3;
  JButton B1,B2,B3,B4,B5,B6,btQtde, btIndice, btLimpar,
                        btPainel1, btPainel2, btPainel3;
  JLabel L1, L11, L12, L2, L21, L3, L31;
  JTextField T11, T12, T21, T31, T32;
  JRadioButton radio1, radio2, radio3;
  ButtonGroup radioGroup;
  DefaultListModel listModel;
  JList lista;
  JComboBox combo;
```

Aplicação Gráfica

```
// continuação do exemplo anterior

  // Dados do painel padrão - Cards e botões
  Container contem = getContentPane();

  public static void main(String args[]) {
    JFrame janela = new PainelCard();
    janela.setVisible(true);
    janela.addWindowListener (new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
  }

  PainelCard() {   // construtor
    // Define propriedades do layout padrão da janela
    setTitle("Gerenciador CardLayout");
    setBounds(100,100,300,351);
    contem.setBackground(new Color(0,128,128));
    contem.setLayout(new BorderLayout());

    radio1 = new JRadioButton("10% do valor");
    radio1.addItemListener(this);
    radio2 = new JRadioButton("20% do valor");
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    radio2.addItemListener(this);
    radio3 = new JRadioButton("30% do valor");
    radio3.addItemListener(this);
    radio1.setMnemonic(KeyEvent.VK_1);
    radio2.setMnemonic(KeyEvent.VK_2);
    radio3.setMnemonic(KeyEvent.VK_3);

    B1 = new JButton("Mostra Texto");
    B1.addActionListener(this);
    B2 = new JButton("Mostra Índice");
    B2.addActionListener(this);
    B3 = new JButton("Adiciona item");
    B3.addActionListener(this);
    B4 = new JButton("Remove item");
    B4.addActionListener(this);
    B5 = new JButton("Remove todos");
    B5.addActionListener(this);
    B6 = new JButton("Quant. Itens");
    B6.addActionListener(this);
    btQtde = new JButton("Quantidade de Itens");
    btQtde.addActionListener(this);
    btIndice = new JButton("Índice selecionado");
```

Aplicação Gráfica

```
// continuação do exemplo anterior

    btIndice.addActionListener(this);
    btLimpar = new JButton("Apaga item");
    btLimpar.addActionListener(this);

    L1 = new JLabel("PAINEL 1 ",JLabel.CENTER);
    L2 = new JLabel("PAINEL 2",JLabel.CENTER);
    L3 = new JLabel("PAINEL 3",JLabel.CENTER);
    L11 = new JLabel("Digite um valor");
    L11.setForeground(Color.blue);
    L12 = new JLabel("% do valor", JLabel.RIGHT);
    L12.setForeground(Color.blue);
    L21 = new JLabel("Sem seleção");
    L21.setForeground(Color.black);
    L31 = new JLabel("Conteúdo");
    L31.setForeground(Color.blue);
    L31.setFont(new Font("Arial",Font.BOLD,15));

    T11 = new JTextField(5);
    T12 = new JTextField(5);
    T12.setEditable(false);
    T21 = new JTextField();
    T21.addActionListener(this);
```

Aplicação Gráfica

```
// continuação do exemplo anterior

    T31 = new JTextField();
    T32 = new JTextField();

    listModel = new DefaultListModel();
    listModel.addElement("Abaxaxi");
    listModel.addElement("Abacate");
    listModel.addElement("Banana");
    listModel.addElement("Maça");
    listModel.addElement("Mamão");
    listModel.addElement("Melão");
    listModel.addElement("Melancia");
    listModel.addElement("Pera");
    listModel.addElement("Uva");
    lista = new JList(listModel);
    lista.addListSelectionListener(this); //registra lista

    btPainel1 = new JButton("Painel 1");
    btPainel1.addActionListener(this);
    btPainel2 = new JButton("Painel 2");
    btPainel2.addActionListener(this);
    btPainel3 = new JButton("Painel 3");
    btPainel3.addActionListener(this);
```

Aplicação Gráfica

```
// continuação do exemplo anterior

    // Define painéis que formarão o layout padrão
    painelCard = new JPanel();
    painelCard.setLayout(new CardLayout());
    painelBotoes = new JPanel();
    painelBotoes.setLayout(new GridLayout(1,3));

