Programação Orientada a Objetos

Herança e Interfaces com
Graphic User Interfaces

Sumário

Interfaces e Herança

- Lembre-se em Java não há herança múltipla!
- Se uma classe já herda de outra classe X. Ela pode apenas implementar outras interfaces, não herdar de outras classes.
- Interface: Classe com todos os métodos abstratos e todos os atributos Constantes e de escopo de classe.
- Interfaces Java são muito usadas em sistemas de interface gráfica (janelas).

Exemplos de Interfaces

```
interface  Carro {
   /* public abstract */ void mover();
  /*public static final */ int NUM_RODAS=4;
}
interface Lancha  {
    /* public abstract */ void mover();
   /*public static final */ int NUM_VELAS=0;
}
```

Herança e Interfaces

- Uma classe em Java pode implementar várias interfaces. Sintaxe:

  public class Class1 extends Class2 implements Interface1,Interface2,Interface3 {

  ……}

- Uma Interface pode herdar de várias interfaces.

```
interface  Carro extends Veiculo,Ativo {
   /* public abstract */ void mover();
  /*public static final */ int NUM_RODAS=4;
}
```

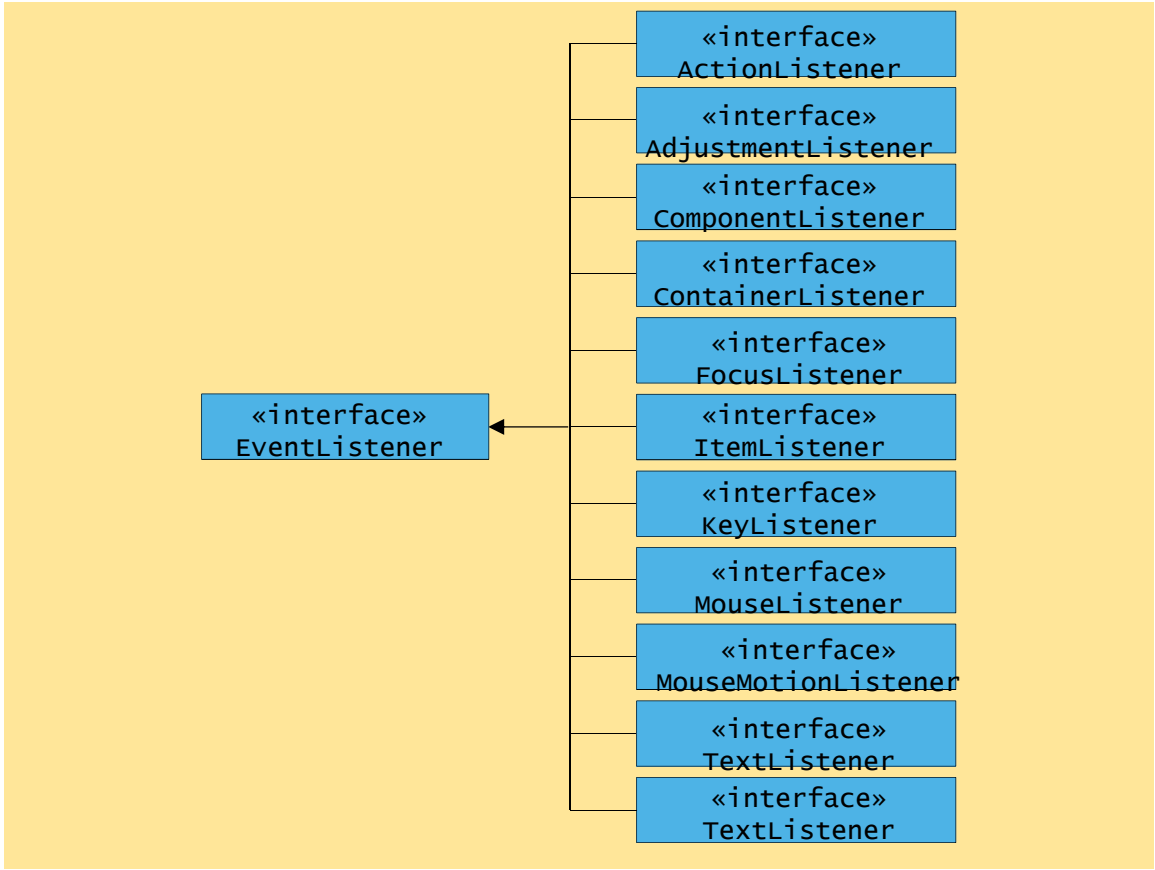
Tratamento de Eventos

- Interfaces Gráficas são dirigidas a eventos
  - Generate *events* when user interacts with GUI
  - e.g., moving mouse, pressing button, typing in text field, etc.
  - Class `java.awt.AWTEvent`

Tratamento de Eventos

- Modelo de tratamento de eventos
  - Três partes
    - Origem do Evento - Event source
      - Componente GUI com o qual o usuário interage
    - Objeto de Evento - Event object
      - Encapsula informação sobre o evento que ocorreu
    - Objeto que deve receber o evento - Event listener
      - Recebe o objeto de evento e responde (faz tratamento)
      - Obedece a uma interface pré-definida com métodos para tratar o evento
  - O programador deve realizar duas tarefas
    - Registrar o “event listener” para cada “event source”
    - Implementar o método de tratamento (event handler)

Event-listener interfaces of package `java.awt.event`



Mouse Event Handling

- Event-listener interfaces for mouse events
  - `MouseListener`
  - `MouseMotionListener`
  - Listen for `MouseEvents`

`MouseListener` and `MouseMotionListener` interface methods

`MouseListener` and **`MouseMotionListener`** interface methods	
Methods of interface `MouseListener`	
`public void mousePressed( MouseEvent event )`	
	Called when a mouse button is pressed while the mouse cursor is on a component.
`public void mouseClicked( MouseEvent event )`	
	Called when a mouse button is pressed and released while the mouse cursor remains stationary on a component.
`public void mouseReleased( MouseEvent event )`	
	Called when a mouse button is released after being pressed. This event is always preceded by a `mousePressed` event.
`public void mouseEntered( MouseEvent event )`	
	Called when the mouse cursor enters the bounds of a component.
`public void mouseExited( MouseEvent event )`	
	Called when the mouse cursor leaves the bounds of a component.
Methods of interface `MouseMotionListener`	
`public void mouseDragged( MouseEvent event )`	
	Called when the mouse button is pressed while the mouse cursor is on a component and the mouse is moved while the mouse button remains pressed. This event is always preceded by a call to `mousePressed`. All drag events are sent to the component on which the drag began.
`public void mouseMoved( MouseEvent event )`	
	Called when the mouse is moved when the mouse cursor on a component. All move events are sent to the component over which the mouse is currently positioned.

Programa de Exemplo para Tratamento de Eventos de Mouse

```
1   // MouseTracker.java
2   // Demonstrating mouse events.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class MouseTracker extends JFrame
8      implements MouseListener, MouseMotionListener {
9
10    private JLabel statusBar;
11
12    // set up GUI and register mouse event handlers
13    public MouseTracker()
14    {
15       super( "Demonstrating Mouse Events" );
16
17       statusBar = new JLabel();
18       getContentPane().add( statusBar, BorderLayout.SOUTH );
19
20       addMouseListener( this );          // listens for
21       addMouseMotionListener( this );    // mouse-motio
22
23       setSize( 275, 100 );
24       setVisible( true );
25    }
26
```

MouseTracker.java

Lines 20-21

Register `JFrame` to receive mouse events

```
27       // MouseListener event handlers
28       // handle event when mouse released immediately after press
29       public void mouseClicked( MouseEvent event )
30       {
31          statusBar.setText( "Clicked at [" + event.getX() +
32             ", " + event.getY() + "]" );
33       }
34
35       // handle event when mouse pressed
36       public void mousePressed( MouseEvent event )
37       {
38          statusBar.setText( "Pressed at [" + event.getX() +
39             ", " + event.getY() + "]" );
40       }
41
42       // handle event when mouse released after dragging
43       public void mouseReleased( MouseEvent event )
44       {
45          statusBar.setText( "Released at [" + event.getX() +
46             ", " + event.getY() + "]" );
47       }
48
49       // handle event when mouse enters area
50       public void mouseEntered( MouseEvent event )
51       {
```

Invoked when user presses and releases mouse button

Invoked when user presses mouse button

Invoked when user releases mouse button after dragging mouse

Invoked when mouse cursor enters `JFrame`

```
52          statusBar.setText( "Mouse entered at [" + event.getX() +
53             ", " + event.getY() + "]" );
54          getContentPane().setBackground( Color.GREEN );
55       }
56
57       // handle event when mouse exits area
58       public void mouseExited( MouseEvent event )
59       {
60          statusBar.setText( "Mouse outside window" );
61          getContentPane().setBackground( Color.WHITE );
62       }
63
64       // MouseMotionListener event handlers
65       // handle event when user drags mouse with button pressed
66       public void mouseDragged( MouseEvent event )
67       {
68          statusBar.setText( "Dragged at [" + event.getX() +
69             ", " + event.getY() + "]" );
70       }
71
72       // handle event when user moves mouse
73       public void mouseMoved( MouseEvent event )
74       {
75          statusBar.setText( "Moved at [" + event.getX() +
76             ", " + event.getY() + "]" );
77       }
78
```

Invoked when mouse cursor exits `JFrame`

Invoked when user drags mouse cursor

`ker.java`

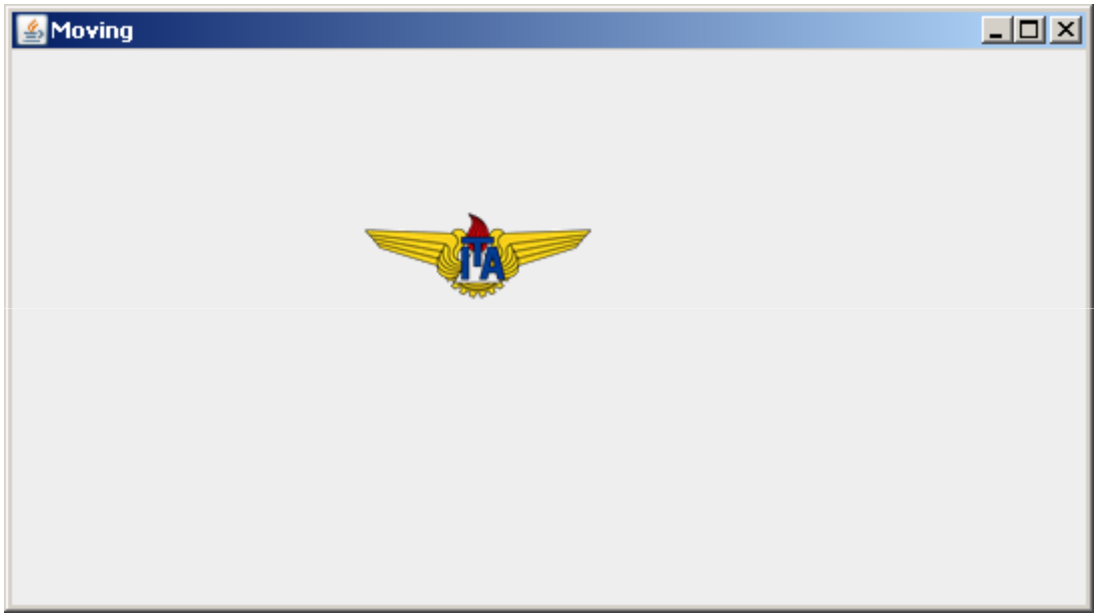
Invoked when user moves mouse cursor

```
79       public static void main( String args[] )
80       {
81          MouseTracker application = new MouseTracker();
82          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
83       }
84
85    } // end class MouseTracker
```

MouseTracker.java

Exercício 1

- Crie uma pequena imagem na janela.
  - Utilize a classe Jlabel para isso e o método setIcon
  - Este método espera um objeto que implemente a interface Icon.
  - Use ImageIcon e a imagem ita.gif
    - ImageIcon icon=new ImageIcon(“ita.gif”);


Moving

Exercício 1.2

- Faça com que a imagem seja arrastada pelo mouse quando estiver “dragging”

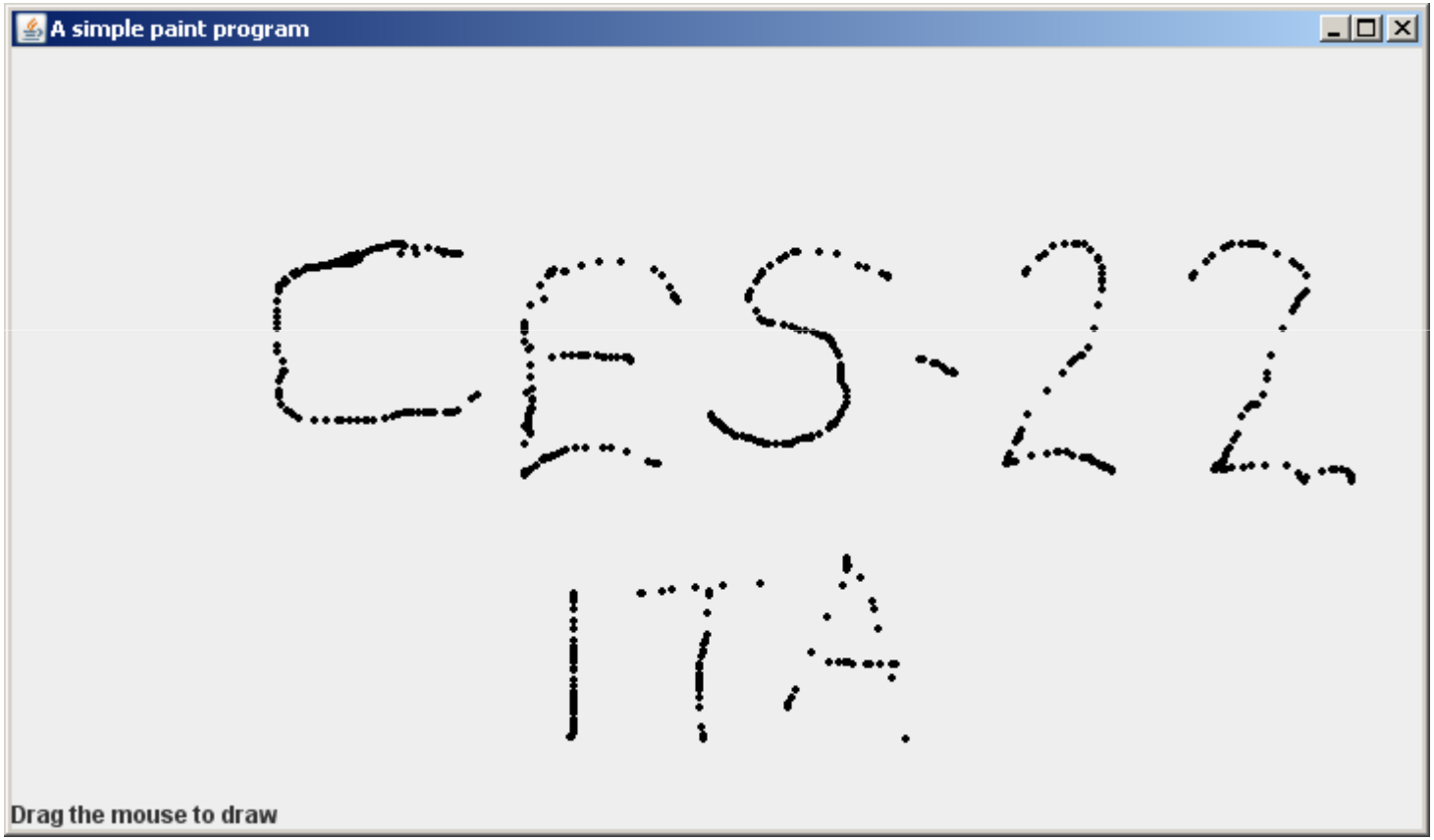
Classes Adaptadoras - Adapter Classes

- Adapter class
  - Implementam uma interface
  - Provêem implementação default para cada método de interface
  - Úteis principalmente quando nem todos os métodos de uma interface são necessários

