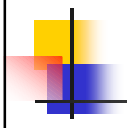
JAVA – INTERFACE GRÁFICA

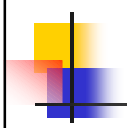
- A Linguagem JAVA e o paradigma orientado a objetos fornecem uma série de recursos que permitem a criação da interface gráfica com usuário (GUI)
- Os conceitos da programação orientada à objetos como: Herança; Polimorfismo e Sobrecarragamento entre outros permitem que a programação seja feita utilizando uma série de classes e métodos que estão disponíveis – Rapidez e qualidade no projeto



JAVA – INTERFACE GRÁFICA

- Podemos classificar as aplicações em três tipos:
  - APLICAÇÕES BASEADAS EM CONSOLE
  - APLICAÇÕES GRÁFICAS BASEADAS EM JANELAS
  - APLICAÇÕES GRÁFICAS BASESADAS NA INTERNET (APPLETs)
- Maiores informações

http://java.sun.com/docs/books/tutorial/uiswing/TOC.html

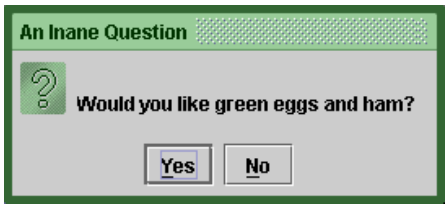


JAVA – INTERFACE GRÁFICA

- APLICAÇÕES BASEADAS EM CONSOLE
  - Não possuem interface gráfica
  - Utilizam o console do sistema
  - Interação é feita através de texto apenas
  - Consiste de uma classe qualquer derivada de Object e devem possuir a função - **public static void main(String args[])**
  - Saída de dados
    - `System.out`
  - Entrada de dados (classes)
    - `System.in`
    - `BufferedReader`
    - `InputStreamReader`

JAVA – INTERFACE GRÁFICA

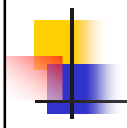
- APLICAÇÕES BASEADAS EM JANELA WINDOW
  - Possuem interface gráfica
  - Classe normalmente derivada de `JFrame` (Window) ou então `JDialog` (Caixa de diálogo)



  - Este tipo de aplicação é executada diretamente sobre a plataforma gráfica (Windows; KDE; etc.)

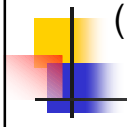
JAVA – INTERFACE GRÁFICA

- APLICAÇÕES BASEADAS EM JANELA WINDOW
  - Consiste de uma janela que possui borda, um título e botões (maximizar; minimizar; fechar; etc.
  - JFrame – Janela
  - JDialog – Janela dependente de outra
  - JInternalFrame – Janela interna a uma outra



INTERFACE GRÁFICA

- A linguagem java possui dois pacotes para a criação de interfaces gráficas:
- AWT (Abstract Windowing ToolKit) – Conjunto de classes para criação de aplicações que usam a interface gráfica.
- SWING – Parte da JFC, toda escrita em java, que implementa uma série de componentes gráficos para interface com o usuário. Os componentes podem ser utilizados em multiplataformas. Esta "biblioteca" implementa os componentes existentes no conjunto AWT (Button; Scrollbar; Label; etc.) e outros como (tree view; list box; tabbed panes; etc.)

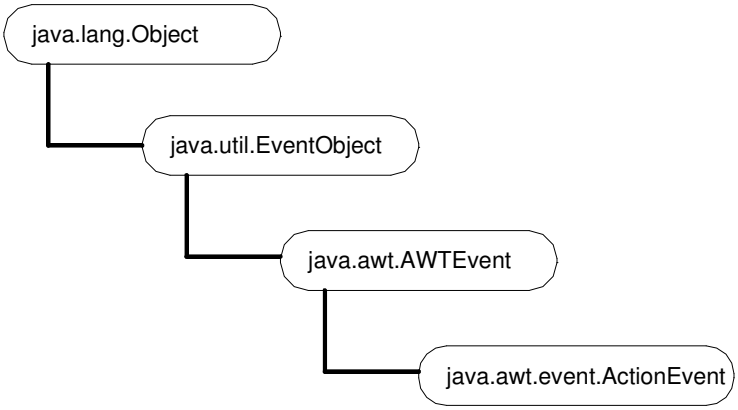


HIERARQUIA DOS COMPONENTES

- Classes definidas pelo pacote Swing (import javax.swing.*;)

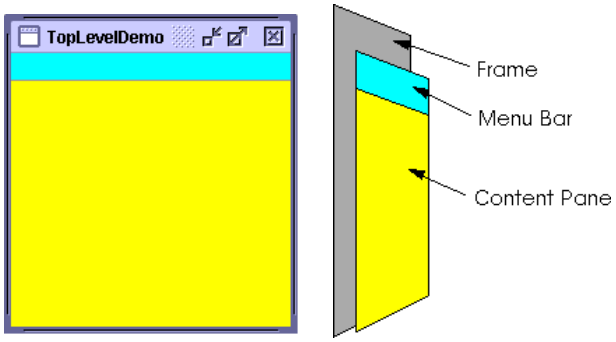
HIERARQUIA DOS EVENTOS

- Classes para manipulação de eventos (import java.awt.event.*;)



INTERFACE GRÁFICA- Window

- ESTRUTURA DE UMA JANELA



- CONTAINER – Objeto que irá conter os controles da interface do usuário. Para ser visto o controle deve estar ligado a algum **container**.

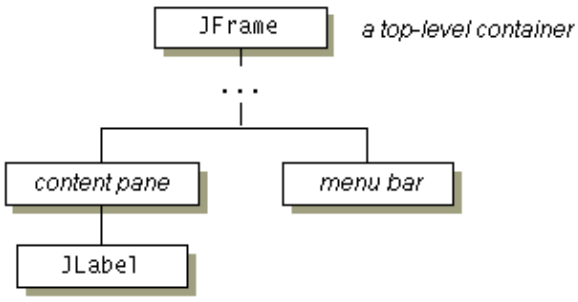
INTERFACE GRÁFICA- Window

- Uma aplicação sempre possui um **container** principal que é a raiz de todos os outros.
- Pode ser adicionado ao **container** uma barra de menus. Esta barra será posicionada no topo do mesmo
- Para recuperar um container de um JFrame, por exemplo, deve utilizar o seguinte método:

```
Container c = getContentPane();
```

INTERFACE GRÁFICA- Window

- ESTRUTURA DE UMA JANELA - continuação



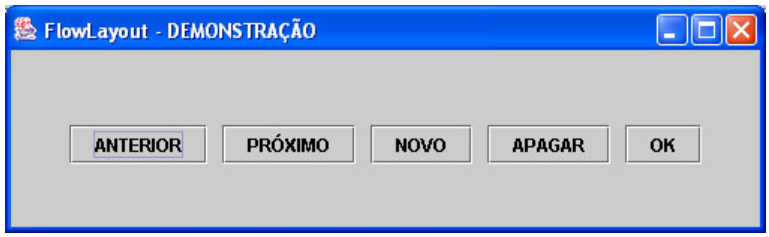
- Existem vários tipos de containers utilizados pela linguagem Java.