    // Define painel P1 - primeira aba
    P1 = new JPanel();
    P11 = new JPanel();
    P12 = new JPanel();
    P1.setLayout(new FlowLayout(FlowLayout.CENTER));
    radioGroup = new ButtonGroup();
    radioGroup.add(radio1);
    radioGroup.add(radio2);
    radioGroup.add(radio3);
    P1.add(L1);
    P12.add(radio1);
    P12.add(radio2);
    P12.add(radio3);
    P11.add(T11);
    P12.add(T12);
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    P11.add(L11);
    P12.add(L12);
    P11.setLayout(new FlowLayout(FlowLayout.CENTER));
    P11.setBackground(new Color(200,200,200));
    P12.setLayout(new GridLayout(2,3));
    P12.setBackground(new Color(200,200,200));
    P11.add(L11);
    P11.add(T11);
    P12.add(radio1);
    P12.add(radio2);
    P12.add(radio3);
    P12.add(L12);
    P12.add(T12);
    P1.add(P11);
    P1.add(P12);
    // Define painel P2 - segunda aba
    P2 = new JPanel();
    P2.setLayout(new GridLayout(2,1));
    JScrollPane painel = new JScrollPane(lista);
    P2.add(painel); // corresponde ao P21
    P22 = new JPanel();
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    P22.setLayout(new GridLayout(5,1));
    P22.add(L21);
    P22.add(T21);
    P22.add(btQtde);
    P22.add(btIndice);
    P22.add(btLimpar);
    P2.add(P22);
    // Define painel P3 - terceira aba
    P3 = new JPanel();
    P3.setBackground(new Color(190,190,190));
    P3.setLayout(new GridLayout(5,2));
    String [] cores = {"Branco", "Vermelho", "Azul",
                                             "Verde"};
    combo = new JComboBox(cores);
    combo.addItemListener(this);
    P3.add(L31);
    P3.add(combo);
    P3.add(B1);
    P3.add(B4);
    P3.add(B2);
    P3.add(B5);
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    P3.add(B3);
    P3.add(T31);
    P3.add(B6);
    P3.add(T32);
    // Define componentes do painel de botões - abas
    painelBotoes.add(btPainel1);
    painelBotoes.add(btPainel2);
    painelBotoes.add(btPainel3);
    // Define painéis de conteúdo - Cards das abas
    painelCard.add(P1,"p1");
    painelCard.add(P2,"p2");
    painelCard.add(P3,"p3");
    // Define cada painel no painel padrão
    contem.add("North",painelBotoes);
    contem.add("South",painelCard);
  }
  public void actionPerformed(ActionEvent e) {
    // Definições do layout padrão Card
    CardLayout c1=(CardLayout) painelCard.getLayout();
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    if (e.getSource() == btPainel1)
      c1.show(painelCard,"p1");     // mostra painel 1
    if (e.getSource() == btPainel2)
      c1.show(painelCard,"p2");     // mostra painel 2
    if (e.getSource() == btPainel3)
      c1.show(painelCard,"p3");     // mostra painel 3

    // Definições para PAINEL 2
    if (e.getSource() == T21) {
      // adiciona itens a lista
      listModel.addElement(T21.getText());
      T21.setText(" ");
    }
    if (e.getSource() == btQtde)
      T21.setText("Quantidade: " +listModel.getSize());
    if (e.getSource() == btIndice)
      T21.setText("Índice selecionado: " +
                                  lista.getSelectedIndex());
    if (e.getSource() == btLimpar) {
      int index = lista.getSelectedIndex();
      L21.setText("Removido: " +
                                  lista.getSelectedValue());
      listModel.remove(index);
    }
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    // Definições para PAINEL 3
    if (e.getSource() == B1)
      L31.setText("Texto: " + combo.getSelectedItem());
    if (e.getSource() == B2)
      L31.setText("Índice: " +combo.getSelectedIndex());
    if (e.getSource() == B3)
      if (T31.getText().length()!=0) {
        combo.addItem(T31.getText());
        T31.setText("");
      }
    if (e.getSource() == B4)
      combo.removeItemAt(combo.getSelectedIndex());
    if (e.getSource() == B5)
      combo.removeAllItems();
    if (e.getSource() == B6)
      T32.setText(" " + combo.getItemCount());
  }
  public void itemStateChanged(ItemEvent e) {
    // Definições para Painel 3
    if (e.getSource() == T31)
      T31.setText(" " + combo.getSelectedItem());
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    else {
      // Definições para Painel 1
      if (T11.getText().length()==0)
        return;
      try
      {
        valor = Float.parseFloat(T11.getText());
        if (e.getSource() == radio1)
          total = (valor * 10)/100;
        if (e.getSource() == radio2)
          total = (valor * 20)/100;
        if (e.getSource() == radio3)
          total = (valor * 30)/100;
      }
      catch (NumberFormatException erro) {
        T12.setText("Erro");
        return;
      }
      T12.setText(" " + total);
    }
  }
```

Aplicação Gráfica