Event-adapter classes and the interfaces they implement in package `java.awt.event`

Event-adapter class	Implements interface
`ComponentAdapter`	`ComponentListener`
`ContainerAdapter`	`ContainerListener`
`FocusAdapter`	`FocusListener`
`KeyAdapter`	`KeyListener`
`MouseAdapter`	`MouseListener`
`MouseMotionAdapter`	`MouseMotionListener`
`WindowAdapter`	`WindowListener`

Exemplo de Uso de Classes Adaptadoras



```
1   // Painter.java
2   // Using class MouseMotionAdapter.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class Painter extends JFrame {
8      private int pointCount = 0;
9
10    // array of 1000 java.awt.Point references
11    private Point points[] = new Point[ 1000 ];
12
13    // set up GUI and register mouse event handler
14    public Painter()
15    {
16       super( "A simple paint program" );
17
18       // create a label and place it in SOUTH of BorderLayout
19       getContentPane().add( new JLabel( "Drag the mouse to draw" ),
20          BorderLayout.SOUTH );
21
22       addMouseMotionListener(
23
24          new MouseMotionAdapter() {  // anonymous inner class
25
```

Painter.java

Line 22

Register `MouseMotionListener` to listen for window's mouse-motion events

```
26             // store drag coordinates and repaint
27             public void mouseDragged( MouseEvent event )
28             {
29                if ( pointCount < points.length ) {
30                   points[ pointCount ] = event.getPoint();
31                   ++pointCount;
32                   repaint();
33                }
34             }
35
36          } // end anonymous inner class
37
38       ); // end call to addMouseMotionListener
39
40       setSize( 300, 150 );
41       setVisible( true );
42
43    } // end Painter constructor
44
45    // draw oval in a 4-by-4 bounding box at specified location on window
46    public void paint( Graphics g )
47    {
48       super.paint( g ); // clears drawing area
49
50       for ( int i = 0; i < points.length && points[ i ] != null; i++
51          g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
52    }
```

Override method `mouseDragged`, but not method `mouseMoved`

Store coordinates where mouse was dragged, then repaint `JFrame`

`Painter.java`

Draw circle of diameter 4 where user dragged cursor

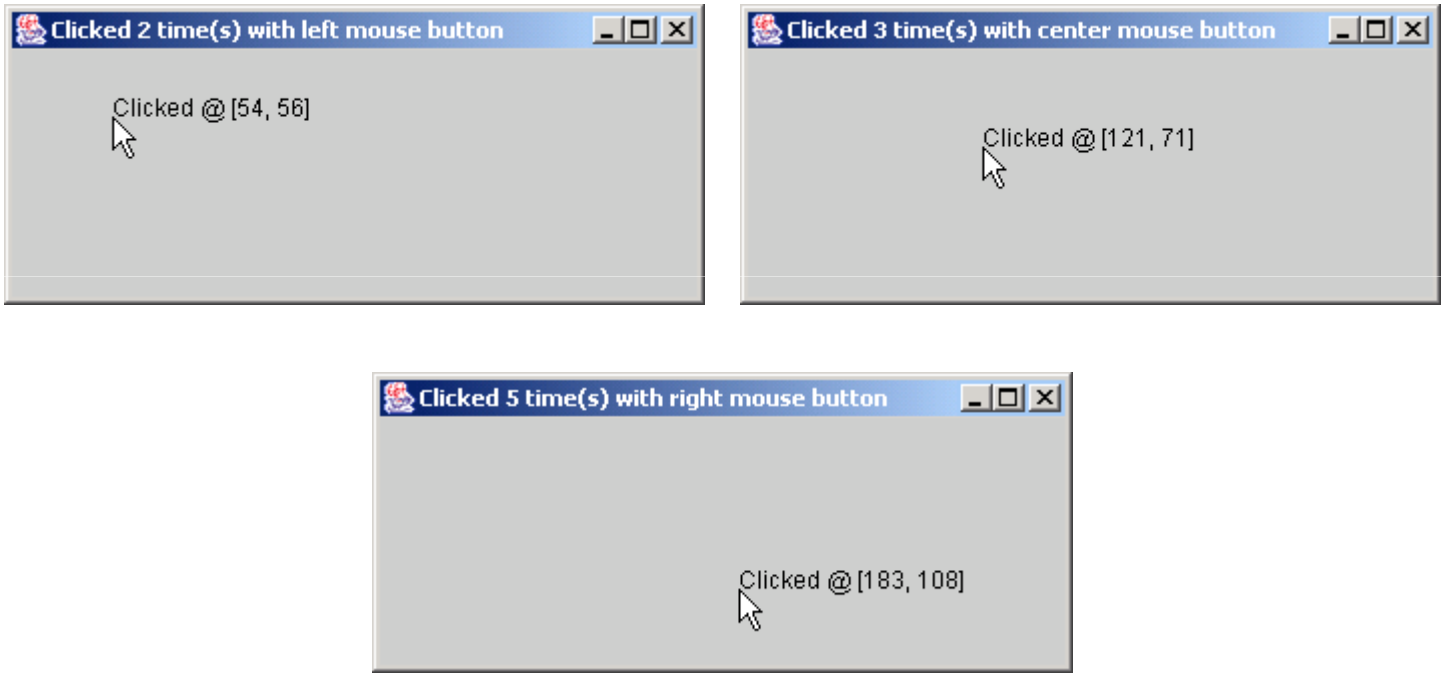
```
53
54   public static void main( String args[] )
55       {
56          Painter application = new Painter();
57          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
58       }
59
60    } // end class Painter
```

Painter.java

Herdando de uma classe adaptadora

–

Exemplo



```
1   // MouseDetails.java
2   // Demonstrating mouse clicks and distinguishing between mouse buttons.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class MouseDetails extends JFrame {
8      private int xPos, yPos;
9
10    // set title bar String; register mouse listener; size and show window
11    public MouseDetails()
12    {
13       super( "Mouse clicks and buttons" );
14
15       addMouseListener( new MouseClickHandler() );
16
17       setSize( 350, 150 );
18       setVisible( true );
19    }
20
21    // draw String at location where mouse was clicked
22    public void paint( Graphics g )
23    {
24       // call superclass paint method
25       super.paint( g );
26
```

Register mouse listener

MouseDetails.java

Line 15

```
27          g.drawString( "Clicked @ [" + xPos + ", " + yPos + "]",
28             xPos, yPos );
29       }
30
31       public static void main( String args[] )
32       {
33          MouseDetails application = new MouseDetails();
34          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
35       }
36
37       // inner class to handle mouse events
38       private class MouseClickHandler extends MouseAdapter {
39
40          // handle mouse click event and determine which button was pressed
41          public void mouseClicked( MouseEvent event )
42          {
43             xPos = event.getX();
44             yPos = event.getY();
45
46             String title = "Clicked " + event.getClickCount() + " time(s)";
47
48             if ( event.isMetaDown() ) // right mouse button
49                title += " with right mouse button";
50
51             else if ( event.isAltDown() ) // middle mouse button
52                title += " with center mouse button";
```

.java

Invoke method `mouseClicked` when user clicks mouse

Store mouse-cursor coordinates where mouse was clicked

Determine number of times user has clicked mouse

Determine if user clicked right mouse button

Determine if user clicked middle mouse button

```
53
54             else  // left mouse button
55                title += " with left mouse button";
56
57             setTitle( title );  // set title bar of window
58             repaint();
59
60          } // end method mouseClicked
61
62       } // end private inner class MouseClickHandler
63
64    } // end class MouseDetails
```

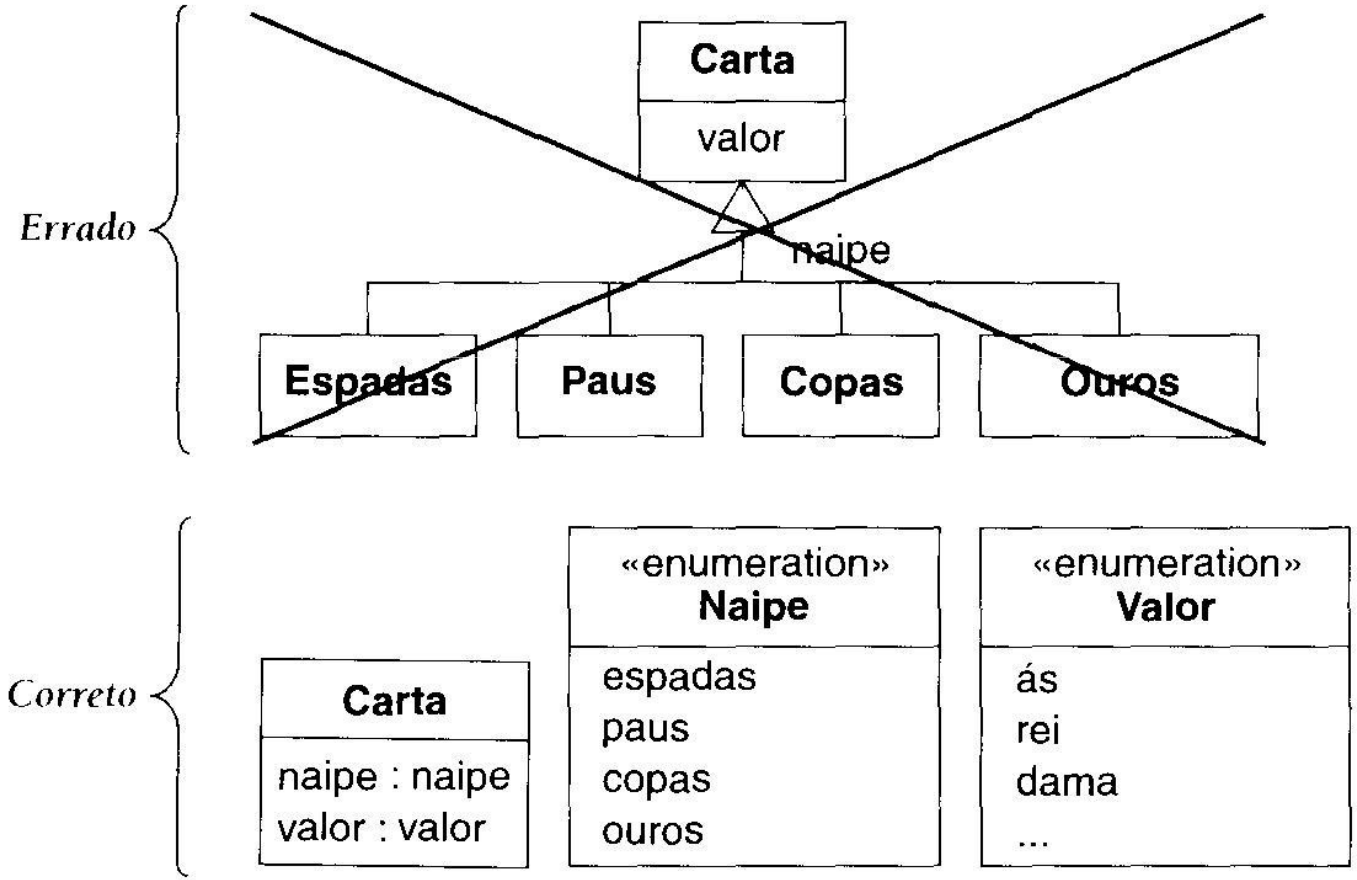
MouseDetails.java

Enumerações

- Além das interfaces, há outro tipo de "classe" especial em OO e Java: Enumerações

- São úteis para representar conjuntos de valores finitos e com "poucos" elementos

Enumerações



Enumerações em Java

```
enum Naipe { ESPADAS, PAUS, COPAS, OURO}
enum Valor { AS, REI, DAMA, VALETE,
   DEZ,NOVE,OITO,SETE,SEIS,CINCO,QUATRO,TRES,DOIS }

class Carta {
public Naipe naipe;    public Valor valor;
public Carta(Naipe naipe,Valor valor) {
this.naipe=naipe;
this.valor=valor;
}
}
```

Enumerações em Java – cont.

```
public class EnumerationTest {
  public static void main(String[] args) {
          for (Naipe n : Naipe.values())   {
                  for( Valor v: Valor.values()) {
                          System.out.printf("%s de %s\n", v, n);
                  }
           }
           Carta carta=new Carta(Naipe.OURO,Valor.AS);
           System.out.printf("%s de %s", carta.valor,carta.naipe);
    }
}
```

Resultado

>AS de ESPADA
REI de ESPADA
DAMA de ESPADA
VALETE de ESPADA
DOIS de ESPADA
TRES de ESPADA
QUATRO de ESPADA
CINCO de ESPADA
SEIS de ESPADA
SETE de ESPADA
OITO de ESPADA
NOVE de ESPADA
DEZ de ESPADA
AS de PAUS

Exercício com Enumerações

- Construa uma classe para representar camisas tamanhos: pequeno, médio, grande ou extra-grande, estilos: manga curta, manga longa, polo e cores: amarelo, azul, cinza, vermelho e preto.
  - Liste todas as possibilidades.

Reflection: Analisando classes em tempo de execução

- Java permite analisar classes em tempo de execução. Isto é, listar seus métodos, campos, construtores e obter informações sobre ele.
- Uso da classe de nome Class. Esta classe traz informações sobre uma classe (ou tipo) qualquer.
  - MinhaClasse.class
  - Class cl=Class.forName(“java.util.Date”);
  - Object obj=cl.newInstance();
  - Class cl2=Double [].class;
  - //Também funciona para tipos primitivos
  - Class cl3=int.class;

Reflection

- Através de Reflection é possível determinar construtores, métodos, campos e superclasses e seus respectivos modificadores (public, private, static, etc)

- Constructor, Method, Field, Modifier são classes auxiliares que guardam informações sobre……
  - O que vc acha?