INTERFACE GRÁFICA- Window

- Entre os tipos de containers podemos citar:
  - BorderLayout
  - FlowLayout
  - GridLayout
  - BoxLayout
  - CardLayout
  - GridBagLayout
- Container facilita a disposição e o gerenciamento dos objetos que fazem parte da interface gráfica, ao invés de informar a posição específica de cada objeto da interface.

INTERFACE GRÁFICA- LAYOUTS

- **FlowLayout**
  - Componentes dispostos em uma linha, sequencialmente da esquerda para direita na ordem em que foram adicionados.



  - Caso o espaço de uma linha não seja suficiente, múltiplas linhas são utilizadas.

INTERFACE GRÁFICA- LAYOUTS

- **FlowLayout**
  - Os componentes podem ser dispostos da seguinte forma:

CENTRALIZADOS (**`FlowLayout.CENTER`**);

ALINHADOS À ESQUERDA (**`FlowLayout.LEFT`**)

ALINHADOS À DIREITA (**`FlowLayout.RIGHT`**)

```
public FlowLayout(int align, int hgap, int vgap)
align - alinhamento dos componentes
hgap - distância na horizontal entre componentes
vgap - distância na vertical entre componentes
```

INTERFACE GRÁFICA- LAYOUTS

- **FlowLayout – Exemplo**

```
...
//Recupera o container da janela (JFrame)
Container c = getContentPane();
//Ajusta o modo de gerenciamento
c.setLayout(new
FlowLayout(FlowLayout.CENTER,10,50));
//adiciona componentes(botões) ao container
btnAnt = new JButton("ANTERIOR");
c.add(btnAnt);
btnProx = new JButton("PRÓXIMO");
c.add(btnProx);
```

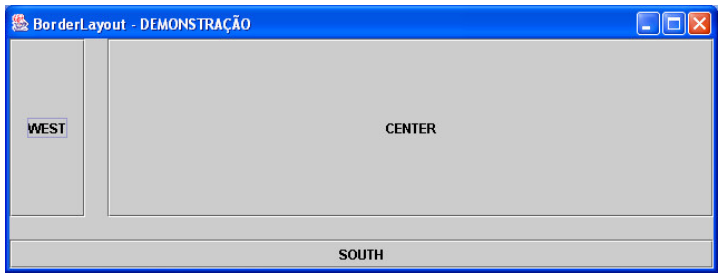

INTERFACE GRÁFICA- LAYOUTS

- **FlowLayout – Exemplo**

```
//continua...
btnNew = new JButton("NOVO");
c.add(btnNew);
btnDelete = new JButton("APAGAR");
c.add(btnDelete);
bntOk = new JButton("OK");
c.add(bntOk);
...
```

INTERFACE GRÁFICA- LAYOUTS

- **BorderLayout**
  - Componentes dispostos cinco regiões: Norte(NORTH); Sul(SOUTH); Leste(EAST); Oeste(WEST) e Centro (CENTER). Cada região pode conter no máximo um componente.



INTERFACE GRÁFICA- LAYOUTS

- **BorderLayout**
  - Um Componente ocupa toda a área de uma região.
  - Componente CENTRAL expande e ocupa áreas não utilizadas (LESTE e/ou OESTE). Se área centro não é utilizada a mesma é deixada vazia.

```
public BorderLayout(int hgap, int vgap)
hgap – distância na horizontal entre componentes
vgap – distância na vertical entre componentes
```

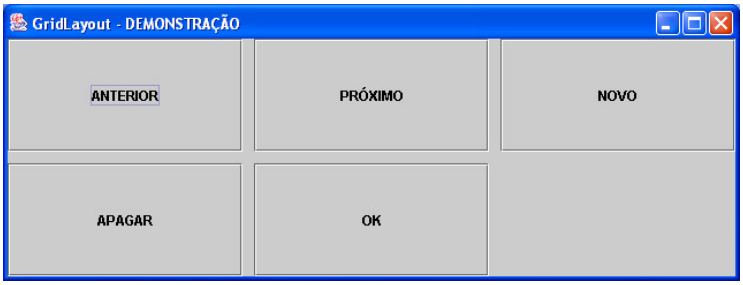
INTERFACE GRÁFICA- LAYOUTS

- **BorderLayout – Exemplo**

```
...
//Recupera o container da janela (JFrame)
Container c = getContentPane();
//Ajusta o modo de gerenciamento
c.setLayout(new BorderLayout(20,20));
btnNew = new JButton("WEST");
c.add(btnNew,BorderLayout.WEST);
btnDelete = new JButton("CENTER");
c.add(btnDelete,BorderLayout.CENTER);
bntOk = new JButton("SOUTH");
c.add(bntOk,BorderLayout.SOUTH);
...
```


INTERFACE GRÁFICA- LAYOUTS

- **GridLayout**
  - A àrea é dividida em retângulos iguais, conforme o número de linhas e colunas especificadas. Um único componente ocupa toda a área deste retângulo



INTERFACE GRÁFICA- LAYOUTS

- **GridLayout**
- Quando o número de linhas é especificado o número de colunas é calculado automaticamente, conforme a quantidade de objetos existentes. Se o número de linhas é igual a zero, a quantidade de colunas é respeitada.

```
public GridLayout(int rows, int cols,
int hgap, int vgap)
```

row – número de linhas
cols – número de colunas
hgap – distância na horizontal entre componentes
vgap – distância na vertical entre componentes

INTERFACE GRÁFICA- LAYOUTS

- **GridLayout – Exemplo**

```
...
//Recupera o container da janela (JFrame)
Container c = getContentPane();
//Ajusta o modo de gerenciamento
//Apesar de ser indiciado 7 colunas
//apenas 3 serão mostradas pois foi
//especificado o número de 2 linhas
c.setLayout(new GridLayout(2,7,10,10));
//continua...
```

INTERFACE GRÁFICA- LAYOUTS

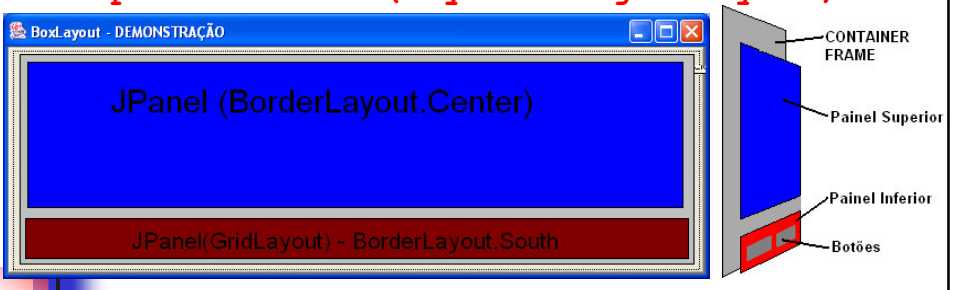
- **GridLayout – Exemplo**

```
btnAnt = new JButton("ANTERIOR");
c.add(btnAnt);
btnProx = new JButton("PRÓXIMO");
c.add(btnProx);
btnNew = new JButton("NOVO");
c.add(btnNew);
btnDelete = new JButton("APAGAR");
c.add(btnDelete);
bntOk = new JButton("OK");
c.add(bntOk);
...
```


INTERFACE GRÁFICA- PAINÉIS

- Normalmente utiliza-se em uma interface mais de um gerenciador de layout.
- O componente **JPanel** é muito útil para organizar os elementos da interface. JPanel é semelhante a um container onde é possível estabelecer como será seu layout interno

```
public JPanel(LayoutManager layout)
```