```
// continuação do exemplo anterior

  // Método para Painel 2
  public void valueChanged(ListSelectionEvent e) {
    L21.setText("Selecionado: " +
                                lista.getSelectedValue());
  }
}
```

Aplicação Gráfica

Opções de Menu

Um outro recurso de interação em aplicações gráficas são os menus, sendo em Java possível a criação de 2 tipos:

a) Menu de barra (na barra da janela)
b) Menu suspenso (pop-up - botão direito)

A criação destes dois tipos de menu exige uma seqüência de passos e objetos a serem usados na aplicação. Primeiramente, será trabalhada a criação de um menu suspenso e depois o pop-up.

Para criar um menu suspenso é necessário construir uma barra de menu:

```
JMenuBar identificador;
```

`identificador` → corresponde ao nome da barra de menu

Aplicação Gráfica

A definição para esta barra ser padrão da janela exige o acionamento do método **setJMenuBar**:

```
setJMenuBar (identificador);
```

`identificador` → corresponde ao nome da barra de menu

Com estas 2 instruções é criada uma barra de menu, porém a mesma ainda esta vazia (sem opções). A criação e definição do **menu** emprega a classe **JMenu**:

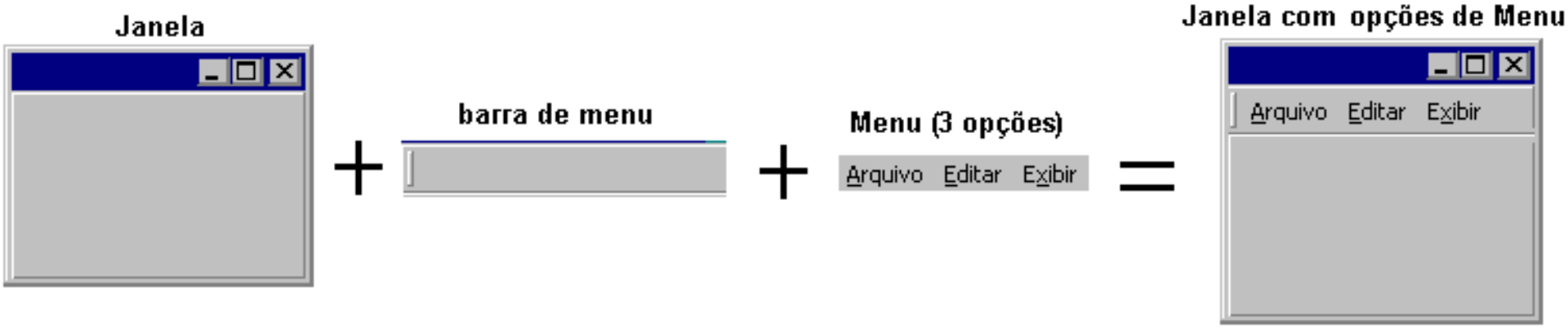
```
JMenu identificador do menu = new JMenu()
```

`identificador do menu` → corresponde ao nome do menu que será inserido na janela gráfica

A inclusão deste menu na barra vazia acontece com o método **add**:

```
identificador da barra.add(identificador do menu);
```

Aplicação Gráfica



Após esta inserção (menu na barra de menu) é necessário incluir nele (menu) os itens que o formarão:

```
JMenuItem identificador do item = new JMenuItem();
```

`identificador do item` → nome da cada item contido no menu

O método **`add`** também é usado para inserir um item de cada vez no menu.