- Copie o exemplo ReflectionTest.java do site da disciplina para o projeto do cap.2
  - Deve surgir um erro de compilação nessa classe. Identifique e corrija o erro. Mas não altere o código da classe pra corrigir.

```
01 package reflection;
02
03 import java.util.*;
04 import java.lang.reflect.*;
05
06 public class ReflectionTest {
07    public static void main(String[] args)    {
08       // read class name from command line args or user input
09       String name;
10       if (args.length > 0) name = args[0];
11       else       {
12          Scanner in = new Scanner(System.in);
13          System.out.println("Enter class name (e.g. java.util.Date): ");
14          name = in.next();
15       }
16
17       try       {
18          // print class name and superclass name (if != Object)
19          Class cl = Class.forName(name);
20          Class supercl = cl.getSuperclass();
21          String modifiers = Modifier.toString(cl.getModifiers());
22          if (modifiers.length() > 0) System.out.print(modifiers + " ");
23          System.out.print("class " + name);
24          if (supercl != null && supercl != Object.class) System.out.print(" extends "
25                + supercl.getName());
26
```

ReflectionTest.java

```
27            System.out.print("\n{\n");
28            printConstructors(cl);
29            System.out.println();
30            printMethods(cl);
31            System.out.println();
32            printFields(cl);
33            System.out.println("}");
34         }
35         catch (ClassNotFoundException e)
36         {
37            e.printStackTrace();
38         }
39         System.exit(0);
40      }
41
42      /**
43       * Prints all constructors of a class
44       * @param cl a class
45   */
```

ReflectionTest.java

```
46    public static void printConstructors(Class cl)
47    {
48       Constructor[] constructors = cl.getDeclaredConstructors();
49
50       for (Constructor c : constructors)
51       {
52          String name = c.getName();
53          System.out.print("   ");
54          String modifiers = Modifier.toString(c.getModifiers());
55          if (modifiers.length() > 0) System.out.print(modifiers + " ");
56          System.out.print(name + "(");
57
58          // print parameter types
59          Class[] paramTypes = c.getParameterTypes();
60          for (int j = 0; j < paramTypes.length; j++)
61          {
62             if (j > 0) System.out.print(", ");
63             System.out.print(paramTypes[j].getName());
64          }
65          System.out.println(");");
66       }
67    }
68
```

ReflectionTest.java

```
69 -72    /** Prints all methods of a class @param cl a class */
73    public static void printMethods(Class cl)
74    {
75       Method[] methods = cl.getDeclaredMethods();
76
77       for (Method m : methods)
78       {
79          Class retType = m.getReturnType();
80          String name = m.getName();
81
82          System.out.print("   ");
83          // print modifiers, return type and method name
84          String modifiers = Modifier.toString(m.getModifiers());
85          if (modifiers.length() > 0) System.out.print(modifiers + " ");
86          System.out.print(retType.getName() + " " + name + "(");
87
88          // print parameter types
89          Class[] paramTypes = m.getParameterTypes();
90          for (int j = 0; j < paramTypes.length; j++)
91          {
92             if (j > 0) System.out.print(", ");
93             System.out.print(paramTypes[j].getName());
94          }
95          System.out.println(");");
96       }
97    }
98
```

ReflectionTest.java

```
99    /**
100     * Prints all fields of a class
101     * @param cl a class
102     */
103    public static void printFields(Class cl)
104    {
105       Field[] fields = cl.getDeclaredFields();
106
107       for (Field f : fields)
108       {
109          Class type = f.getType();
110          String name = f.getName();
111          System.out.print("   ");
112          String modifiers = Modifier.toString(f.getModifiers());
113          if (modifiers.length() > 0) System.out.print(modifiers + " ");
114          System.out.println(type.getName() + " " + name + ";");
115       }
116    }
117 }
```

ReflectionTest.java

Resultado para GregorianCalendar

```
public class java.util.GregorianCalendar extends java.util.Calendar
{
   public java.util.GregorianCalendar(java.util.TimeZone);
   public java.util.GregorianCalendar(java.util.Locale);
   public java.util.GregorianCalendar(java.util.TimeZone, java.util.Locale);
   public java.util.GregorianCalendar(int, int, int);
   public java.util.GregorianCalendar(int, int, int, int, int);
   public java.util.GregorianCalendar(int, int, int, int, int, int);
   java.util.GregorianCalendar(int, int, int, int, int, int, int);
   public java.util.GregorianCalendar();

   public void add(int, int);
   public boolean equals(java.lang.Object);
   public int hashCode();
   public java.lang.Object clone();
```

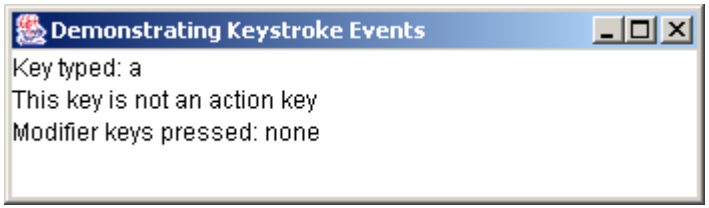
Métodos Úteis de Eventos de Entrada de Mouse – InputEvent methods that help distinguish among left-, center- and right-mouse-button clicks

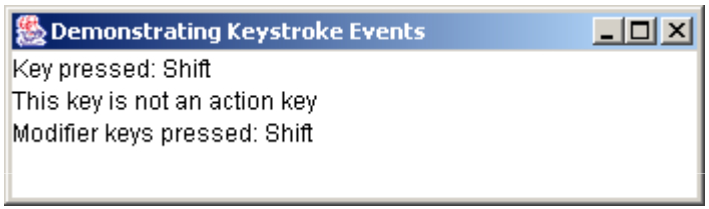
`InputEvent` method	Description
`isMetaDown()`	**Returns `true` when the user clicks the right mouse** button on a mouse with two or three buttons. To simulate a right-mouse-button click on a one-button mouse, the user can hold down the *Meta* key on the keyboard and click the mouse button.
`isAltDown()`	**Returns `true` when the user clicks the middle mouse** button on a mouse with three buttons. To simulate a middle-mouse-button click on a one- or two-button mouse, the user can press the *Alt* key on the keyboard and click the only- or left-mouse button, respectively.

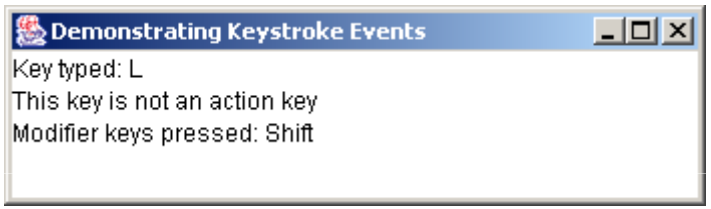
Tratamento de Eventos de Teclado

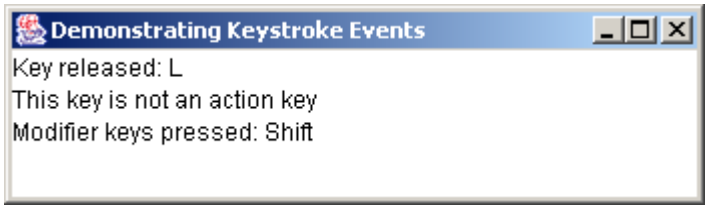
- Interface `KeyListener`
  - Handles *key events*
    - Generated when keys on keyboard are pressed and released
    - `KeyEvent`
      - Contains *virtual key code* that represents key

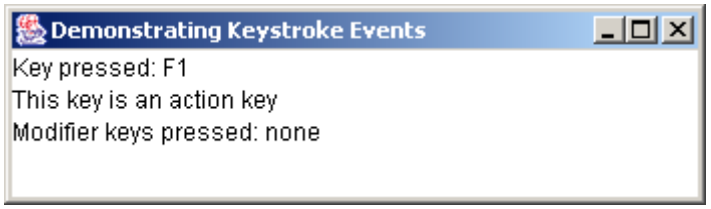
Tratamento de Eventos de Teclado: KeyDemo.java

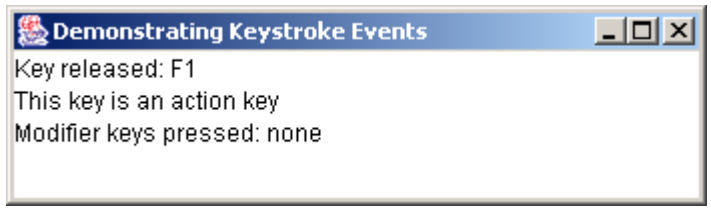












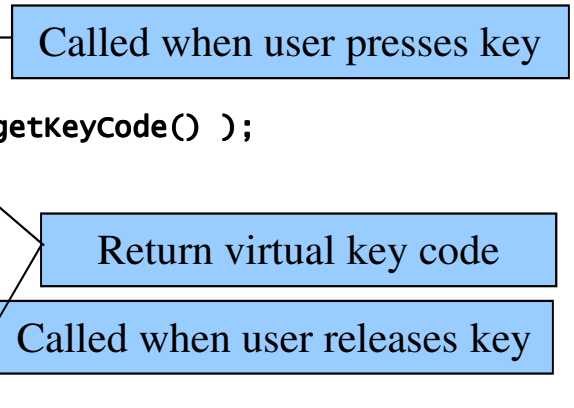
```
1   // KeyDemo.java
2   // Demonstrating keystroke events.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class KeyDemo extends JFrame implements KeyListener {
8      private String line1 = "", line2 = "", line3 = "";
9      private JTextArea textArea;
10
11    // set up GUI
12    public KeyDemo()
13    {
14       super( "Demonstrating Keystroke Events" );
15
16       // set up JTextArea
17       textArea = new JTextArea( 10, 15 );
18       textArea.setText( "Press any key on the keyboard..." );
19       textArea.setEnabled( false );
20       textArea.setDisabledTextColor( Color.BLACK );
21       getContentPane().add( textArea );
22
23       addKeyListener( this );// allow frame to p
24
25       setSize( 350, 100 );
26       setVisible( true );
```

KeyDemo.java

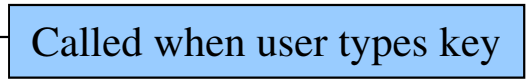
Line 23

Register `JFrame` for key events

```
27
28       } // end KeyDemo constructor
29
30       // handle press of any key
31       public void keyPressed( KeyEvent event )
32       {
33          line1 = "Key pressed: " + event.getKeyText( event.getKeyCode() );
34          setLines2and3( event );
35       }
36
37       // handle release of any key
38       public void keyReleased( KeyEvent event )
39       {
40          line1 = "Key released: " + event.getKeyText( event.getKeyCode() );
41          setLines2and3( event );
42       }
43
44       // handle press of an action key
45       public void keyTyped( KeyEvent event )
46       {
47          line1 = "Key typed: " + event.getKeyChar();
48          setLines2and3( event );
49       }
50
51       // set second and third lines of output
52       private void setLines2and3( KeyEvent event )
53       {
```



KeyDemo.java



```
54          line2 = "This key is " + ( event.isActionKey() ? "" : "not " ) +
55             "an action key";
56
57          String temp = event.getKeyModifiersText( event.getModifiers() );
58
59          line3 = "Modifier keys pressed: " +
60             ( temp.equals( "" ) ? "none" : temp );
61
62          textArea.setText( line1 + "\n" + line2 + "\n" + l
63       }
64
65       public static void main( String args[] )
66       {
67          KeyDemo application = new KeyDemo();
68          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
69       }
70
71    } // end class KeyDemo
```

Determine if *modifier keys* (e.g., *Alt*, *Ctrl*, *Meta* and *Shift*) were used

KeyDemo.java

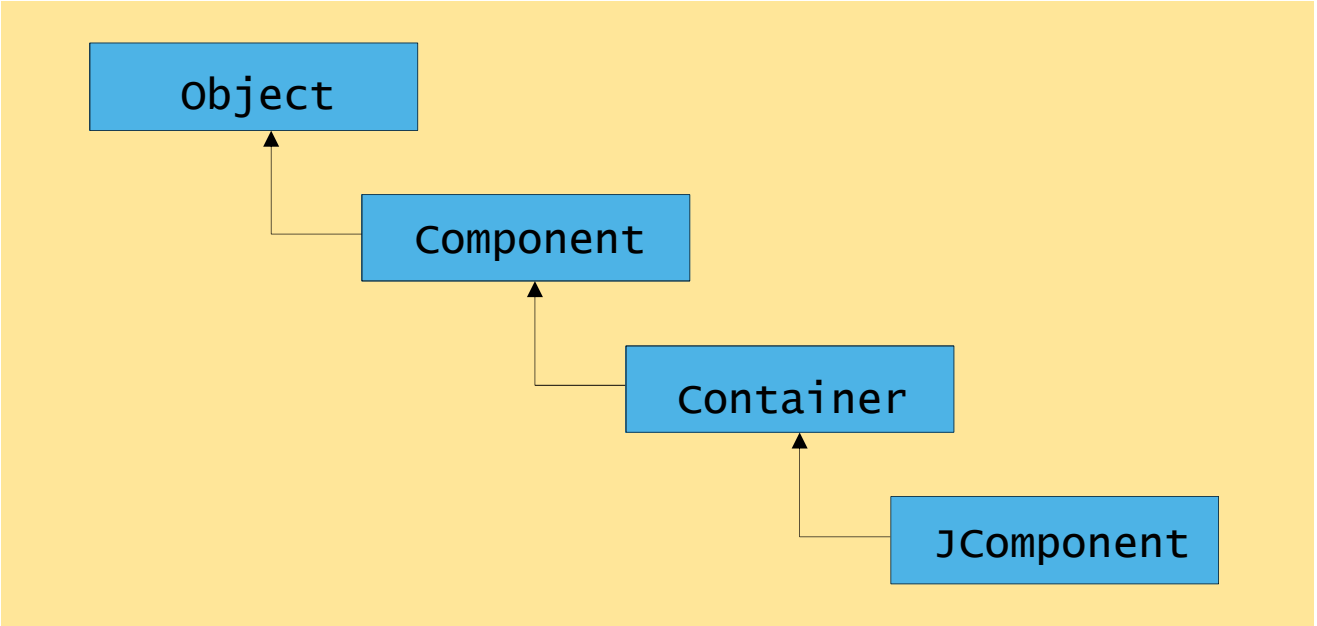
Introdução GUI com Swing

- Graphical User Interface (GUI)
  - Swing permite diferentes “estilos” de programa (“look” and “feel”)
  - Provê componentes que fornecem funcionalides comuns em GUI: butões, menus, listas de seleção, janelas, diálogos, etc.
  - Construído a partir de GUI components (controls, widgets, etc.)
    - Interação via Mouse, teclado, etc.