INTERFACE GRÁFICA- PAINÉIS

```
...
super("Layout - JPanel - DEMONSTRAÇÃO");
Container c = getContentPane();
brdLayout = new BorderLayout();
c.setLayout(brdLayout);
pnlPainel = new JPanel();
c.add(pnlPainel, BorderLayout.CENTER);
pnlPainel.setBackground(Color.blue);
pnlBotes = new JPanel(new
   GridLayout(0,5,20,20));
btnAnt = new JButton("ANTERIOR");
//continua...
```


INTERFACE GRÁFICA- PAINÉIS

```
pnlBotes.add(btnAnt);
btnProx = new JButton("PRÓXIMO");
pnlBotes.add(btnProx);
btnNew = new JButton("NOVO");
pnlBotes.add(btnNew);
bntOk = new JButton("OK");
pnlBotes.add(bntOk);
c.add(pnlBotes, BorderLayout.SOUTH);
pnlBotes.setBackground(Color.red);
...
```

INTERFACE GRÁFICA- EVENTOS

- EVENTO – Ação que ocorre e que pode "percebida" por um objeto
- Quando ocorre um evento o objeto que o recebeu é notificado. Caso o objeto ofereça uma **resposta** ao tipo de evento recebido, o evento então será tratado pelo objeto
- Para se trabalhar com eventos é necessário:
  - Declarar uma classe (handler) que será responsável pelo tratamento.
  - Implementar o tratamento (*handler*) para o evento na classe criada
  - Associar um "ouvinte" (*listener*) para um determinado tipo de evento

INTERFACE GRÁFICA- EVENTOS

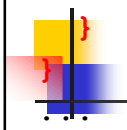
- Tipos de eventos

AÇÃO PRODUTORA DO EVENTO	CLASSE OUVINTE (LISTENER)
Clique de um botão; Digitar <enter> após digitar um texto; escolher um item de um menu	ActionListener
Fechar uma janela; Minimizar; Restaurar o tamanho original; Ativar; Destativar; etc.	WindowListener
Pressionar o botão do mouse; Soltar o botão; Passar o Mouse sobre um componente	MouseListener
Movimentar o mouse; Arrastar (clicar e movimentar)	MouseMotionListener
Tornar um componente visível; Alterar a posição de um componente	ComponentListener
Componente recebe o foco (cursor) do teclado; Componente perde o foco	FocusListener
Elemento selecionado em uma lista é alterado (JList; Jtable)	ListSelectionListener
Propriedade de um componente é alterada	PropertyChangeListener
Utilização de teclado (Pressionar uma tecla; digitar uma tecla; soltar uma tecla	KeyListener
Para maiores informações - veja a classe ouvinte base	EventListener

INTERFACE GRÁFICA- EVENTOS

- EXEMPLO – Clique de um botão

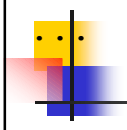
```
//Para tratar um clique de botão então será
//utilizada a interface ActionListener
class ButtonHandler implements ActionListener{
//A interface ActionListener possui um método
//actionPerformed que será disparado sempre
//que o evento ocorrer
public void actionPerformed(ActionEvent e){
   String s =  "Botão NEW pressionado. Por
   enquanto só faço isto!";
   //Mostra uma mensagem na tela
   JOptionPane.showMessageDialog(null,s);
   }
}
```



INTERFACE GRÁFICA- EVENTOS

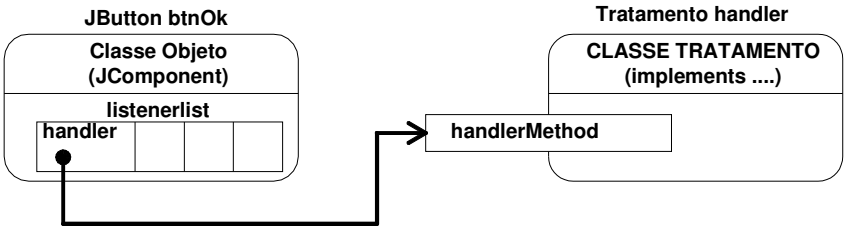
- EXEMPLO – Clique de um botão

```
//Utilização da classe ButtonHandler
private JButton bntOk;
...
//Objeto que será responsável por tratar
//o evento (handler)
ButtonHandler handler = new ButtonHandler();
//Associar um ouvinte de eventos ao
//objeto da interface gráfica
btnNew.addActionListener(handler);
```



INTERFACE GRÁFICA- EVENTOS

- EXEMPLO – Clique de um botão



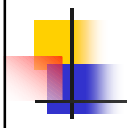
- Na inicialização da aplicação o objeto é criado (Classe Objeto). Além disso é criado o objeto que será responsável por tratar um determinado tipo de evento (Classe Tratamento)
- O Objeto criado para tratamento é associado ao objeto. A lista de ouvintes do objeto irá então possuir um instância do objeto.

INTERFACE GRÁFICA- EVENTOS

- EXEMPLO – Clique de um botão
- Quando o evento for disparado, o componente é notificado do evento que ocorreu. Caso o objeto possua algum tratamento para aquele tipo de evento o método para tratamento será então executado (handlerMethod)
- No exemplo do botão:
  - **Classe tratamento** – class ButtonHandler implents actionListener
  - **Objeto** – JButton btnOk
  - **Objeto Tratamento** – ButtonHandler handler
  - **handlerMethod** – public void actionPerfomed(ActionEvent e)

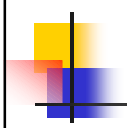
INTERFACE GRÁFICA- EVENTOS

- Para o tratamento de eventos existem dois conceitos importantes normalmente são utilizados
  - **CLASSE INTERNA** – Neste caso a classe para o tratamento dos eventos é criada internamente à definição da classe que contém a interface.
  - Esta classe não pode ser acessada externamente.
  - Neste caso é possível acessar os objetos da interface gráfica dentro da definição da classe que irá tratar os eventos



INTERFACE GRÁFICA- EVENTOS

```
public class JButtonSample2 extends JFrame{
  private JButton btnProx, btnAnt, btnNew,
   btnDelete, btnOk;
  public JButtonSample2(){
    ...
    ButtonHandler handler = new
   ButtonHandler();
    btnNew.addActionListener(handler);
    ...
  }
```



INTERFACE GRÁFICA- EVENTOS

```
//Classe Interna para tratamento de eventos
  class ButtonHandler implements
   ActionListener{
    public void actionPerformed(ActionEvent e)
    {
      ...
      if (obj == btnNew)  //btnNew pode ser
   acessado na classe interna!
        s = "Botão NEW pressionado. Por
        enquanto só faço isto!";
    }
  }
}
```


INTERFACE GRÁFICA- EVENTOS

- **CLASSE ANÔMIMA** – Neste caso a classe para o tratamento dos eventos é criada internamente à definição da classe que contém a interface. Porém esta classe não possui nome. Esta classe não pode ser acessada externamente.
- Esta classe permite que os objetos da interface sejam acessados dentro da mesma

```
public class JButtonSample3 extends JFrame{
  private JButton btnProx, btnAnt, btnNew,
   btnDelete, btnOk;
  ...
 public JButtonSample3(){
  //CLASSE ANÔNIMA para o tatamento de
  //eventos
```