```
identificador do menu.add(identificador do item);
```

Aplicação Gráfica

A elaboração de menu pode envolver outros componentes, tornando sua apresentação mais interativa e agradável por meio do uso de:

- imagens (ícones);
- separadores;
- teclas de atalho (*Mnemonic*);
- subitens ou submenus;
- botões de rádio;
- caixa de seleção.

Aplicação Gráfica

Menu Suspenso (pop-up)

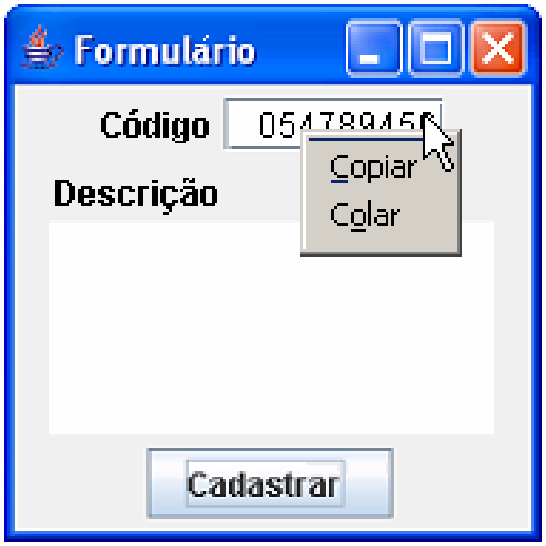
Este tipo de menu é criado como o anterior (menu de barra), porém cada um destes menus estão sempre atrelados a um componente da aplicação gráfica e sem título.

O acionamento deste menu acontece com o clique do botão direito do mouse sobre o componente da aplicação, podendo existirem vários menus diferentes quando uma janela for composta por vários componentes gráficos.

No entanto, existem diferenças entre estas duas opções de menu em Java, onde a elaboração dos suspensos exige o uso de uma classe especial para permitir aos vários objetos da janela o reconhecimento deste menu.

Aplicação Gráfica

Esta classe especial pode ser usada dentro da própria aplicação que está sendo desenvolvida, se constituindo na forma mais simples de implementar os menus pop-up (suspenso).




```
JPopupMenu identificador do menu pop-up;
```

`identificador do menu pop-up` → nome fornecido ao menu

Aplicação Gráfica

Métodos Importantes no Uso de Jmenu e JPopupMenu

- **JMenuBar**: cria barra de menu vazia
- **setJMenuBar**: define a barra de menu padrão
- **JMenu**: cria uma opção de menu
- **JMenuItem**: cria um item de menu
- **JPopupMenu**: cria um menu suspenso em um componente
- **setAccelerator**: define atalho para cada opção de menu
- **setMnemonic**: define uma letra como atalho e a sublinha
- **addSeparator**: inclui um separador no menu
- **ActionListener**: acompanha evento nos 2 tipos de menu, similar aos botões, com inserção do menu (**add**) e implementação do método **actionPerformed**

Aplicação Gráfica

- **ImageIcon**: permite inclusão de imagem na opção do menu, conforme é feito para os botões
- <menu popup>.**show**(componente, posição X, posição Y): mostra o menu suspenso em uma posição na tela em relação ao seu componente de origem
- <menu>.**add**<**item de menu**>: adiciona um item de menu no menu
- <barra>.**add**<**menu**>: adiciona uma opção de menu na barra

Aplicação Gráfica

```
/* Síntese
 *   Objetivo: averiguar as opções de menu
 *   Entrada:  código
 *   Saída:    sem saída (treino em ambiente gráfico)
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Menus extends JFrame implements
 ActionListener {
  JMenuBar barra, suspenso;
  JMenu menuArquivo, menuSalvar;
  JPopupMenu mPopup;
  JMenuItem miNovo, miAbrir, miSair, miSalvar,
            miSalvarC, miSalvarT, miCopiar, miColar;
  JTextField campo;
  JButton btSair;
  Container contem;

  public static void main(String[] args) {
    JFrame janela = new Menus();
    janela.setVisible(true);
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    janela.addWindowListener (new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
         System.exit(0);
      }
    });
  }
  Menus() {    // construtor
    setTitle("Menus");
    setBounds(50,50,150,150);
    contem = getContentPane();
    contem.setBackground(new Color(0,128,128));

    // Montado o menu de barra
    barra = new JMenuBar();
    suspenso = new JMenuBar();
    mPopup = new JPopupMenu();