Visão Geral dos Componentes Swing

- Componentes GUI Swing
  - Package `javax.swing`
  - Componentes originados do AWT (package `java.awt`)
  - Propriedades *look and feel*
    - Appearance and how users interact with program
  - *Lightweight components*
    - Written completely in Java

Hierarquia de Classes de Compontes Swing



Overview of Swing Components

- Class `Component`
  - Contains method `paint` for drawing `Component` onscreen
- Class `Container`
  - Collection of related components
  - Contains method `add` for adding components
- Class `JComponent`
  - *Pluggable look and feel* for customizing look and feel
  - Shortcut keys (*mnemonics*)
  - Common event-handling capabilities

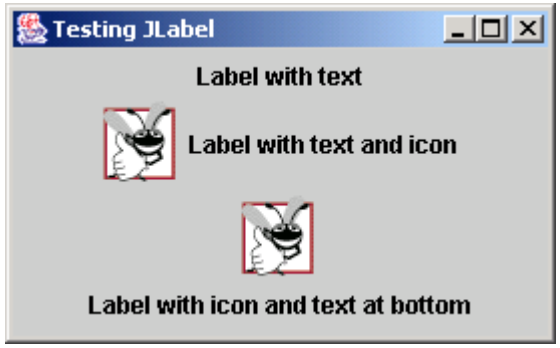
Alguns Componentes Básicos

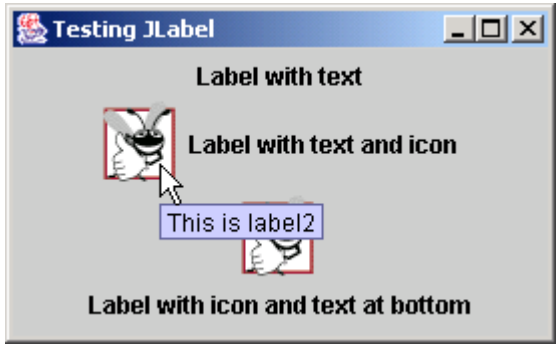
- JFrame
  - Janela onde podem ser colocados outros componentes visíveis
- JLabel
  - Controle para Exibição de Textos não editáveis
- JTextField, JPasswordField
  - Campos de Texto que podem ser editados pelo usuário (texto visivel ou não)
- JButton
  - Componente que dispara um evento ao ser clicado
- JCheckBox and JRadioButton
  - Componentes que podem ser marcados como selecionados ou não selecionados
- JComboBox
  - Componente que permite a escolha de um elemento de uma lista
- Jlist
  - Componente que permite a escolha de um ou mais elementos de uma lista

JLabel

- Label (Rótulos)
  - Provê textos em GUI
  - Definido através da classe `Jlabel`
  - `Derivado de JComponent`
  - Pode apresentar:
    - Linha de texto simples
    - Imagens
    - Textos e imagens

LabelTest.java





```
1   //  LabelTest.java
2   // Demonstrating the JLabel class.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class LabelTest extends JFrame {
8      private JLabel label1, label2, label3;
9
10    // set up GUI
11    public LabelTest()
12    {
13       super( "Testing JLabel" );
14
15       // get content pane and set its layout
16       Container container = getContentPane();
17       container.setLayout( new FlowLayout() );
18
19       // JLabel constructor with a string argument
20       label1 = new JLabel( "Label with text" );
21       label1.setToolTipText( "This is label1" );
22       container.add( label1 );
23
```

LabelTest.java

Line 8

Line 20

Line 21

Declare three `JLabel`s

Create first `JLabel` with text "`Label with text`"

Tool tip is text that appears when user moves cursor over `JLabel`

```
24          // JLabel constructor with string, Icon and alignment arguments
25          Icon bug = new ImageIcon( "bug1.gif" );
26          label2 = new JLabel( "Label with text and icon", bug
27             SwingConstants.LEFT );
28          label2.setToolTipText( "This is label2" );
29          container.add( label2 );
30
31          // JLabel constructor no arguments
32          label3 = new JLabel();
33          label3.setText( "Label with icon and text at bottom" );
34          label3.setIcon( bug );
35          label3.setHorizontalTextPosition( SwingConstants.CENTER );
36          label3.setVerticalTextPosition( SwingConstants.BOTTOM );
37          label3.setToolTipText( "This is label3" );
38          container.add( label3 );
39
40          setSize( 275, 170 );
41          setVisible( true );
42
43       } // end constructor
44
45       public static void main( String args[] )
46       {
47          LabelTest application = new LabelTest();
48          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
49    }
50    }
```

Create second `JLabel` with text to left of image

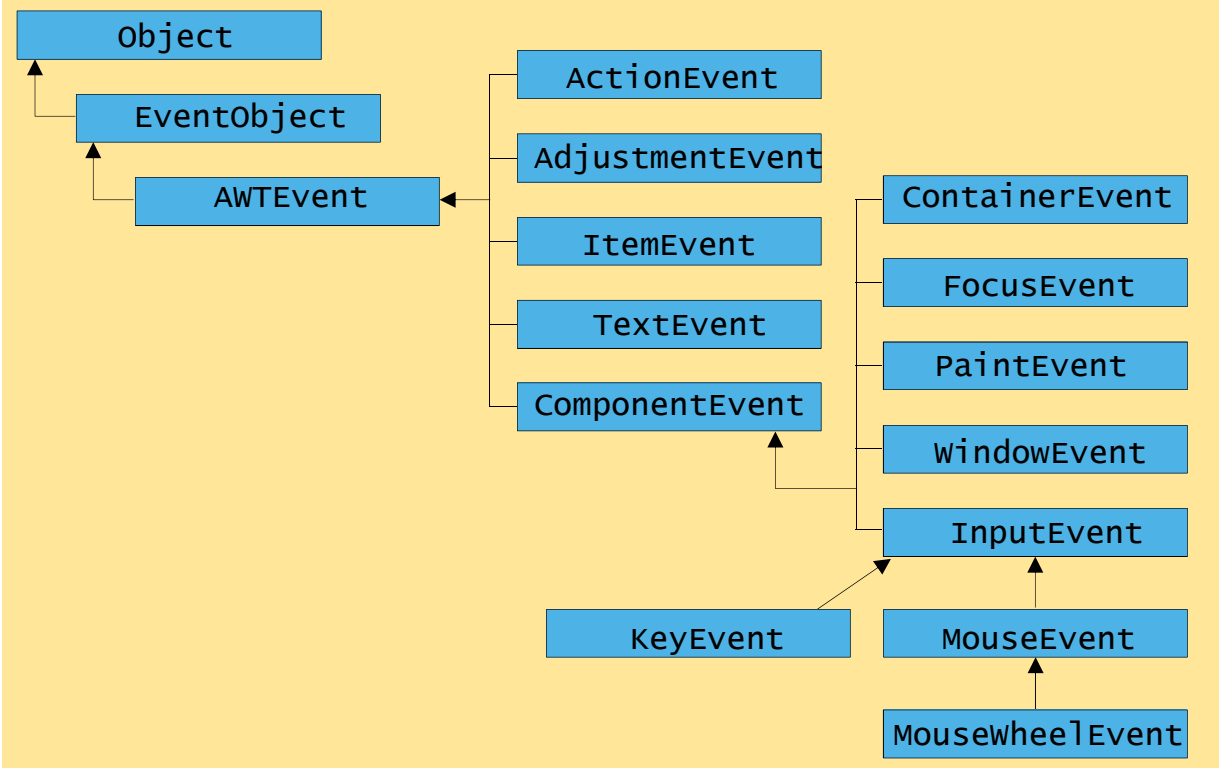
Create third `JLabel` with text below image

LabelTest.java

Lines 16-17

Lines 32-37

Some event classes of package `java.awt.event`



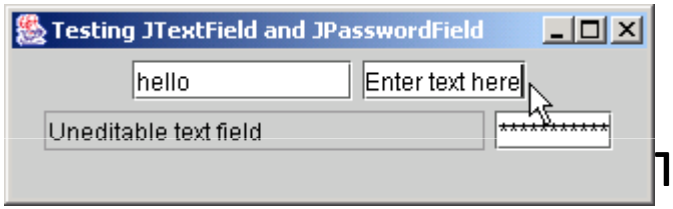
TextFields

- `JTextField`
  - Single-line area in which user can enter text
- `JPasswordField`
  - Extends `JTextField`
  - Hides characters that user enters


Testing JTextField and JPasswordField
Enter text here
Uneditable text field


Testing JTextField and JPasswordField
hello
Enter text here
Uneditable text field


Message
textField1: hello
OK


Testing JTextField and JPasswordField
hello
Enter text here
Uneditable text field

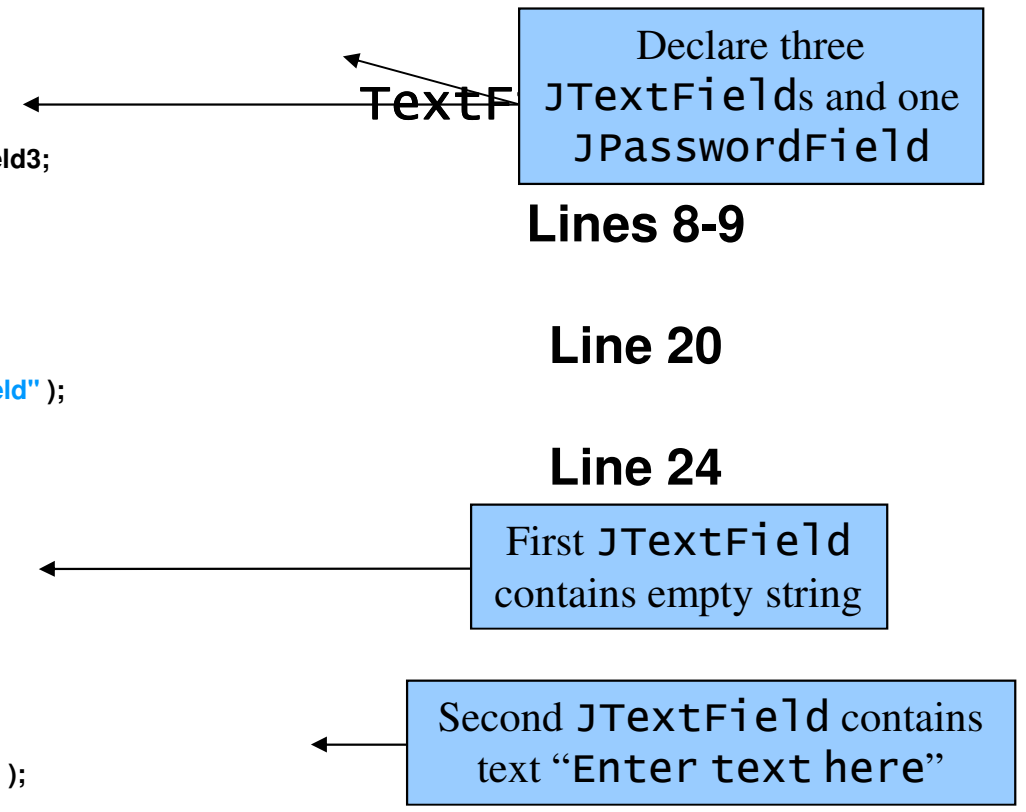
ldTest.java


Message
textField2: Enter text here
OK


Testing JTextField and JPasswordField
hello
Enter text here
Uneditable text field

`TextFieldTest.java`

```
1  // TextFieldTest.java
2  // Demonstrating the JTextField class.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class TextFieldTest extends JFrame {
8     private JTextField textField1, textField2, textField3;
9     private JPasswordField passwordField;
10
11    // set up GUI
12    public TextFieldTest()
13    {
14       super( "Testing JTextField and JPasswordField" );
15
16       Container container = getContentPane();
17       container.setLayout( new FlowLayout() );
18
19       // construct textfield with default sizing
20       textField1 = new JTextField( 10 );
21       container.add( textField1 );
22
23       // construct textfield with default text
24       textField2 = new JTextField( "Enter text here" );
25       container.add( textField2 );
26
```



```
27          // construct textfield with default text,
28          // 20 visible elements and no event handler
29          textField3 = new JTextField( "Uneditable text field", 20
30          textField3.setEditable( false );
31          container.add( textField3 );
32
33          // construct passwordfield with default text
34          passwordField = new JPasswordField( "Hidden text" );
35          container.add( passwordField );
36
37          // register event handlers
38          TextFieldHandler handler = new TextFieldHandler();
39          textField1.addActionListener( handler );
40          textField2.addActionListener( handler );
41          textField3.addActionListener( handler );
42          passwordField.addActionListener( handler );
43
44          setSize( 325, 100 );
45          setVisible( true );
46
47       } // end constructor TextFieldTest
48
49       public static void main( String args[] )
50       {
51          TextFieldTest application = new TextFieldTest();
52          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
53       }
```

Third `JTextField` contains uneditable text

`JPasswordField` contains text "`Hidden text`," but text appears as series of asterisks (*)

Register GUI components with `TextFieldHandler` (register for `ActionEvents`)

TextFieldTest.java

Line 30

Line 34

```
55       // private inner class for event handling
56       private class TextFieldHandler implements ActionListener {
57
58          // process textfield events
59          public void actionPerformed( ActionEvent event )
60          {
61             String string = "";
62
63             // user pressed Enter in JTextField textField1
64             if ( event.getSource() == textField1 )
65                string = "textField1: " + event.getActionCommand();
66
67             // user pressed Enter in JTextField textField2
68             else if ( event.getSource() == textField2 )
69                string = "textField2: " + event.getActionCommand();
70
71             // user pressed Enter in JTextField textField3
72             else if ( event.getSource() == textField3 )
73                string = "textField3: " + event.getActionCommand();
74
75             // user pressed Enter in JTextField passwordField
76             else if ( event.getSource() == passwordField ) {
77                string = "passwordField: " +
78                   new String( passwordField.getPassword() );
79    }
81             JOptionPane.showMessageDialog( null, string );
82
83          } // end method actionPerformed
84
85       } // end private inner class TextFieldHandler
86
87    } // end class TextFieldTest
```

Every `TextFieldHandler` instance is an `ActionListener`

Method `actionPerformed` invoked when user presses Enter in GUI field

TextFieldTest.java

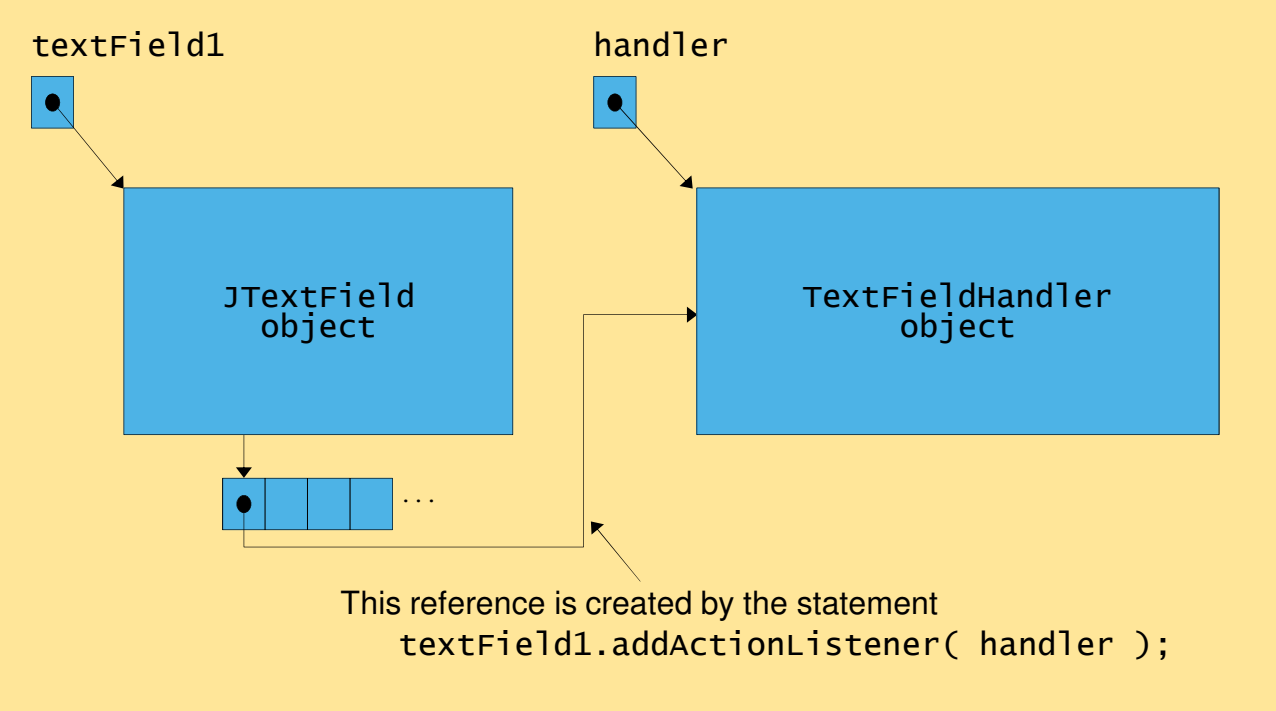
Line 56

Line 59

Como o tratamento de eventos funciona em Componentes?