INTERFACE GRÁFICA- EVENTOS

```
ActionListener handler = new ActionListener(){
    public void actionPerformed(ActionEvent e){
      ...
      if (obj == btnNew) //btnNew pode ser
      acessado na classe anônima!
        s = "Botão NEW pressionado. Por
        enquanto só faço isto!";
        ...
      }
    };
    btnNew.addActionListener(handler);
    ...
  }
}
```


INTERFACE GRÁFICA- EVENTOS

- EVENTOS DE MOUSE
  - Utilizam as seguintes interfaces: **`MouseListener`** e **`MouseMotionListener`**
  - MouseListener – Interage com os seguintes eventos:

EVENTO PRODUZIDO	MÉTODO PARA O TRATAMENTO DO EVENTO
Mouse é clicado (pressionado e liberado) sobre um componente	`void mouseClicked(MouseEvent e)`
Mouse entra na área de um componente	`void mouseEntered(MouseEvent e)`
Mouse sai da área de um componente	`void mouseExited(MouseEvent e)`
Mouse é pressionado sobre um componente	`void mousePressed(MouseEvent e)`
Mouse é liberado sobre um componente	`void mouseReleased(MouseEvent e)`

  - Para associar o tratamento de eventos de Mouse a um componente deve ser utilizado o método **`addMouseListener`**

INTERFACE GRÁFICA- EVENTOS

- EVENTOS DE MOUSE
  - Todos os métodos acima devem estar presentes mesmo que não estejam sendo utilizados neste caso o código será apenas - **{ }**
  - As informações que o método pode utilizar, como, por exemplo, a posição do mouse, estão contidas no objeto - **`MouseEvent e`**
  - Se o mouse é pressionado sobre um botão os seguintes eventos são disparados: MOUSE_PRESSED; MOUSE_RELEASED;MOUSE_CLICKED

INTERFACE GRÁFICA- EVENTOS

- EVENTOS DE MOUSE
  - MouseMotionListener – Interage com os seguintes eventos:

EVENTO PRODUZIDO	MÉTODO PARA O TRATAMENTO DO EVENTO
Mouse é clicado sobre um componente e então arrastado	`void mouseDragged(MouseEvent e)`
Mouse foi movimentado sobre um componente mas nenhum botão é clicado	`void mouseMoved(MouseEvent e)`

  - Para associar o tratamento de eventos de movimento do Mouse a um componente deve ser utilizado o método **`addMouseMotionListener`**
  - As informações que o método pode utilizar, como, por exemplo, a posição do mouse, estão contidas no objeto - **`MouseEvent e`**

INTERFACE GRÁFICA- EVENTOS

- EVENTOS DE MOUSE – Exemplo

```
public class JButtonEvents extends JFrame{
  private JLabel lblMouseStatus,
   lblMouseMsg;
  public JButtonEvents(){
    EventHandler handler = new
   EventHandler();
    this.addMouseListener(handler);
    ...
  }
```


INTERFACE GRÁFICA- EVENTOS

```
class EventHandler implements
 MouseListener, ...{
  public void mouseClicked(MouseEvent e){
    String s = "Mouse foi clicado no ponto -
     (" + e.getX() + " , " + e.getY() + ")";
    lblMouseMsg.setText(s);
    System.out.println(s);
  }
  public void mouseEntered(MouseEvent e){}
  public void mouseExited(MouseEvent e){}
  public void mousePressed(MouseEvent e){...}
  public void mouseReleased(MouseEvent e){...}
    ...
  }
}
```

INTERFACE GRÁFICA- EVENTOS

- EVENTOS DE JANELA (WINDOW)
  - WindowListener – Interage com os seguintes eventos:

EVENTO PRODUZIDO	MÉTODO PARA O TRATAMENTO DO EVENTO
Janela está ativa, ou seja, o cursor do teclado está posicionado sobre a mesma	`void windowActivated(WindowEvent e)`
Janela foi completamente fechada. Disparado depois que a janela foi completamente destruida	`void windowClosed(WindowEvent e)`
Janela está prestes a ser fechada. O primeiro evento a ser disparado antes de fechar a janela	`void windowClosing(WindowEvent e)`
Janela está desativada. Cursor do teclado está posicionado sobre outra janela do sistema	`void windowDeactivated(WindowEvent e)`
Janela minimizada,volta ao seu tamanho original	`void windowDeiconified(WindowEvent e)`
Disparado quando a Janela é minimizada	`void windowIconified(WindowEvent e)`
Disparado a primeira vez que uma janela se toma visível	`void windowOpened(WindowEvent e)`

INTERFACE GRÁFICA- EVENTOS

- EVENTOS DE JANELA (WINDOW)
  - Para associar o tratamento de eventos é necessário utilizar o método **`addWindowListener`**
  - Todos os métodos acima devem estar presentes mesmo que não estejam sendo utilizados neste caso o código será apenas - { }
  - As informações que o método pode utilizar, como o estado da janela, por exemplo, estão contidas no objeto - **`WindowEvent e`**

INTERFACE GRÁFICA- EVENTOS

EVENTOS DE JANELA (WINDOW) – Exemplo

```
public class JButtonEvents extends JFrame{
  public JButtonEvents(){
    //Será associado à janela um ouvinte de
   eventos de Janelas
    this.addWindowListener(handler);
    ...
  }
  class EventHandler implements WindowListener,
   ...{
    ...
    public void windowClosed(WindowEvent e){}
    public void windowActivated(WindowEvent e){
      System.out.println("Janela ativada!);}
    //continua...
```


INTERFACE GRÁFICA- EVENTOS

EVENTOS DE JANELA (WINDOW) – Exemplo

```
public void windowDeactivated(WindowEvent e){
  System.out.println("Janela desativada!);}
  public void windowDeiconified(WindowEvent e){
  System.out.println("O Tamanho Original
    restaurado!"; }
  public void windowIconified(WindowEvent e){
    System.out.println("Janela foi
    minimizada!");}
    public void windowOpened(WindowEvent e){
    //JOptionPane.showMessageDialog(null,"A
    janela foi aberta!");}
  }
  ...
}
```

INTERFACE GRÁFICA- EVENTOS

- CLASSES ADAPTADORAS
  - Todos os métodos definidos nas interfaces para tratamento de eventos devem ser codificados, pois interface possui somente a assinatura do método.
  - Método não utilizado necessita pelo menos da instrução **{ }**
  - Uma forma de resolver este problema é a utilização de classes Adaptadoras. São classes são abstratas que possuem todos os métodos declarados um um código do tipo **{ }**

INTERFACE GRÁFICA- EVENTOS

- CLASSES ADAPTADORAS
  - Para utilizar o tratamento de eventos a partir de classes adaptadoras basta derivar a classe adaptadora correspondente e então redefinir apenas os métodos necessários. Esta classe pode tratar somente um tipo de evento. Exemplo:

```
class EventHandler extends MouseAdapter
```

INTERFACE CLASSE PARA TRATAMENTO DE EVENTOS	CLASSE ADAPTADORA
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListenter	WindowAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter

INTERFACE GRÁFICA- JLABEL

- Consiste de um rótulo. Pode conter um texto e/ou uma imagem associada. Não pode ser selecionado. Utilizado apenas para exibir informações. O Label pode possui um dica (tooltip) associado ao mesmo. É possível associar um texto HTML a um label
- **Construtor**