    menuArquivo = new JMenu("Arquivo");
    menuArquivo.setMnemonic(KeyEvent.VK_A);
    menuArquivo.addActionListener(this);
    menuSalvar = new JMenu("Gravar");
    miNovo = new JMenuItem("Novo",new ImageIcon(
                                          "MenuNovo.gif"));
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    miNovo.addActionListener(this);
    miNovo.setAccelerator(KeyStroke.getKeyStroke(
                  KeyEvent.VK_N,ActionEvent.ALT_MASK));
    miNovo.setMnemonic(KeyEvent.VK_N);
    miAbrir = new JMenuItem("Abrir", new ImageIcon(
                                        "MenuAbrir.gif"));
    miAbrir.addActionListener(this);
    miAbrir.setAccelerator(KeyStroke.getKeyStroke(
                  KeyEvent.VK_A,ActionEvent.ALT_MASK));
    miAbrir.setMnemonic(KeyEvent.VK_B);
    miSalvar = new JMenuItem("Salvar", new ImageIcon(
                                       "MenuSalvar.gif"));
    miSalvar.addActionListener(this);
    miSalvar.setAccelerator(KeyStroke.getKeyStroke(
                 KeyEvent.VK_S,ActionEvent.CTRL_MASK));
    miSalvar.setMnemonic(KeyEvent.VK_S);
    miSalvarC = new JMenuItem("Salvar Como");
    miSalvarC.setAccelerator(KeyStroke.getKeyStroke(
                 KeyEvent.VK_C,ActionEvent.CTRL_MASK));
    miSalvarC.addActionListener(this);
    miSalvarC.setMnemonic(KeyEvent.VK_C);
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    miSalvarT = new JMenuItem("Salvar Tudo");
    miSalvarT.addActionListener(this);
    miSalvarT.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_T,ActionEvent.CTRL_MASK));
    miSalvarT.setMnemonic(KeyEvent.VK_T);
    miSair = new JMenuItem("Sair", new ImageIcon(
                                        "MenuSair.gif"));
    miSair.addActionListener(this);
    miSair.setAccelerator(KeyStroke.getKeyStroke(
                 KeyEvent.VK_X,ActionEvent.ALT_MASK));
    miSair.setMnemonic(KeyEvent.VK_A);
    menuSalvar.add(miSalvar);
    menuSalvar.add(miSalvarC);
    menuSalvar.add(miSalvarT);
    menuArquivo.add(miNovo);
    menuArquivo.add(miAbrir);
    menuArquivo.add(menuSalvar);
    menuArquivo.addSeparator();
    menuArquivo.add(miSair);
    barra.add(menuArquivo);
    setJMenuBar(barra);
```

Aplicação Gráfica

```
// continuação do exemplo anterior

    // Itens do menu suspenso
    miCopiar = new JMenuItem("Copiar");
    miCopiar.addActionListener(this);
    miCopiar.setMnemonic(KeyEvent.VK_P);
    miColar = new JMenuItem("Colar");
    miColar.addActionListener(this);
    miColar.setMnemonic(KeyEvent.VK_O);
    mPopup.add(miCopiar);
    mPopup.add(miColar);

    // Montado conteúdo do painel padrão
    JLabel etiqueta;
    etiqueta = new JLabel("Código");
    etiqueta.setForeground(Color.black);
    campo = new JTextField(30);
    campo.addActionListener(this);
    MouseListener registra = new MostraPopup();
    campo.addMouseListener(registra);

    btSair = new JButton("Encerrar");
    btSair.addActionListener(this);
    btSair.setToolTipText("Pressione botão e saia.");
    btSair.setMnemonic(KeyEvent.VK_X);
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    JPanel painel = new JPanel();
    painel.setLayout(new BorderLayout());
    painel.add("West",etiqueta);
    painel.add("Center", campo);
    painel.add("South",btSair);
    contem.add(painel);
  }
  public void actionPerformed(ActionEvent e) {
    // menu de barra
    if (e.getSource() == miNovo) {
      campo.setText("");
      JOptionPane.showMessageDialog(null,"Menu Novo",
        "Menu Novo",JOptionPane.INFORMATION_MESSAGE);
    }
    if (e.getSource() == miAbrir)
      JOptionPane.showMessageDialog(null,
                          "Menu Abrir", "Menu Abrir",
                    JOptionPane.INFORMATION_MESSAGE);
    if (e.getSource() == miSalvar)
      JOptionPane.showMessageDialog(null,
                  "Submenu Salvar", "Submenu Salvar",
                     JOptionPane.INFORMATION_MESSAGE);
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    if (e.getSource() == miSalvar)
      JOptionPane.showMessageDialog(null,
                  "Submenu Salvar", "Submenu Salvar",
                     JOptionPane.INFORMATION_MESSAGE);
    if (e.getSource() == miSalvarC)
      JOptionPane.showMessageDialog(null,
        "Submenu Salvar Como", "Submenu Salvar Como",
                     JOptionPane.INFORMATION_MESSAGE);
    if (e.getSource() == miSalvarT)
                  JOptionPane.showMessageDialog(null,
       "Submenu Salvar Tudo", "Submenu Salvar Tudo",
                     JOptionPane.INFORMATION_MESSAGE);
    if (e.getSource() == miSair)
       System.exit(0);