- Como o manipulador de eventos é registrado ?
  - Resposta:
    - Através do método de componente `addActionListener`
    - Linhas 39-42 de `TextFieldTest.java`
- Como o componente sabe chamar o respectivo `actionPerformed`?
  - Resposta:
    - Evento é enviado apenas para os listeners do tipo apropriado
    - Cada tipo de evento tem uma interface event-listener correspondente
      - » Um ID do evento ocorrido especifica seu tipo

Registro de Eventos para `JTextField textField1`



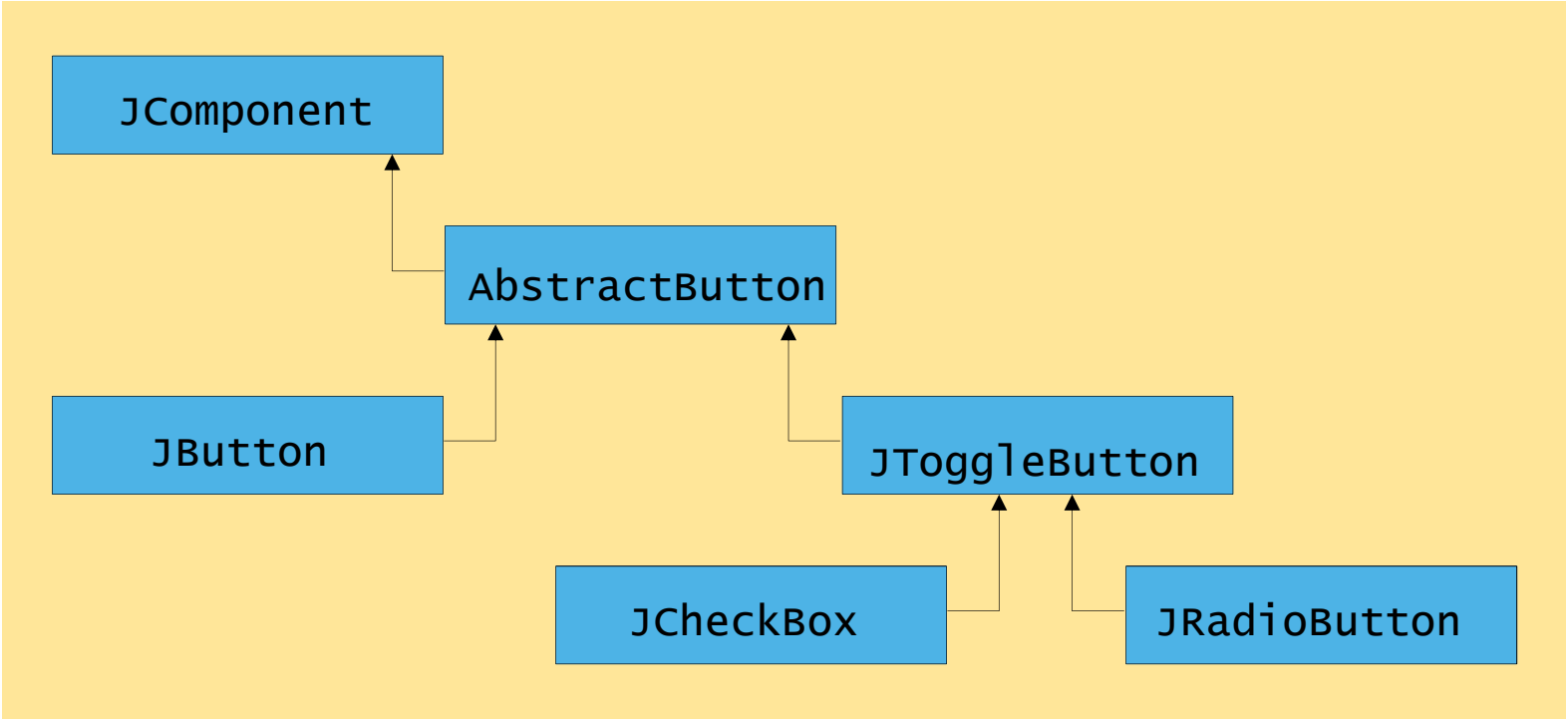
Exercício

- Altere o programa TextFieldTest para que seja alterado a barra de título da janela, para o valor do campo alterado, qualquer que seja tal campo
  - Dica: O ‘handler’ precisará de uma referência para a janela.

JButton

- Button
  - Component user clicks to trigger a specific action
  - Several different types
    - Command buttons
    - Check boxes
    - Toggle buttons
    - Radio buttons
  - `javax.swing.AbstractButton` subclasses
    - Command buttons are created with class `JButton`
      - Generate `ActionEvents` when user clicks button

Swing button hierarchy



`ButtonTest.java`

```
1   // ButtonTest.java
2   // Creating JButtons.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class ButtonTest extends JFrame {
8      private JButton plainButton, fancyButton;
9
10    // set up GUI
11    public ButtonTest()
12    {
13       super( "Testing Buttons" );
14
15       // get content pane and set its layout
16       Container container = getContentPane();
17       container.setLayout( new FlowLayout() );
18
19       // create buttons
20       plainButton = new JButton( "Plain Button" );
21       container.add( plainButton );
22
23       Icon bug1 = new ImageIcon( "bug1.gif" );
24       Icon bug2 = new ImageIcon( "bug2.gif" );
25       fancyButton = new JButton( "Fancy Button", bug1 );
26       fancyButton.setRolloverIcon( bug2 );
27       container.add( fancyButton );
```

Create two references to `JButton` instances

Instantiate `JButton` with text

Instantiate `JButton` with image and *rollover* image

ButtonTest.java

Line 8

Line 20

Lines 24-26

```
28
29          // create an instance of inner class ButtonHandler
30          // to use for button event handling
31          ButtonHandler handler = new ButtonHandler();
32          fancyButton.addActionListener( handler );
33          plainButton.addActionListener( handler );
34
35          setSize( 275, 100 );
36          setVisible( true );
37
38       } // end ButtonTest constructor
39
40       public static void main( String args[] )
41       {
42          ButtonTest application = new ButtonTest();
43          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
44       }
45
46       // inner class for button event handling
47       private class ButtonHandler implements ActionListener {
48
49          // handle button event
50          public void actionPerformed( ActionEvent event )
51          {
52             JOptionPane.showMessageDialog( ButtonTest.this,
53                "You pressed: " + event.getActionCommand() );
54          }
55       }
56    }
```

Instantiate `ButtonHandler` for `JButton` event handling

Register `JButton`s to receive events from `ButtonHandler`

When user clicks `JButton`, `ButtonHandler` invokes method `actionPerformed` of all registered listeners

ButtonTest.java

Line 31

Lines 32-33

Line 50

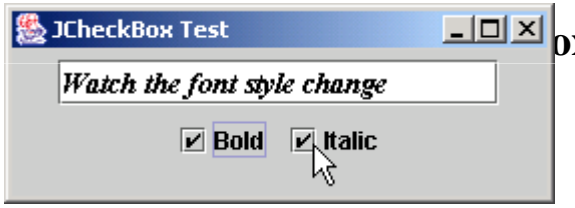
JCheckBox and JRadioButton

- State buttons
  - On/Off or `true`/`false` values
  - Java provides three types
    - `JToggleButton`
    - `JCheckBox`
    - `JRadioButton`


JCheckBox Test
Watch the font style change
Bold
Italic


JCheckBox Test
Watch the font style change
Bold
Italic


JCheckBox Test
Watch the font style change
Bold
Italic


JCheckBox Test
Watch the font style change
Bold
Italic

oxTest.java

```
1   // CheckBoxTest.java
2   // Creating JCheckBox buttons.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class CheckBoxTest extends JFrame {
8      private JTextField field;
9      private JCheckBox bold, italic;
10
11    // set up GUI
12    public CheckBoxTest()
13    {
14       super( "JCheckBox Test" );
15
16       // get content pane and set its layout
17       Container container = getContentPane();
18       container.setLayout( new FlowLayout() );
19
20       // set up JTextField and set its font
21       field = new JTextField( "Watch the font style change", 20 );
22       field.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
23       container.add( field );
24
```

CheckBoxTest.java

Line 9

Declare two `JCheckBox` instances

Line 22

Set `JTextField` font to Serif, 14-point plain

```
25          // create checkbox objects
26          bold = new JCheckBox( "Bold" );
27          container.add( bold );
28
29          italic = new JCheckBox( "Italic" );
30          container.add( italic );
31
32          // register listeners for JCheckBoxes
33          CheckBoxHandler handler = new CheckBoxHandler();
34          bold.addItemListener( handler );
35          italic.addItemListener( handler );
36
37          setSize( 275, 100 );
38          setVisible( true );
39
40       } // end CheckBoxText constructor
41
42       public static void main( String args[] )
43       {
44          CheckBoxTest application = new CheckBoxTest();
45          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
46       }
47
```

Instantiate `JCheckBox`s for bolding and italicizing `JTextField` text, respectively

Register `JCheckBox`s to receive events from `CheckBoxHandler`

CheckBoxTest.java

Lines 26 and 29

Lines 34-35

```
48       // private inner class for ItemListener event handling
49       private class CheckBoxHandler implements ItemListener {
50          private int valBold = Font.PLAIN;
51          private int valItalic = Font.PLAIN;
52
53          // respond to checkbox events
54          public void itemStateChanged( ItemEvent event )
55          {
56             // process bold checkbox events
57             if ( event.getSource() == bold )
58                valBold = bold.isSelected() ? Font.BOLD : Font.PLAIN;
59
60             // process italic checkbox events
61             if ( event.getSource() == italic )
62                valItalic = italic.isSelected() ? Font.I
63
64             // set text field font
65             field.setFont( new Font( "Serif", valBold + valItalic, 14 ) );
66
67          } // end method itemStateChanged
68
69       } // end private inner class CheckBoxHandler
70
71    } // end class CheckBoxTest
```

When user selects `JCheckBox`, `CheckBoxHandler` invokes method `itemStateChanges` of all registered listeners

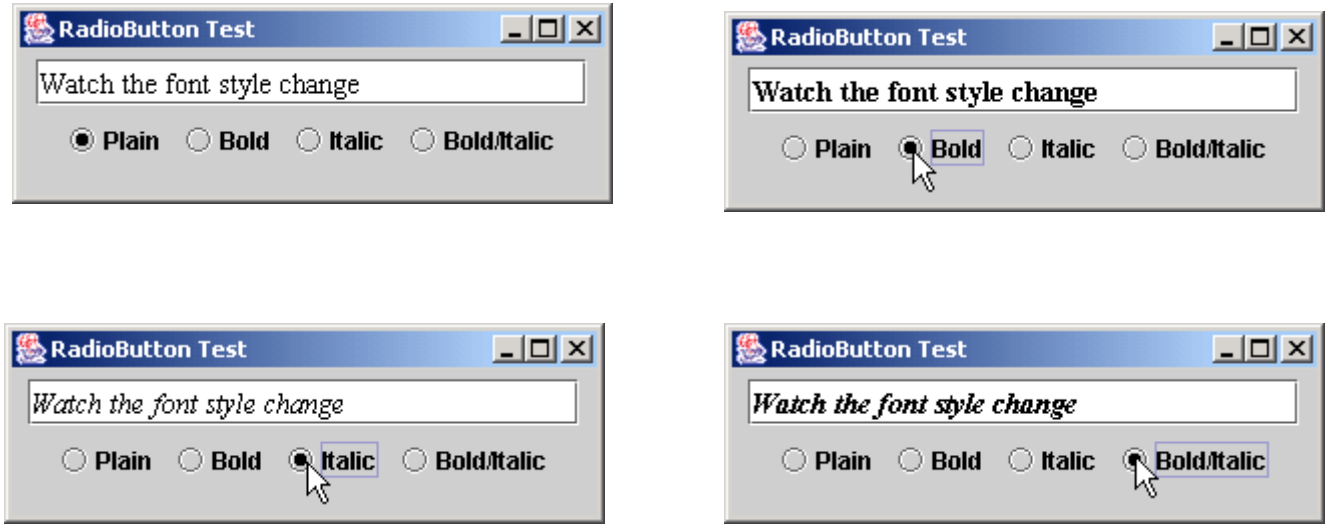
Change `JTextField` font, depending on which `JCheckBox` was selected

CheckBoxTest.java

Line 54

Line 65

Exemplo com Radio Button



```
1   //  RadioButtonTest.java
2   // Creating radio buttons using ButtonGroup and JRadioButton.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class RadioButtonTest extends JFrame {
8      private JTextField field;
9      private Font plainFont, boldFont, italicFont, boldItalicFont;
10     private JRadioButton plainButton, boldButton, italicButton,
11        boldItalicButton;
12     private ButtonGroup radioGroup;
13
14     // create GUI and fonts
15     public RadioButtonTest()
16     {
17        super( "RadioButton Test" );
18
19        // get content pane and set its layout
20        Container container = getContentPane();
21        container.setLayout( new FlowLayout() );
22
23        // set up JTextField
24        field = new JTextField( "Watch the font style change", 25 );
25        container.add( field );
26
```