`JLabel(Icon image)` – Cria um label apenas com um ícone

`JLabel(String text)` – Cria um label com um determinado texto

```
JLabel(String text, Icon icon,
int horizontalAlignment)
```

Cria um label, utilizando um texto; um imagem e um alinhamento horizontal (LEFT, CENTER, RIGHT)

INTERFACE GRÁFICA- JLABEL

- **Métodos principais**

`public void setText(String text)` – Altera o texto

`public String getText()` – Recupera o texto

- **Eventos principais**

`PropertyChangeListener; ComponentListener; FocusListener; KeyListener; MouseListener`

JLABEL - Exemplo

```
...
//Cria um ícone
Icon icnFace = new ImageIcon("C:\\FACE.gif");
//Cria um label e ajusta algumas propriedades
lblLabel1 = new JLabel("Proprietario");
lblLabel1.setToolTipText("Nome do Proprietário
   do veículo");
lblLabel2 = new JLabel("Proprietário com Ícone",
   icnFace, JLabel.CENTER);
//
lblLabel3 = new JLabel(icnFace);
lblLabel3.setToolTipText("Label sem o ícone");
lblLabel3.setText("Ajusta o texto");
lblLabel3.setHorizontalAlignment(JLabel.RIGHT);
...
```


INTERFACE GRÁFICA- JTextField

- Consiste de uma linha de texto que pode ser digitada e editada. Quando a tecla <ENTER> é pressionada um `actionEvent` é disparado. Outras classes: `JPasswordField`, utilizado para entrada de senhas e `JFormattedTextField`, que permite controlar o tipo de caracter será digitado (somente números) e o uso de máscaras.
- **Construtores principais**

`JTextField(int columns)` – Cria um campo com um número fixo de colunas

`JTextField(String text)` – Cria e inicializa com um texto

`JTextField(String text, int columns)` – Cria um campo, com inicializado com um texto e com um número fixo de colunas(utilizado para calcular o tamanho)

INTERFACE GRÁFICA- JTextField

- **Métodos principais**

`public void setText(String text)` – Altera o texto

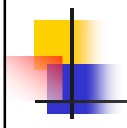
`public String getText()` – Recupera o texto

`public void setFont(Font f)` – Ajusta o tipo de letra (fonte) no campo

`public void addActionListener(ActionListener l)`

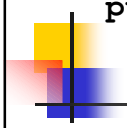
- **Eventos principais**

`PropertyChangeListener; ComponentListener; FocusListener; KeyListener; MouseListener; ActionListener`



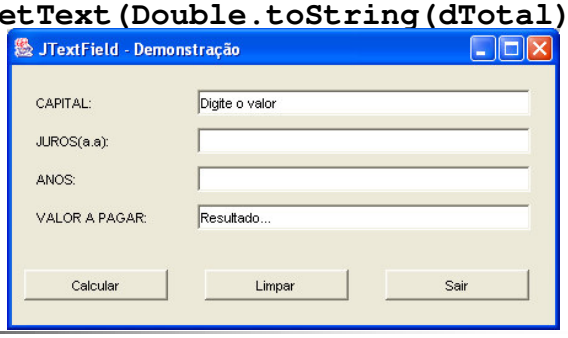
JTextField - Exemplo

```
public class JTextFieldSample extends JFrame{
  public JTextFieldSample(){
    private JTextField txtCapital, txtJuros,
   txtPrazo, txtValor;
    ...
    txtCapital = new JTextField("Digite o
   valor",20);
    txtJuros = new JTextField(5);
    EventHandler handler = new EventHandler();
    txtPrazo.addActionListener(handler);
  }
  class EventHandler implements ActionListener{
    public void actionPerformed(ActionEvent e){
      ...
```



JTextField - Exemplo

```
   ...
   if (obj == txtPrazo) || (obj == btnCalcular)){
    double dTotal;
    String sCap, sJuros;
    dTotal = calculaTotal(sCap,sJuros,
   txtPrazo.getText());
    txtValor.setText(Double.toString(dTotal));}
    }
  }
  ...
}
```



INTERFACE GRÁFICA- JCheckBox

- Consiste de um componente que pode estar selecionado ou não. Caso esteja selecionado, mostra este estado através de uma marca. Em um conjunto destes componentes, mais de um pode estar selecionado. Para se utilizar este componente em um menu, deve ser utilizada a classe **`JToggleButton`**
- **Construtores principais**

`JCheckBox(String text)` – Cria um checkbox não selecionado e com um texto.

`JCheckBox(String text, boolean selected)` – Cria um checkbox que pode estar selecionado ou não

`JCheckBox(Icon icon, boolean selected)` – Cria um checkbox que possui um inicialmente somente um ícone e que pode estar selecionado ou não.

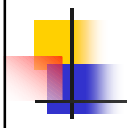
INTERFACE GRÁFICA- JCheckBox

`JCheckBox(String text, Icon icon, boolean selected)` – Cria um checkbox que possui um texto, um ícone e que pode ou não estar selecionado inicialmente

- **Métodos Principais**

`public void setSelected(boolean b)` – Altera o estado selecionado ou não sem no entanto disparar nenhum evento

`public boolean isSelected()` – Verifica se o checkbox está ou não selecionado



INTERFACE GRÁFICA- JCheckBox

- **Eventos Principais**

`ActionListener;itemListener;`

`ComponentListener; FocusListener`

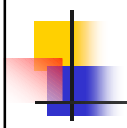
- Neste tipo de componente é normalmente é utilizada a interface **`ItemListener`** para tratamento de eventos:

`public void addItemListener(ItemListener l)`

- Um evento ocorre quando o estado do checkbox é alterado e neste caso o seguinte método deve ser codificado:

`void itemStateChanged(ItemEvent e)`

- O método `int getStateChange()` da classe ItemEvent retorna se o checkBox está ou não selecionado (`ItemEvent.SELECTED`)



INTERFACE GRÁFICA- JRadioButton

- Consiste de um componente que pode estar selecionado ou não. Caso esteja selecionado, mostra este estado através de uma marca. Normalmente em um cojunto destes componentes, somente um pode estar selecionado ao mesmo tempo, sendo assim deve ser utilizado um objeto da classe **`ButtonGroup`** para criar o relacionamento entre estes botões
- **Construtores principais**

`JRadioButton(String text)` – Cria botão com título

`JRadioButton(String text, boolean selected)` – Botão pode estar selecionado ou não

`JRadioButton(Icon icon, boolean selected)` – Cria um botão que possui um inicialmente somente um ícone e que pode estar selecionado ou não

INTERFACE GRÁFICA- JRadioButton

`JRadioButton(String text, Icon icon, boolean selected)` – Cria um botão que possui um texto, um ícone e que pode ou não estar selecionado inicialmente

- **Métodos Principais**

`public void setSelected(boolean b)` – Altera o estado selecionado ou não sem no entanto disparar nenhum evento

`public boolean isSelected()` – Verifica se o botão está ou não selecionado

- **Eventos Principais**

`ActionListener; ComponentListener; FocusListener; ItemListener` (normalmente utilizado)

INTERFACE GRÁFICA - JComboBox

- Consiste de uma lista de itens onde é possível a seleção de um único item. Somente o item selecionado é visível. O ComboBox pode ser editável ou não.
- **Construtores principais**

`JComboBox()` – Cria um combo box, com uma lista vazia.