    // menu suspenso
    if (e.getSource() == miCopiar)
      JOptionPane.showMessageDialog(null,
            "Menu Suspenso Copiar", "Menu Suspenso",
                     JOptionPane.INFORMATION_MESSAGE);
```

Aplicação Gráfica

```
// continuação do exemplo anterior
    if (e.getSource() == miColar)
      JOptionPane.showMessageDialog(null,
            "Menu Suspenso Colar", "Menu Suspenso",
                  JOptionPane.INFORMATION_MESSAGE);

    // componentes
    if (e.getSource() == campo)
      JOptionPane.showMessageDialog(null,
      "Acesso ao Campo Texto", "Acesso ao Campo Texto",
                        JOptionPane.WARNING_MESSAGE);
    if (e.getSource() == btSair)
       System.exit(0);
  }
  class MostraPopup extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
      mPopup.show(e.getComponent(),e.getX(),
                                          e.getY());
    }
  }
}
```

Exercício de Fixação

2) Observando o Painel 1 (Card 1) do exemplo anterior deverá ser inserido um JCheckBox que permite configurar o campo que mostra o valor da porcentagem desejada para apresentar tal valor ou não. Com esta caixa de seleção selecionada o valor deverá ser mostrado neste campo, caso contrário (não selecionado) as características deste campo deverão ser alteradas para a respectiva porcentagem não mostrar o resultado deste cálculo, É importante destacar que o usuário pode alterar esta caixa de seleção quantas vezes ele quiser e o campo envolvido deverá sempre atender esta situação indicada no JCheckBox.

Aplicação Gráfica

DEMONSTRAÇÃO (SwingSet2.jar)

De acordo com a versão Java instalada em seu computador existirá uma aplicação de exemplo que faz demonstrações de vários componentes gráficos estudados nesta aula, inclusive apresentando o código fonte de cada um destes componentes para sua melhor compreensão.

Exemplo para versão Java do JDK1.6.0:

- Localize o diretório de instalação do Java em seu computador (padrão Windows para esta versão seria):
  - Arquivos de programas >> Java >> jdk 1.6.0_17
- Acesse ainda os diretórios a seguir para execução desta demonstração: demo >> jfc >> SwingSet2
- Execute o arquivo **SwingSet2.jar** (até no próprio prompt do DOS indicando o caminho onde este arquivo está) se o Java estiver corretamente configurado na variável de ambiente do Windows
  - java –jar SwingSet2.jar

Exercícios de Fixação

3) Faça um programa que registre até 100 nomes de eventos científicos (congressos, simpósios, worlshops, etc.), onde cada um deverá ser cadastrado pelo nome, ano e código inteiro que será armazenado como chave (único) no cadastro. Além de implementar todas as propriedades da POO, este sistema será feito somente em ambiente gráfico e coletará todos os dados necessários para tal cadastro. No campo de texto para cadastro do nome do evento deverá existir um menu suspenso só com as opções de Copiar e Colar funcionando, enquanto na aplicação existirá um menu de barra com as opções Principal e Sobre. No Principal existiram os itens Novo, separador e Sair, enquanto que no Sobre só existirá o item Sobre o Sistema. Esta opção mostrará uma mensagem de dialogo (Informação) contendo o nome do sistema, o nome do programador e a data (dd/mm/aaaa) em que o mesmo foi disponibilizado, além de sua versão na última linha e sozinha como: Versão 1.5b. O item Novo só limpará todos os campos para um novo cadastro. A tela possuirá 2 botões (Sair e Salvar), onde o primeiro faz a mesma coisa que a opção de menu Sair e o Salvar grava os dados dos campos no array que armazenará até 100 registros validados. Ressalto porém que este botão só grava um dado no array, sem limpar os campos, o que é feito na opção Novo.

Referência de Criação e Apoio ao Estudo

Material para Consulta e Apoio ao Conteúdo

- HORSTMANN, C. S., CORNELL, G., Core Java2 , volume 1, Makron Books, 2001.
  - Capítulo 7, 8 e 9
- FURGERI, S., Java 2: Ensino Didático: Desenvolvendo e Implementando Aplicações, São Paulo: Érica, 372 p., 2002.
  - Capítulo 9 e 10
- Universidade de Brasília (UnB FGA)
  - https://cae.ucb.br/conteudo/unbfga

    (escolha a disciplina **Orientação a Objetos** no menu superior)