RadioButtonTest.java

Declare four `JRadioButton` instances

Line 12

`JRadioButton`s normally appear as a `ButtonGroup`

```
27          // create radio buttons
28          plainButton = new JRadioButton( "Plain", true );
29          container.add( plainButton );
30
31          boldButton = new JRadioButton( "Bold", false );
32          container.add( boldButton );
33
34          italicButton = new JRadioButton( "Italic", fals
35          container.add( italicButton );
36
37          boldItalicButton = new JRadioButton( "Bold/Italic", false );
38          container.add( boldItalicButton );
39
40          // create logical relationship between JRadioButt
41          radioGroup = new ButtonGroup();
42          radioGroup.add( plainButton );
43          radioGroup.add( boldButton );
44          radioGroup.add( italicButton );
45          radioGroup.add( boldItalicButton );
46
47          // create font objects
48          plainFont = new Font( "Serif", Font.PLAIN, 14 );
49          boldFont = new Font( "Serif", Font.BOLD, 14 );
50          italicFont = new Font( "Serif", Font.ITALIC, 14 );
51          boldItalicFont = new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
52          field.setFont( plainFont );  // set initial font
53
```

Instantiate `JRadioButton`s for manipulating `JTextField` text font

`JRadioButton`s belong to `ButtonGroup`

RadioButtonTest
.java

Lines 28-35

Lines 41-45

```
54          // register events for JRadioButtons
55          plainButton.addItemListener( new RadioButtonHandler( plainF
56          boldButton.addItemListener( new RadioButtonHandler( boldFon
57          italicButton.addItemListener(
58             new RadioButtonHandler( italicFont ) );
59          boldItalicButton.addItemListener(
60             new RadioButtonHandler( boldItalicFont ) );
61
62          setSize( 300, 100 );
63          setVisible( true );
64
65       } // end RadioButtonTest constructor
66
67       public static void main( String args[] )
68       {
69          RadioButtonTest application = new RadioButtonTest();
70          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
71       }
72
73       // private inner class to handle radio button events
74       private class RadioButtonHandler implements ItemListener {
75          private Font font;
76
77          public RadioButtonHandler( Font f )
78          {
79             font = f;
80          }
```

Register `JRadioButtons` to receive events from `RadioButtonHandler`

`RadioButtonTest.java`

Lines 55-60

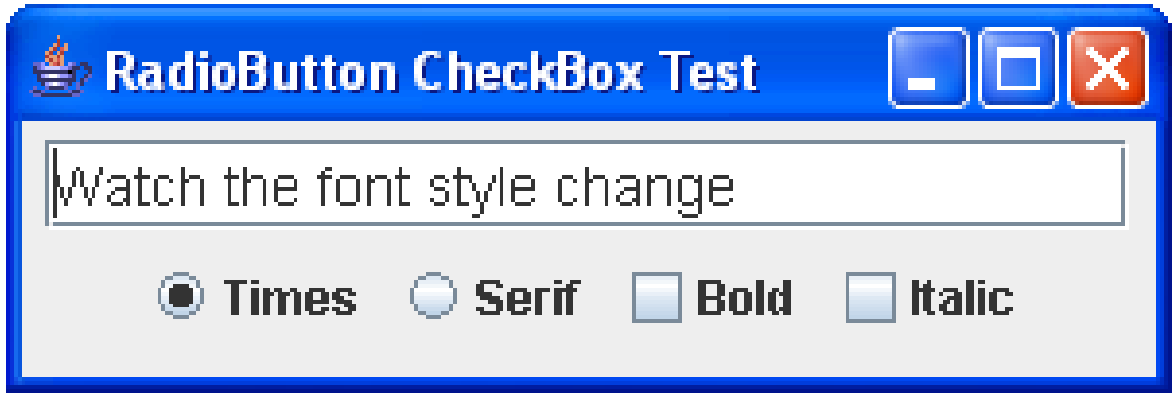
```
81
82        // handle radio button events
83        public void itemStateChanged( ItemEvent event )
84        {
85           field.setFont( font );
86        }
87
88     } // end private inner class RadioButtonHandler
89
90  } // end class RadioButtonTest
```

When user selects `JRadioButton`, `RadioButtonHandler` invokes method `itemStateChanged` of all registered listeners

Set font corresponding to `JRadioButton` selected

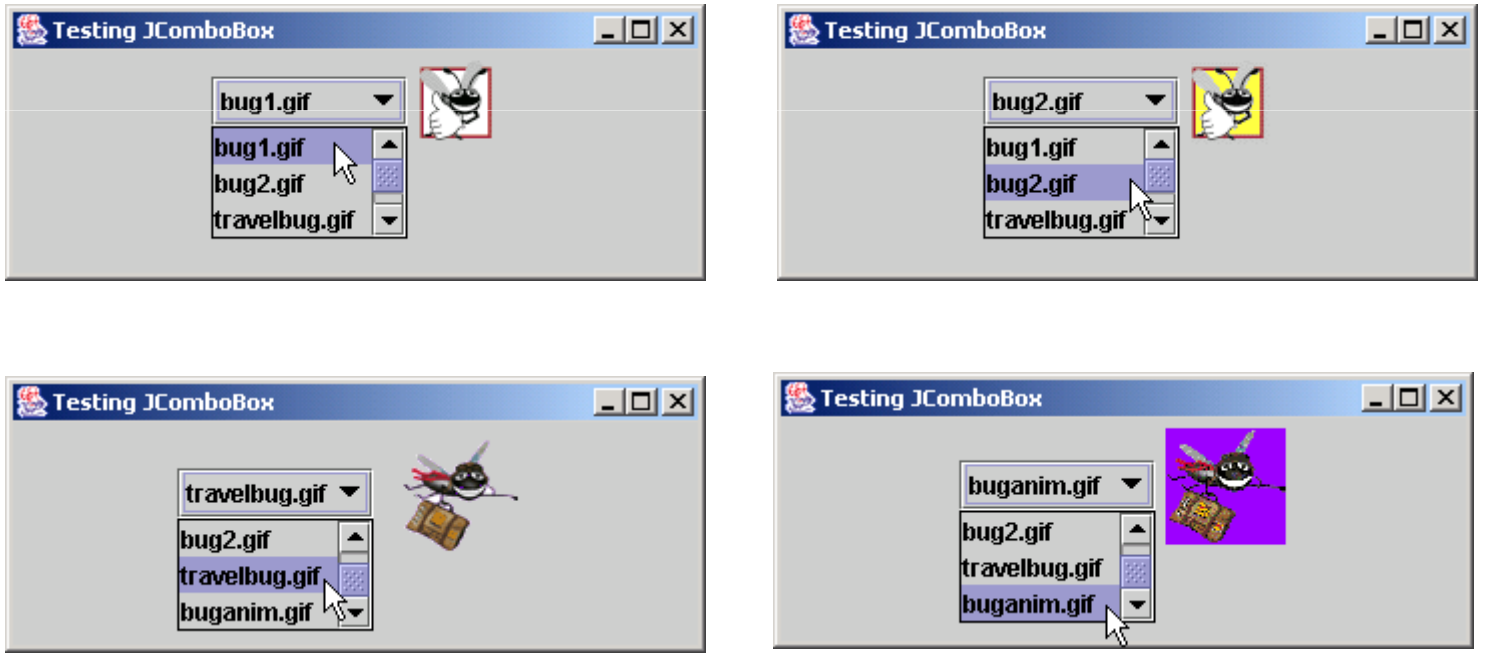
Exercício

- Criar um programa RadioButton CheckBox Test, com dois “radio buttons” com tipos de fontes: “Serif” e “Times” e dois “check box”: “Bold” e “Italic”
  - Dica: Crie um único handler como uma inner class e utilize a referência this para acessar as informações necessárias para contruir o objeto fonte



13.9 JComboBox

- JComboBox
  - List of items from which user can select
  - Also called a *drop-down list*



```
1       // ComboBoxTest.java
2       // Using a JComboBox to select an image to display.
3       import java.awt.*;
4       import java.awt.event.*;
5       import javax.swing.*;
6
7       public class ComboBoxTest extends JFrame {
8          private JComboBox imagesComboBox;
9          private JLabel label;
10
11        private String names[] =
12           { "bug1.gif", "bug2.gif",  "travelbug.gif", "buganim.gif" };
13        private Icon icons[] = { new ImageIcon( names[ 0 ] ),
14           new ImageIcon( names[ 1 ] ), new ImageIcon( names[ 2 ] ),
15           new ImageIcon( names[ 3 ] ) };
16
17        // set up GUI
18        public ComboBoxTest()
19        {
20           super( "Testing JComboBox" );
21
22           // get content pane and set its layout
23           Container container = getContentPane();
24           container.setLayout( new FlowLayout() );
25
```

ComboBoxTest.java

```
26          // set up JComboBox and register its event handler
27          imagesComboBox = new JComboBox( names );
28          imagesComboBox.setMaximumRowCount( 3 );
29          imagesComboBox.addItemListener(
30
31             new ItemListener() {  // anonymous inner class
32
33                // handle JComboBox event
34                public void itemStateChanged( ItemEvent eve
35                {
36                   // determine whether check box selected
37                   if ( event.getStateChange() == ItemEvent.SELECTED )
38                      label.setIcon( icons[
39                         imagesComboBox.getSelectedIndex() ] );
40                }
41
42             }  // end anonymous inner class
43
44          ); // end call to addItemListener
45
46          container.add( imagesComboBox );
47
48          // set up JLabel to display ImageIcons
49          label = new JLabel( icons[ 0 ] );
50          container.add( label );
51
```

Instantiate `JComboBox` to show three `String`s from `names` array at a time

Register `JComboBox` to receive events from anonymous `ItemListener`

When user selects item in `JComboBox`, `ItemListener` invokes method `itemStateChanged` of all registered listeners

Set appropriate `Icon` depending on user selection

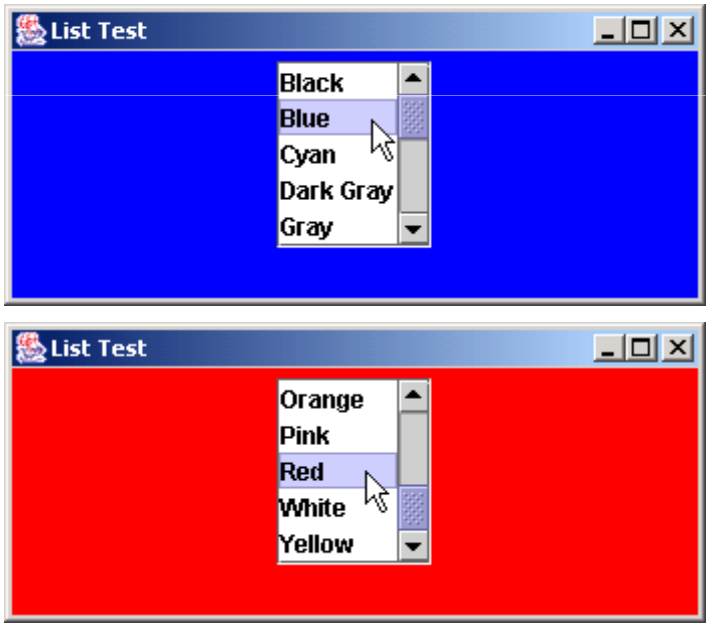
ComboBoxTest.java

```
52          setSize( 350, 100 );
53          setVisible( true );
54
55       } // end ComboBoxTest constructor
56
57       public static void main( String args[] )
58       {
59          ComboBoxTest application = new ComboBoxTest();
60          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
61       }
62
63    } // end class ComboBoxTest
```

ComboBoxTest.java

JList

- List
  - Series of items
  - user can select one or more items
  - Single-selection vs. multiple-selection
  - `JList`



```
1     // ListTest.java
2     // Selecting colors from a JList.
3     import java.awt.*;
4     import javax.swing.*;
5     import javax.swing.event.*;
6
7     public class ListTest extends JFrame {
8        private JList colorList;
9        private Container container;
10
11       private final String colorNames[] = { "Black", "Blue", "Cyan",
12          "Dark Gray", "Gray", "Green", "Light Gray", "Magenta",
13          "Orange", "Pink", "Red", "White", "Yellow" };
14
15       private final Color colors[] = { Color.BLACK, Color.BLUE, Color.CYAN,
16          Color.DARK_GRAY, Color.GRAY, Color.GREEN, Color.LIGHT_GRAY,
17          Color.MAGENTA, Color.ORANGE, Color.PINK, Color.RED, Color.WHITE,
18          Color.YELLOW };
19
20       // set up GUI
21       public ListTest()
22       {
23          super( "List Test" );
24
25          // get content pane and set its layout
26          container = getContentPane();
27          container.setLayout( new FlowLayout() );
```

ListTest.java

```
28
29          // create a list with items in colorNames array
30          colorList = new JList( colorNames );
31          colorList.setVisibleRowCount( 5 );
32
33          // do not allow multiple selections
34          colorList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
35
36          // add a JScrollPane containing JList to content pane
37          container.add( new JScrollPane( colorList ) );
38          colorList.addListSelectionListener(
39
40             new ListSelectionListener() {  // anonymous
41
42                // handle list selection events
43                public void valueChanged( ListSelectionEvent event )
44                {
45                   container.setBackground(
46                      colors[ colorList.getSelectedIndex() ] );
47                }
48
49             } // end anonymous inner class
50
51          ); // end call to addListSelectionListener
52
```

Use `colorNames` array to populate `JList`

`JList` allows single selections

Register `JList` to receive events from anonymous `ListSelectionListener`

`ListTest.java`

When user selects item in `JList`, `ListSelectionListener` invokes method `valueChanged` of all registered listeners

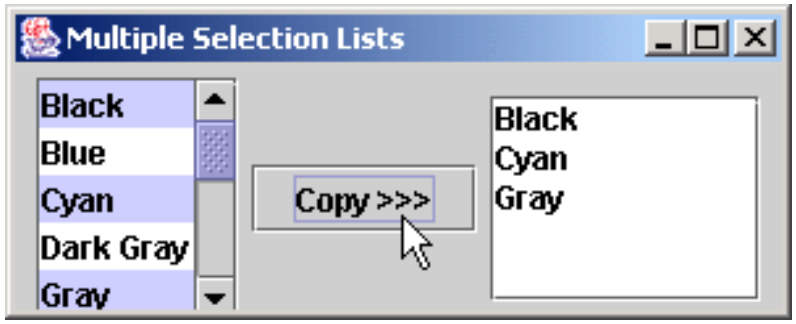
Set appropriate background depending on user selection

```
53          setSize( 350, 150 );
54          setVisible( true );
55
56       } // end ListTest constructor
57
58       public static void main( String args[] )
59       {
60          ListTest application = new ListTest();
61          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
62       }
63
64    } // end class ListTest
```