`JComboBox(Object[] list)` – Cria um ComboBox, onde a lista é inicializada comum array de objetos. Um exemplo seria um vetor do tipo String[].

`JComboBox(Vector list)` – Cria um ComboBox, onde a lista é inicializada com um vetor. Neste caso é utilizada a classe Vector que representa um vetor de tamanho variável e que pode conter qualquer tipo de objeto, sendo pois de grande flexibilidade

INTERFACE GRÁFICA - JComboBox

- **Métodos principais**

`public void setSelectedIndex(int anIndex)` – Ajusta qual elemento da lista está selecionado. O primeiro elemento possui o índice 0 e um índice igual a –1, indica que nenhum elemento está selecionado.

`public void setEditable(boolean aFlag)` – Permite que o Combo Box seja editável, ou seja, é possível acrescentar itens à lista.

`public void addItem(Object anObject)` – Adiciona um novo item à lista

`public Object getSelectedItem()` – Retorna o objeto atualmente selecionado

INTERFACE GRÁFICA - JComboBox

- **Eventos principais**

`ActionListener;ItemListener`

`ActionListener` – Disparado quando é feita um interação com a lista de itens ou então quando <ENTER> é pressionado em um combo editável.

```
public void addActionListener(ActionListener l)
```


INTERFACE GRÁFICA - JComboBox

`ItemListener` – Disparado quando o item selecionado é alterado. Quando um elemento já selecionado é novamente selecionado, nenhum evento é disparado. Quando a seleção sai de um item e vai para outro, dois eventos são disparados.

```
public void addItemListener(ItemListener
aListener)
```

JComboBox – Exemplo

```
public class JComboBoxSample extends JFrame{
  protected JComboBox cmbMetodo;
  protected String[] aMetodos = {"Juros
    Simpes", "Juros Compostos"};
  protected String sMetodoCalculo;
  ...
  //Cria o combo Box e inicializa a lista com
    os elementos do array
  cmbMetodo = new JComboBox(aMetodos);
  //Tratamento eventos
  EventHandler handler = new EventHandler();
  cmbMetodo.addActionListener(handler);
  cmbMetodo.addItemListener(handler);
  class EventHandler implements
    ActionListener, ItemListener{
```


JComboBox – Exemplo

```
public void actionPerformed(ActionEvent e){
      ...
      if ((obj ==
    txtPrazo)||(obj==btnCalcular)||(obj ==
    cmbMetodo))
      setTxtValor(dTotal);
    }
    public void itemStateChanged(ItemEvent e){
      if (e.getSource() == cmbMetodo){
        sMetodoCalculo =
    (String)cmbMetodo.getSelectedItem();
    }
  }
}
```

INTERFACE GRÁFICA - JList

- Consiste de uma lista de itens onde é possível a seleção de um ou mais itens. Caso a lista tenha vários itens barras de rolagens devem ser acrescentadas ao componente.
- **Construtores principais**

`JList()` – Cria um lista de seleção vazia.

`JList(ListModel dataModel)` – Cria uma lista onde os elementos estão contidos em um objeto que implementa a interface `ListModel`. Um exemplo é a classe `DefaultListModel`. Este construtor deve ser utilizado quando se deseja adicionar ou remover itens da lista um vetor de objetos. Um exemplo seria um vetor do tipo String[].

INTERFACE GRÁFICA - JList

`JComboBox(Vector list)` – Cria um ComboBox, onde a lista é inicializada com um vetor. Neste caso é utilizada a classe Vector que representa um vetor de tamanho variável e que pode conter qualquer tipo de objeto, sendo pois de grande flexibilidade

- **Métodos principais**

`public int getSelectedIndex()` - Retorna o índice o primeiro elemento selecionado. Caso nenhum elemento esteja selecionado retorna –1.

`public int[] getSelectedIndices()` - Retorna um vetor com o número dos índices de todos os elementos selecionados

INTERFACE GRÁFICA - JList

`public void setSelectedIndex(int anIndex)`– Ajusta qual elemento da lista está selecionado. O primeiro elemento possui o índice 0 e um índice igual a –1, indica que nenhum elemento da lista está selecionado.

`public Object getSelectedValue()`- Retorna o objeto atualmente selecionado.

`public void setSelectionMode(int selectionMode)`- Ajusta o modo de seleção da lista. Este modo de seleção pode ser: SINGLE_SELECTION (apenas um item por vez; SINGLE_INTERVAL_SELECTION (Um intervalo com vários itens pode ser selecionado) MULTIPLE_INTERVAL_SELECTION (Vários intervalos com vários itens podem ser selecionados).

INTERFACE GRÁFICA - JList

- **Eventos Principais**

`ListSelectionListener` – Ao utilizar esta interface a lista será notificada cada vezv que ouver uma alteração nos itens selecionados na lista

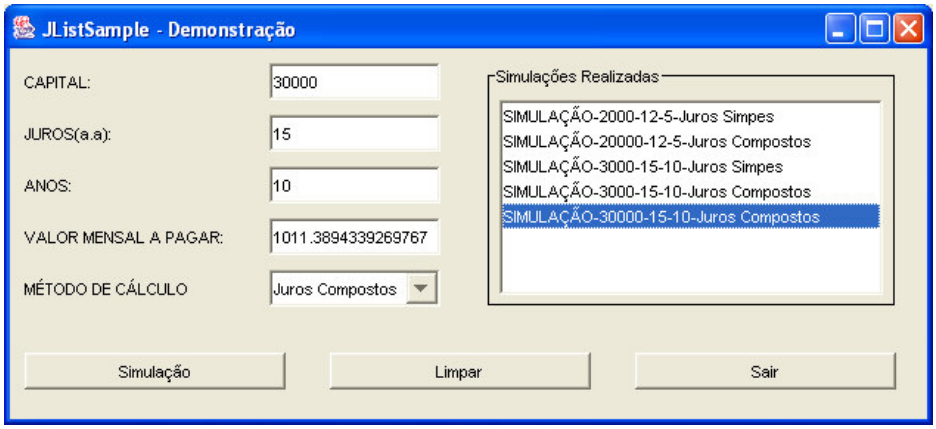
`public void addListSelectionListener( ListSelectionListener listener)`

`MouseListener` – Permite que o componente seja notificado sempre que eventos de Mouse (clique; etc.) ocorrem sobre elementos da lista.

`public void addMouseListener(MouseListener l)`

INTERFACE GRÁFICA - JList

- JList - Exemplo



INTERFACE GRÁFICA - JTextArea

- Consiste de uma área que contém multiplas linhas de texto, onde é possível a edição deste texto utilizando mouse e teclado. No caso de várias linhas barras de rolagens devem ser acrescentadas.
- **Construtores principais**

`JTextArea()` – Cria uma área de texto

`JTextArea(String text, int rows, int columns)` – Cria uma área de texto que é inicializada com a string **Text** e que possui um número especificado de linhas (**rows**) e colunas (**columns**)

- **Métodos principais**

`public void setFont(Font f)` - Ajusta a fonte de texto que será utilizada para mostrar o texto.

INTERFACE GRÁFICA - JTextArea

`public void setEditable(boolean b)` - Permite que o texto seja editável.