ListTest.java

13.11 Multiple-Selection Lists

- Multiple-selection list
  - Select many items from `JList`
  - Allows continuous range selection



```
1   // MultipleSelectionTest.java
2   // Copying items from one List to another.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class MultipleSelectionTest extends JFrame {
8      private JList colorList, copyList;
9      private JButton copyButton;
10     private final String colorNames[] = { "Black", "Blue", "Cyan",
11        "Dark Gray", "Gray", "Green", "Light Gray", "Magenta", "Orange",
12        "Pink", "Red", "White", "Yellow" };
13
14     // set up GUI
15     public MultipleSelectionTest()
16     {
17        super( "Multiple Selection Lists" );
18
19        // get content pane and set its layout
20        Container container = getContentPane();
21        container.setLayout( new FlowLayout() );
22
23        // set up JList colorList
24        colorList = new JList( colorNames );
25        colorList.setVisibleRowCount( 5 );
26        colorList.setSelectionMode(
27           ListSelectionModel.MULTIPLE_INTERVAL_SELECTION );
28        container.add( new JScrollPane( colorList ) );
```

MultipleSelectionTest.java

Use `colorNames` array to populate `JList`

`JList colorList` allows multiple selections

```
29
30          // create copy button and register its listener
31          copyButton = new JButton( "Copy >>>" );
32          copyButton.addActionListener(
33
34             new ActionListener() {  // anonymous inner class
35
36                // handle button event
37                public void actionPerformed( ActionEvent event )
38                {
39                   // place selected values in copyList
40                   copyList.setListData( colorList.getSelectedValues() );
41                }
42
43             } // end anonymous inner class
44
45          ); // end call to addActionListener
46
47          container.add( copyButton );
48
49          // set up JList copyList
50          copyList = new JList( );
51          copyList.setVisibleRowCount( 5 );
52          copyList.setFixedCellWidth( 100 );
53          copyList.setFixedCellHeight( 15 );
54          copyList.setSelectionMode(
55             ListSelectionModel.SINGLE_INTERVAL_SELECTION );
56          container.add( new JScrollPane( copyList ) );
```

MultipleSelectionTest.java

When user presses `JButton`, `JList copyList` adds items that user selected from `JList colorList`

`JList colorList` allows single selections

MultipleSelectionTest.java

```
57
58          setSize( 300, 130 );
59          setVisible( true );
60
61       } // end constructor MultipleSelectionTest
62
63       public static void main( String args[] )
64       {
65          MultipleSelectionTest application = new MultipleSelectionTest();
66          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
67       }
68
69    } // end class MultipleSelectionTest
```

Gerenciamento de Layout

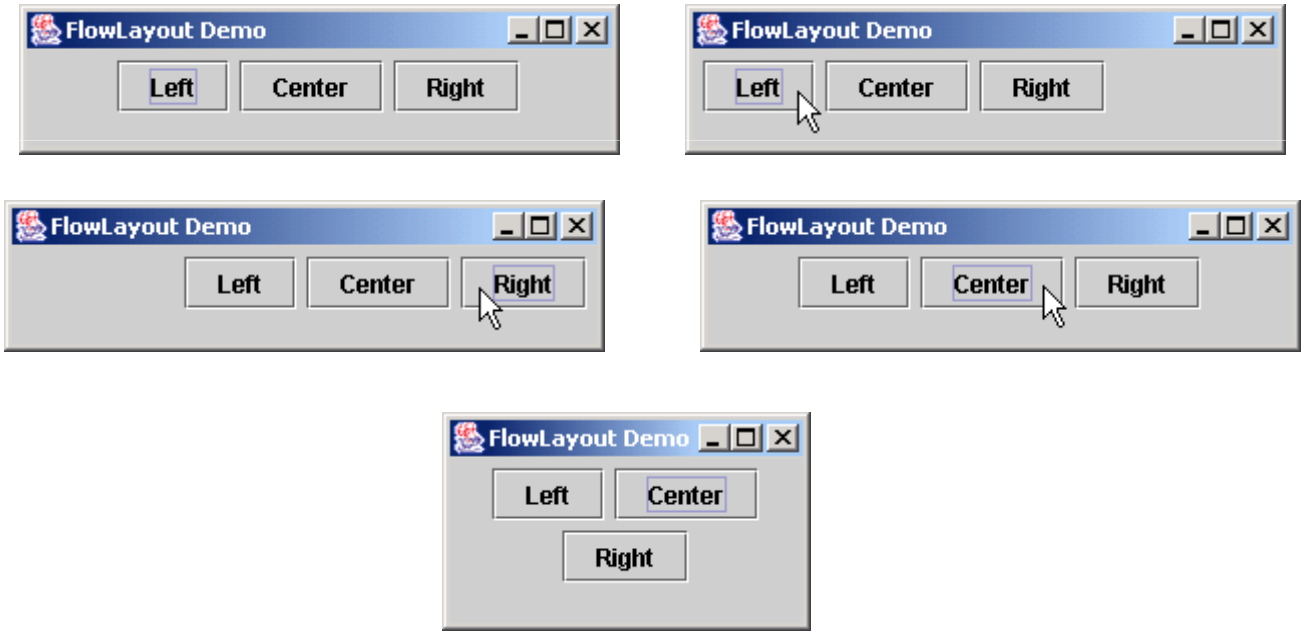
- Layout managers
  - Provided for arranging GUI components
  - Provide basic layout capabilities
  - Processes layout details
  - Programmer can concentrate on basic “look and feel”
  - Interface `LayoutManager`

Alguns Exemplos de Layout managers

Layout manager	Description
`FlowLayout`	Default for `java.awt.Applet`, `java.awt.Panel` and `javax.swing.JPanel`. Places components sequentially (left to right) in the order they were added. It is also possible to specify the order of the components by using the `Container` method `add`, which takes a `Component` and an integer index position as arguments.
`BorderLayout`	Default for the content panes of `JFrame`s (and other windows) and `JApplet`s. Arranges the components into five areas: `NORTH`, `SOUTH`, `EAST`, `WEST` and `CENTER`.
`GridLayout`	Arranges the components into rows and columns.

13.15.1 FlowLayout

- FlowLayout
  - Most basic layout manager
  - GUI components placed in container from left to right



```
1       //  FlowLayoutDemo.java
2       // Demonstrating FlowLayout alignments.
3       import java.awt.*;
4       import java.awt.event.*;
5       import javax.swing.*;
6
7       public class FlowLayoutDemo extends JFrame {
8          private JButton leftButton, centerButton, rightButton;
9          private Container container;
10        private FlowLayout layout;
11
12        // set up GUI and register button listeners
13        public FlowLayoutDemo()
14        {
15           super( "FlowLayout Demo" );
16
17           layout = new FlowLayout();
18
19           // get content pane and set its layout
20           container = getContentPane();
21           container.setLayout( layout );
22
23           // set up leftButton and register listener
24           leftButton = new JButton( "Left" );
25           container.add( leftButton );
```

FlowLayoutDemo.java

Set layout as FlowLayout

s 17 and 21

```
26          leftButton.addActionListener(
27
28             new ActionListener() {  // anonymous inner class
29
30                // process leftButton event
31                public void actionPerformed( ActionEvent event )
32                {
33                   layout.setAlignment( FlowLayout.LEFT );
34
35                   // realign attached components
36                   layout.layoutContainer( container );
37                }
38
39             } // end anonymous inner class
40
41          ); // end call to addActionListener
42
43          // set up centerButton and register listener
44          centerButton = new JButton( "Center" );
45          container.add( centerButton );
46          centerButton.addActionListener(
47
48             new ActionListener() {  // anonymous inner class
49
50                // process centerButton event
51                public void actionPerformed( ActionEvent event )
52                {
53                   layout.setAlignment( FlowLayout.CENTER );
54
```

When user presses left `JButton`, left align components

When user presses center `JButton`, center components

`Demo.java`

```
55                // realign attached components
56                layout.layoutContainer( container );
57             }
58          }
59       );
60
61       // set up rightButton and register listener
62       rightButton = new JButton( "Right" );
63       container.add( rightButton );
64       rightButton.addActionListener(
65
66          new ActionListener() {  // anonymous inner class
67
68             // process rightButton event
69             public void actionPerformed( ActionEvent event )
70             {
71                layout.setAlignment( FlowLayout.RIGHT );
72
73                // realign attached components
74                layout.layoutContainer( container );
75             }
76          }
77       );
78
79       setSize( 300, 75 );
80       setVisible( true );
```

FlowLayoutDemo.java

When user presses right JButton, right components

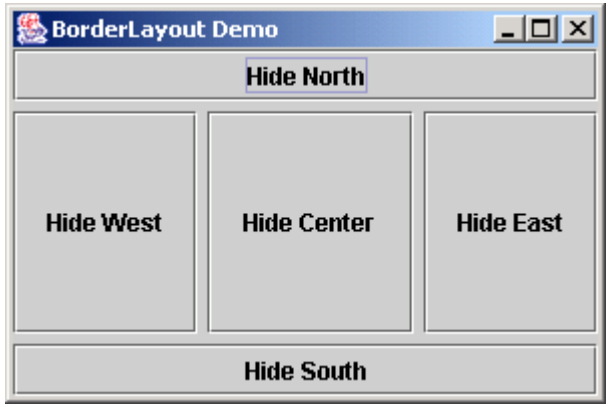
```
81
82       } // end constructor FlowLayoutDemo
83
84       public static void main( String args[] )
85       {
86          FlowLayoutDemo application = new FlowLayoutDemo();
87          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
88       }
89
90    } // end class FlowLayoutDemo
```

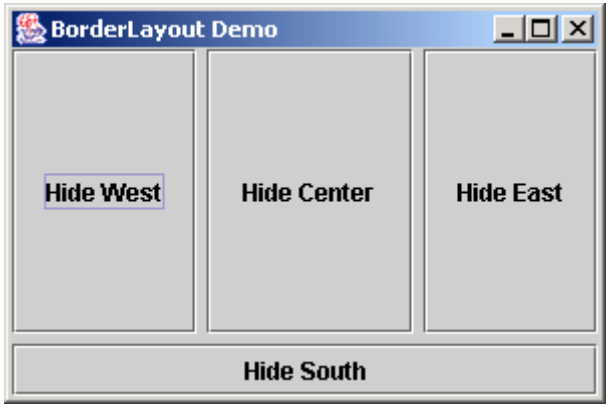
FlowLayoutDemo.java

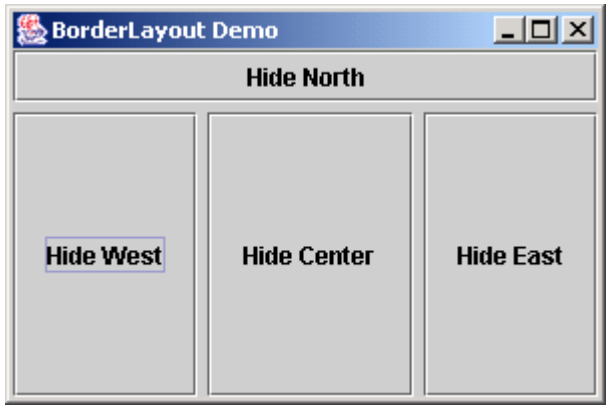
13.15.2 BorderLayout

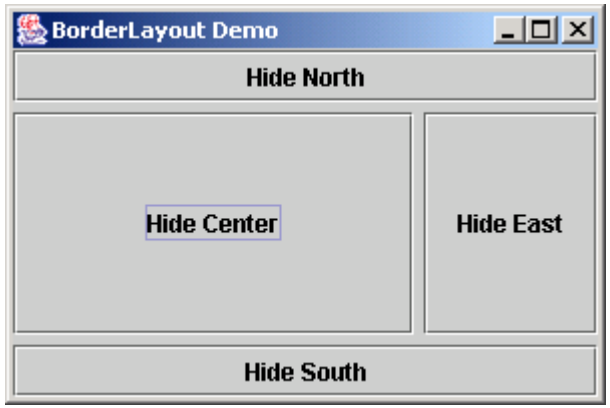
- BorderLayout
  - Arranges components into five regions
    - NORTH (top of container)
    - SOUTH (bottom of container)
    - EAST (left of container)
    - WEST (right of container)
    - CENTER (center of container)

Exemplo - BorderLayout

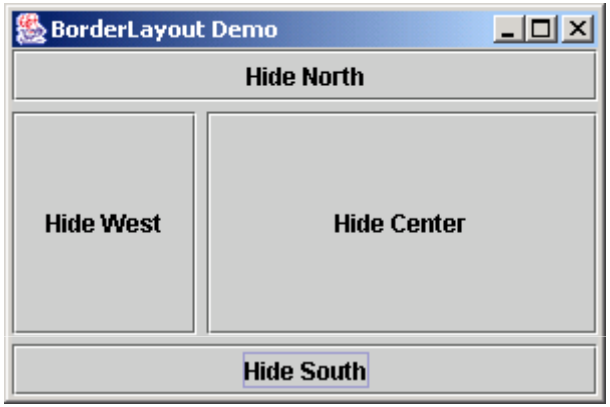


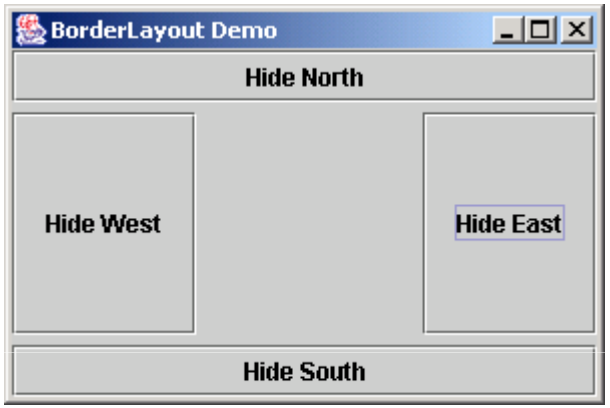






Exemplo - BorderLayout





```
1   // BorderLayoutDemo.java
2   // Demonstrating BorderLayout.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class BorderLayoutDemo extends JFrame implements ActionListener {
8      private JButton buttons[];
9      private final String names[] = { "Hide North", "Hide South",
10       "Hide East", "Hide West", "Hide Center" };
11     private BorderLayout layout;
12
13     // set up GUI and event handling
14     public BorderLayoutDemo()
15     {
16        super( "BorderLayout Demo" );
17
18        layout = new BorderLayout( 5, 5 ); // 5 pixel gaps
19
20        // get content pane and set its layout
21        Container container = getContentPane();
22        container.setLayout( layout );
23
24        // instantiate button objects
25        buttons = new JButton[ names.length ];
26
```

BorderLayoutDemo.java

Set layout as `BorderLayout` with 5-pixel horizontal and vertical gaps

```
27          for ( int count = 0; count < names.length; count++ ) {
28             buttons[ count ] = new JButton( names[ count ] );
29             buttons[ count ].addActionListener( this );
30          }
31
32          // place buttons in BorderLayout; order not important
33          container.add( buttons[ 0 ], BorderLayout.NORTH );
34          container.add( buttons[ 1 ], BorderLayout.SOUTH );
35          container.add( buttons[ 2 ], BorderLayout.EAST );
36          container.add( buttons[ 3 ], BorderLayout.WEST );
37          container.add( buttons[ 4 ], BorderLayout.CENTER );
38
39          setSize( 300, 200 );
40          setVisible( true );
41
42       } // end constructor BorderLayoutDemo
43
44       // handle button events
45       public void actionPerformed( ActionEvent event )
46       {
47          for ( int count = 0; count < buttons.length; count++ )
48
49             if ( event.getSource() == buttons[ count ] )
50                buttons[ count ].setVisible( false );
51             else
52                buttons[ count ].setVisible( true );
```

Place `JButton`s in regions specified by `BorderLayout`

`BorderLayoutDemo.java`

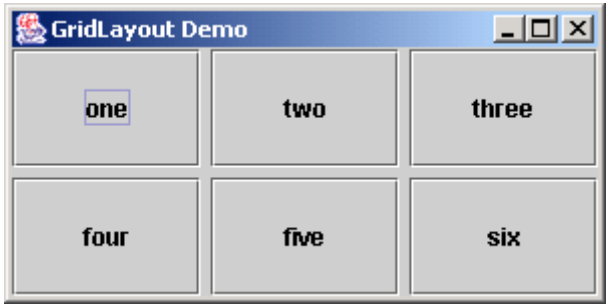
When `JButton`s are "invisible," they are not displayed on screen, and `BorderLayout` rearranges

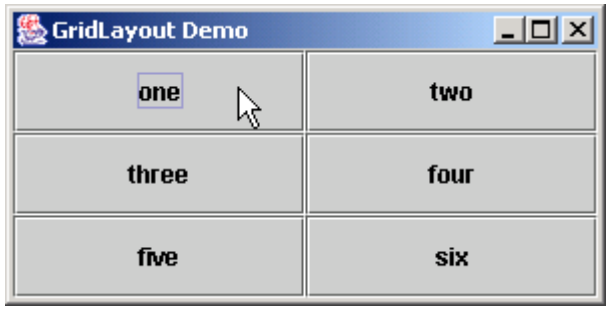
```
53
54          // re-layout the content pane
55          layout.layoutContainer( getContentPane() );
56       }
57
58       public static void main( String args[] )
59       {
60          BorderLayoutDemo application = new BorderLayoutDemo();
61          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
62       }
63
64    } // end class BorderLayoutDemo
```

BorderLayoutDemo.java

GridLayout

- GridLayout
  - Divides container into grid of specified row an columns
  - Components are added starting at top-left cell
    - Proceed left-to-fight until row is full





```
1       // GridLayoutDemo.java
2       // Demonstrating GridLayout.
3       import java.awt.*;
4       import java.awt.event.*;
5       import javax.swing.*;
6
7       public class GridLayoutDemo extends JFrame implements ActionListener {
8          private JButton buttons[];
9          private final String names[] =
10           { "one", "two", "three", "four", "five", "six" };
11        private boolean toggle = true;
12        private Container container;
13        private GridLayout grid1, grid2;
14
15        // set up GUI
16        public GridLayoutDemo()
17        {
18           super( "GridLayout Demo" );
19
20           // set up layouts
21           grid1 = new GridLayout( 2, 3, 5, 5 );
22           grid2 = new GridLayout( 3, 2 );
23
24           // get content pane and set its layout
25           container = getContentPane();
26           container.setLayout( grid1 );
```

GridLayoutDemo.java

Create `GridLayout grid1` with 2 rows and 3 columns

Create `GridLayout grid2` with 3 rows and 2 columns

```
27
28          // create and add buttons
29          buttons = new JButton[ names.length ];
30
31          for ( int count = 0; count < names.length; count++ ) {
32             buttons[ count ] = new JButton( names[ count ] );
33             buttons[ count ].addActionListener( this );
34             container.add( buttons[ count ] );
35          }
36
37          setSize( 300, 150 );
38          setVisible( true );
39
40       } // end constructor GridLayoutDemo
41
42       // handle button events by toggling between layouts
43       public void actionPerformed( ActionEvent event )
44       {
45          if ( toggle )
46             container.setLayout( grid2 );
47          else
48             container.setLayout( grid1 );
49
50          toggle = !toggle;   // set toggle to opposite value
51          container.validate();
52       }
```

GridLayoutDemo.java

Toggle current GridLayout when user presses JButton

```
53
54       public static void main( String args[] )
55       {
56          GridLayoutDemo application = new GridLayoutDemo();
57          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
58       }
59
60    } // end class GridLayoutDemo
```

GridLayoutDemo.java

Panels

- Panel
  - Helps organize components
  - Class `JPanel` is `JComponent` subclass
  - May have components (and other panels) added to them

```
1   // PanelDemo.java
2   // Using a JPanel to help lay out components.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class PanelDemo extends JFrame {
8      private JPanel buttonPanel;
9      private JButton buttons[];
10
11    // set up GUI
12    public PanelDemo()
13    {
14       super( "Panel Demo" );
15
16       // get content pane
17       Container container = getContentPane();
18
19       // create buttons array
20       buttons = new JButton[ 5 ];
21
22       // set up panel and set its layout
23       buttonPanel = new JPanel();
24       buttonPanel.setLayout( new GridLayout( 1, buttons.length ) );
25
```

PanelDemo.java

Line 23

Create `JPanel` to hold `JButton`s

```
26          // create and add buttons
27          for ( int count = 0; count < buttons.length; count++ ) {
28             buttons[ count ] = new JButton( "Button " + ( count + 1 ) );
29             buttonPanel.add( buttons[ count ] );
30          }
31
32          container.add( buttonPanel, BorderLayout.SOUTH );
33
34          setSize( 425, 150 );
35          setVisible( true );
36
37       } // end constructor PanelDemo
38
39       public static void main( String args[] )
40       {
41          PanelDemo application = new PanelDemo();
42          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
43       }
44
45    } // end class PanelDemo
```

Add JButtons to JPanel

Add JPanel to SOUTH region of Container

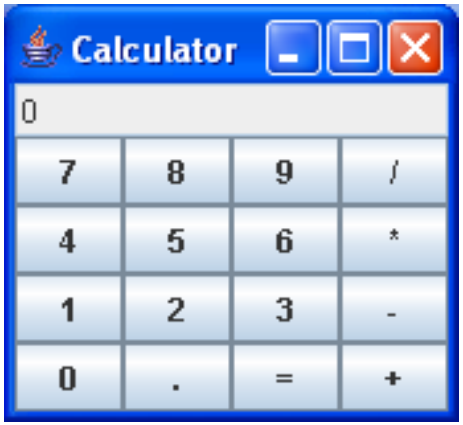
PanelDemo.java

Line 29

Line 32

Exercício

- Crie uma calculadora simples usando dois “panels” (1 panel com BorderLayout, outro com GridLayout 4x 4)
- Crie dois handlers um para números e ponto(insere no campo de texto), outro para comandos (+, - ,*,/,=)
- Use as classes wrapper para transformar texto em número e vice-versa



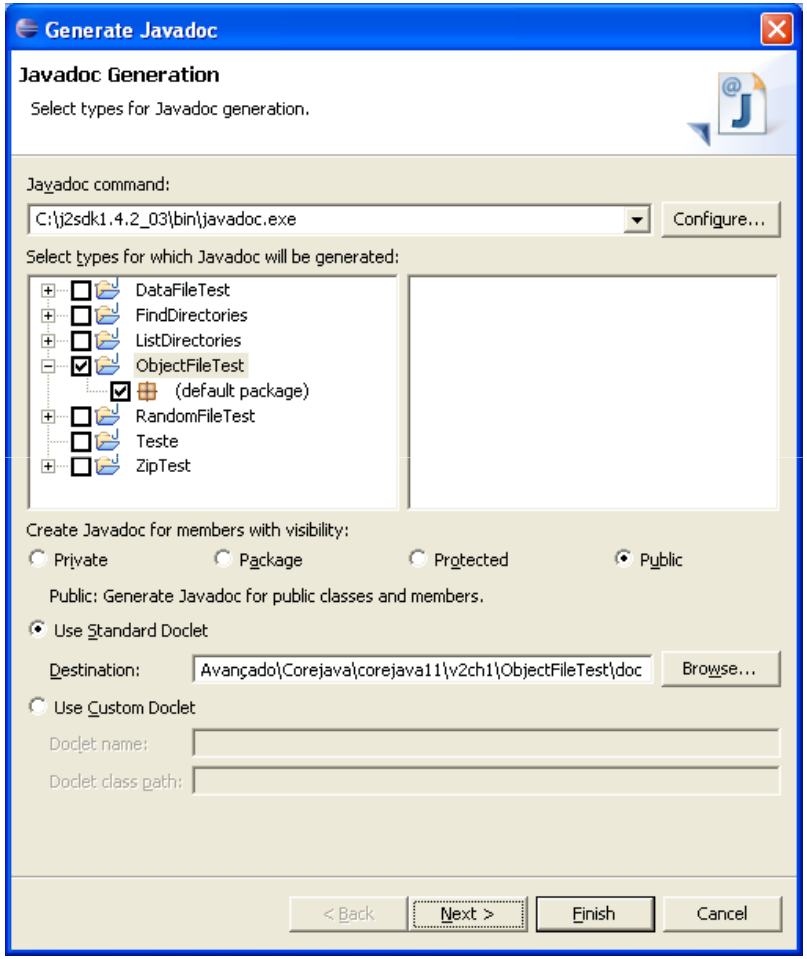
Documentação Automática de Programas Java

- Documentação Navegável em HTML

- Todas as classes da biblioteca Java tem documentação em HTML e suas classes também podem ter….

5.2. 10. Javadoc: Documentação automática em java

- Criação automática de documentação para programas Java com base no código-fonte e em modelos disponíveis. Base da documentação do própria Java.



Documentação de Elementos em Javadoc

- Elementos podem ser classes, interfaces, campos, construtores e comentários de métodos. Formato Geral:

```
Line 1: /** Description of the element being documented
 Line 2: * @tagA Description of tagA
 Line 3: * @tagB Description of tagB
 Line 4: */
```

Exemplo Comentário de Método/Construtor

```
/**
Returns an Image object that can then be painted on the screen.
The url argument must specify an absolute {@link URL}. The name
* argument is a specifier that is relative to the url argument.
* <p>
* This method always returns immediately, whether or not the
* image exists. When this applet attempts to draw the image on
* the screen, the data will be loaded. The graphics primitives
* that draw the image will incrementally paint on the screen.
*
* @param url an absolute URL giving the base location of the image
* @param name the location of the image, relative to the url argument
* @return the image at the specified URL
* @see Image
*/
public Image getImage(URL url, String name) {
        try {           return getImage(new URL(url, name));
        } catch (MalformedURLException e) {         return null;        }       }
```

Documentação de Elementos em Javadoc

- Elementos podem ser classes, interfaces, métodos e variáveis
- O comentário de um elemento deve ser logo anterior ao elemento. Caso contrário será desconsiderado pelo javadoc
- Um erro comum é colocar algum comando entre o comentário e o elemento. Por exemplo, um comando de importação entre o comentário de uma classe e a própria classe.
- /**
- */
- import java.lang.*;
- public class MinhaClasse
- Correto:
- /**
- */
- public class MinhaClasse

Comentários sobre Pacotes

- Usar arquivo package-info.java ou package.html
  - Preferível utilizar package-info.java (a partir de jdk 1.5)
- Exemplo package-info.java:

```
/**
  * Provides the classes necessary to create an applet and the classes an applet uses to communicate with its
      applet context.
 * <p>
*The applet framework involves two entities:
* the applet and the applet context. An applet is an embeddable window (see the
* {@link java.awt.Panel} class) with a few extra methods that the applet context
* can use to initialize, start, and stop the applet.
*
* @since 1.0
* @see java.awt
*/
package java.lang.applet;
```

Comentários sobre Pacotes

- Exemplo arquivo Package.html:

<HTML>

<BODY> Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.

<p>

The applet framework involves two entities: the applet and the applet context.

An applet is an embeddable window (see the {@link java.awt.Panel} class)

with a few extra methods that the applet context can use to initialize, start, and stop the applet.

@since 1.0

@see java.awt

</BODY>

</HTML>

Executando o Javadoc e campos específicos

- **JavaDoc**
  - Used to generate on-line documentation

```
javadoc Foo.java Bar.java
```

  - JavaDoc 1.4 Home Page
    - http://java.sun.com/j2se/1.4/docs/tooldocs/javadoc/

- **@author**
  - Specifies the author of the document
  - Must use `javadoc -author ...` to generate in output

```
/** Description of some class ...
 *
 * @author <A HREF="mailto:brown@lmbrown.com">
 *         Larry Brown</A>
 */
```

- **@version**
  - Version number of the document
  - Must use `javadoc -version ...` to generate in output
- **@param**
  - Documents a method argument
- **@return**
  - Documents the return type of a method

Mais comandos Javadoc

- **-author**
  - Includes author information (omitted by default)
- **-version**
  - Includes version number (omitted by default)
- **-noindex**
  - Tells `javadoc` not to generate a complete index
- **-notree**
  - Tells `javadoc` not to generate the tree.html class hierarchy
- **-link, -linkoffline**
  - Tells `javadoc` where to look to resolve links to other packages

```
-link http://java.sun.com/j2se/1.3/docs/api
-linkoffline http://java.sun.com/j2se/1.3/docs/api
             c:\jdk1.3\docs\api
```

Um Exemplo - Javadoc

```
/** Ship example to demonstrate OOP in Java.
 *
 * @author <A HREF="mailto:brown@corewebprogramming.com">
 *         Larry Brown</A>
 * @version 2.0
 */

public class Ship {
  private double x=0.0, y=0.0, speed=1.0, direction=0.0;
  private String name;

  /** Build a ship with specified parameters. */

  public Ship(double x, double y, double speed,
              double direction, String name) {
    setX(x);
    setY(y);
    setSpeed(speed);
    setDirection(direction);
    setName(name);
  }
```

Linha de comando

```
> javadoc -linkoffline http://java.sun.com/j2se/1.3/docs/api
                        c:\jdk1.3\docs\api
          -author -version -noindex -notree Ship.java
```

Linha de comando

> javadoc –author –version –noindex –notree Ship.java

Resultado


Ship.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Class Deprecated Help
FRAMES NO FRAMES
Class Ship
java.lang.Object
|
+--Ship
public class Ship
extends Object
Ship example to demonstrate OOP in Java.
Version:
2.0
Author:
Larry Brown
Constructor Summary
Ship(double x, double y, double xspeed, double direction, String name)
Build a ship with specified parameters.
Ship(String name)
Build a ship with default values (x=0, y=0, speed=1.0, direction=0.0).
Method Summary
My Computer

Exercício sobre Documentação Automática

- Escreva comentários para as classes, seus campos e métodos do exemplo Ship do projeto da aula 1
- Gerar a documentação automática referente ao exemplo do projeto Ship da aula 1