`public void setText(String t)` - Ajusta o texto contido no componente.

`public String getSelectedText()` - Retorna o texto dentro do componente que está selecionado.

`public void setLineWrap(boolean wrap)` - Caso o valor de **wrap** seja **true** permite que o texto seja quebrado em várias linhas, quando o tamanho do mesmo for maior que a largura do componente.

- **Eventos**

MouseListener; KeyListener

```
public void addMouseListener(MouseListener l)
public void addKeyListener(KeyListener l)
```


INTERFACE GRÁFICA - JTextArea

- Para adicionar barras de rolagens (JSCrollPane) a um área de texto o seguinte código deve ser utilizado.

```
//Cria texta area com 10 linha e 15 colunas
txaEditor = new JTextArea(10,15);
//Cria as barras de rolagens. A classe
//JSCrollPane fornece uma vista do componente
//juntamente com as barras de rolagens
JScrollPane scrTextArea = new
   JScrollPane(txaEditor);
//A fim de exibir o componente é necessário
//adicionar o mesmo ao container, porém neste
//caso é adicionado a vista do componente com
//as barras de rolagens
getContentPane().add(scrTextArea);
```

JTextArea - Exemplo

```
public class JTextAreaSample extends JFrame{
  protected JTextArea txaObservacoes;
  ...
  public JTextAreaSample(){
    ...
    //Cria a área de texto
    txaObservacoes = new JTextArea(3,80);
    //Adiciona barras de rolagens à área de
   texto
    JScrollPane TxtAreaScrollPane = new
   JScrollPane(txaObservacoes);
    pnlObservacoes.add(TxtAreaScrollPane);
    ...
  }
```


JTextArea - Exemplo

```
  class EventHandler implements ActionListener{
    public void actionPerformed(ActionEvent e){
        ...
      if (obj == btnCalcular){
         String sObs;
         //Recupera o texto da área de texto
         sObs = txaObservacoes.getText();
         ...
      }
    }
  }
}
```

INTERFACE GRÁFICA - Menus

Para se trabalhar com menus, várias classes devem ser utilizadas:

- **`JMenuBar`** – Barra de menus. Normalmente uma janela ou possui apenas um objeto deste tipo. Uma barra de menus possui vários objetos da classe Jmenu. Para adicionar uma barra de menu a uma janela deve ser utilizado o método: `public void setJMenuBar(JMenuBar menubar)`
- **`JMenu`** – Consiste de uma área que é mostrada assim logo que é clicada. Este objeto é o menu propriamente dito. Um menu contém vários objetos da classe **`JMenuItem`**. Caso um objeto seja classe seja adicionado a outro objeto da classe **`JMenu`** cria-se um sub-menu.
- **`JMenuItem`** – Este objeto representa uma opção do menu.

INTERFACE GRÁFICA - Menus

- **Construtores principais**

`JMenuBar()` – Cria uma barra de menu

`JMenu(String s)` – Cria um menu, cujo nome é dado pela string s

`JMenuItem(String text, int mnemonic)` – Cria um item de menu, cujo nome é dado pela string **s** e utiliza uma tecla de atalho indicada pelo inteiro **mnemonic**

- **Métodos principais**

`public JMenu add(JMenu c)` – Adiciona um menu a um objeto JMenuBar

`public JMenuItem add(JMenuItem menuItem)` – Adiciona um item de menu a um objeto da classe JMenu

JAVA – INTERFACE GRÁFICA - Menus

- **Eventos**

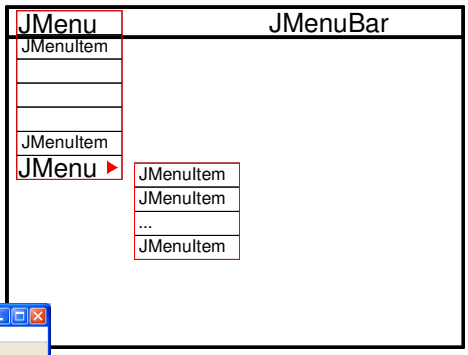
Os menus trabalham com eventos da mesma forma que os Objetos da classe JButton (botões)

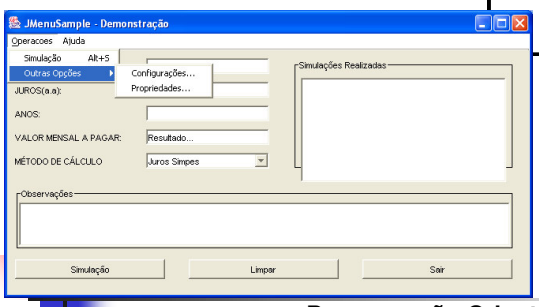
ActionListener – Ações e cliques em itens de menu. A cada item do menu (JMenuItem) deve ser associada uma ação diferente através do método:

```
public void addActionListener(ActionListener l)
```


INTERFACE GRÁFICA - Menus

- Exemplo





Menus - Exemplo

```
public class JMenuSample extends JFrame{
  JMenuBar menuBar;  JMenu menu, submenu;
  JMenuItem menuItem, menuItemSimulacao,
   mnuItmSobre;
 public JMenuSample(){
   menuBar = new JMenuBar();
   setJMenuBar(menuBar);
   menu = new JMenu("Operacoes");
   menu.setMnemonic(KeyEvent.VK_O);
   menuBar.add(menu);
   menuItemSimulacao = new
   JMenuItem("Simulação", KeyEvent.VK_S);
   menu.add(menuItemSimulacao);
   submenu = new JMenu("Outras Opções");
   menuItem = newJMenuItem("Configurações...");
```


Menus - Exemplo

```
  menuItem = new JMenuItem("Propriedades...");
  submenu.add(menuItem);
  menu.add(submenu);
  //Um segundo menu será criado
  menu = new JMenu("Ajuda"); menuBar.add(menu);
  menuItem = new JMenuItem("Ajuda do
    aplicativo"); menu.add(menuItem);
  mnuItmSobre = new JMenuItem("Sobre...");
  menu.add(mnuItmSobre);
  //Eventos
  menuItemSimulacao.addActionListener(handler);
  mnuItmSobre.addActionListener(handler);
      ...
   }
}
```

CRIANDO UMA APPLET

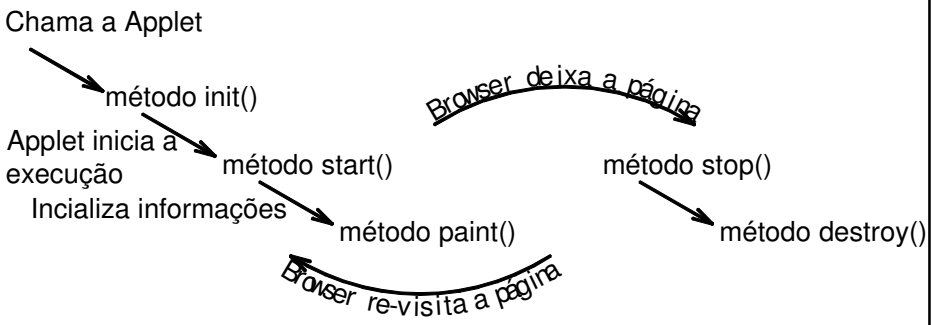
- Uma Applet é um aplicativo gráfico, criado na linguagem java, que pode ser executado em um navegador (Browser).
- Possibilita um aumento das capacidades da WWW uma vez que permite:
  - Animações
  - Imagens com som
  - Efeitos gráficos
  - Programas interativos, como jogos
  - Possibilitam a criação de conexões de rede com o host de origem

CRIANDO UMA APPLET

- **`public void init()`** - É o primeiro método executado e é executado apenas uma vez.
- **`public void start()`** - É executado toda vez que o applet aparece no browser.
- **`public void paint()`** - É executado toda vez que o applet aparece no browser. Recebe umainstância da classe Graphics. (Applet)
- **`public void stop()`** - É executado toda vez que o applet passa a não ser exibido pelo browser.
- **`public void destroy()`** - É executado quando o browser não precisa mais do applet.
- Os métodos acima podem ser especializados conforme a necessidade não sendo obrigatória sua presença.

CRIANDO UMA APPLET

- Sequência de métodos chamados pela Applet



CRIANDO UMA APPLET

- As applets podem ser criadas a partir da classe Applet (**`java.applet.Applet`**) ou então a partir da classe JApplet
- A classe JApplet (**`javax.swing.JApplet`**) é uma especialização da classe Applet e permite a utilização dos componentes SWING para a criação da interface com usuário.
- Todos os componentes e recursos vistos até aqui para a criação de interface de usuário podem ser utilizados para a criação de applets (JApplets), inclusive menus.

A CLASSE JApplet

- **Métodos principais**

`public Container getContentPane()` – Recupera o container principal da Applet

`public URL getCodeBase()` – Recupera a URL onde a applet está sendo executada, este método pode ser utilizado para carregar arquivos, de imagens, por exemplo.
Image image = getImage(getCodeBase(), "imgDir/a.gif");

`public void showStatus(String msg)` – Permite mostrar na barra de status do nagevador uma mensagem de texto.

`public AudioClip getAudioClip(URL url, String name)` – Retorna um objeto AudioClip(arquivo de som) que pode pode ser executado. O arquivo se encontra na URL especificada e possui o nome indicado pela String.

A CLASSE JApplet

`public Container getContentPane()` – Recupera o container principal da Applet

`public URL getCodeBase()` – Recupera a URL onde a applet está sendo executada, este método pode ser utilizado para carregar arquivos, de imagens, por exemplo.
Image image = getImage(getCodeBase(), "imgDir/a.gif");

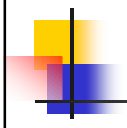
`public void showStatus(String msg)` – Permite mostrar na barra de status do nagevador uma mensagem de texto.

`public AudioClip getAudioClip(URL url, String name)` – Retorna um objeto AudioClip(arquivo de som) que pode pode ser executado. O arquivo se encontra na URL especificada e possui o nome indicado pela String.

A CLASSE JApplet

`public String getParameter(String name)` – Retorna uma string que contém o valor de um parâmetro que foi passado para a applet através do código HTML. O nome do parâmetro deve informado para o método.

`public Image getImage(URL url, String name)` – Retorna um objeto do tipo Image que poderá ser visualizado na tela. O argumento URL equivale ao endereço do arquivo e o argumento String representa seu nome.



EXIBINDO UMA APPLET

- A applet é chamada a partir de um arquivo HTML. Para isto é utilizada a tag <applet>
- Exemplo:

```
<html>
<applet  code=AppletSample.class
         width=640
         height=240>
</applet>
</html>
```

- O utilitário AppletViewer(.exe), fornecido juntamente com J2SDK, permite a visualização de applets independente do navegador

EXIBINDO UMA APPLET

- É possível passar parâmetros para uma applet dentro de um arquivo HTML. Isto pode ser feito da seguinte forma:

```
//Código HTML que chama a applet
<APPLET>
<APPLET CODE= "AudioApplet.class" WIDTH=50 HEIGHT=50>
<PARAM NAME=Arquivo VALUE="AudioCom.au">
...
</APPLET>
//Código para recuperar os parâmetros
this.getParameter("Arquivo") retorna o valor "AudioCom.au“
```


O QUE UMA APPLET NÃO PODE FAZER

- Uma Applet não pode carregar bibliotecas ou definir métodos nativos (usam keyword `native`)
- Uma applet não pode ler ou escrever arquivos no cliente que a está executando
- Não pode iniciar outros programas
- Não tem acesso a certas propriedades do sistema
- As restrições de segurança acima não se aplicam caso a applet seja carregada a partir do sistema de arquivos local em um diretório que esteja presente na variável CLASSPATH

CRIANDO UMA APPLET – Código básico

```
import java.applet.Applet;
import java.awt.Graphics;
public class AppletApp extends Applet{
  public void paint (Graphics g){
    g.drawString("AppletApp - derived from
    Applet class",25,25);
  }
}
//Utilizando a classe JApplet (Swing) como base
import javax.swing.JApplet;
import java.awt.Graphics;
public class JAppletApp extends JApplet{
  public void paint (Graphics g){
    g.drawString("JAppletApp - derived from
    JApplet class",25,25);
  }
}
```


CRIANDO UMA APPLET – Exemplo 2

```
import javax.swing.*;
import java.awt.*;
import java.net.*;
public class AppletVeiculo extends JApplet {
  private int iPasso;
  private final String sPasso= " Passo ";
  private final String sMsg = "Executando
  método ";
  private JPanel pnlBackGround;
  private Container c;
  String sMessage;
  //continua...
```

CRIANDO UMA APPLET – Exemplo 2

```
public void init(){
  iPasso = 1;
  sMessage=sMsg + " INIT: " +sPasso +iPasso;
  this.showStatus(sMessage);
  System.out.println(sMessage);
  iPasso++;
  //Cria o Painel
  pnlBackGround = new JPanel();
  //Recupera o container
  Container c = getContentPane();
}
//continua...
```


CRIANDO UMA APPLET – Exemplo 2

```
public void start(){
  sMessage = sMsg +" START: "+sPasso+iPasso;
  this.showStatus(sMessage );
  System.out.println(sMessage);
  URL urlCodeBase;
  urlCodeBase = this.getCodeBase();
  sMessage = "Codebase: " +
  urlCodeBase.toString();
  this.showStatus(sMessage);
  System.out.println(sMessage);
  iPasso++;
  //Adiciona um painel vermelho ao container
  Container c = getContentPane();
  //continua...
```

CRIANDO UMA APPLET – Exemplo 2

```
    c.setLayout(new BorderLayout());
    //Ajusta a cor do Painel
    pnlBackGround.setBackground(Color.RED);
    //adiciona painel ao container
    c.add(pnlBackGround,BorderLayout.CENTER);
  }
  public void paint(){
    sMessage = sMsg+" PAINT: "+sPasso + iPasso;
    this.showStatus(sMessage);
    System.out.println(sMessage);
    iPasso++;
  }
  //continua...
```


CRIANDO UMA APPLET – Exemplo 2

```
  public void stop(){
    sMessage = sMsg +" STOP: "+ sPasso +iPasso;
    this.showStatus(sMessage);
    System.out.println(sMessage);
    iPasso++;
  }
  public void destroy(){
    sMessage = sMsg+" DESTROY: "+sPasso+iPasso;
    this.showStatus(sMessage );
    System.out.println(sMessage );
    iPasso++;
  }
}
```