
Programação Orientada a Objetos

Prof. Paulo André Castro

pauloac@ita.br

www.comp.ita.br/~pauloac

ITA – Stefanini

Planejamento

- Aula 1
 - Introdução
 - Conceitos Básicos
 - Classe, Objeto, Método, Herança, interfaces, polimorfismo, Encapsulamento
 - Introdução a linguagem Java
- **Aula 2**
 - Modelagem de Programas Orientada a Objetos
 - Introdução a Padrões de Projeto (Design Patterns)
 - Introdução a Ambientes Integrados de Desenvolvimento
- **Aula 3**
 - **Programando Interfaces Gráficas com Java I**
- Aula 4
 - Programando Interfaces Gráficas com Java - II

Sumário de Hoje

Introdução

Overview dos Componentes Swing

Tratamento de Eventos

Mais componentes Swing

JButton

JCheckBox and JRadioButton

JComboBox

JList

Listas de Seleção Múltipla (Multiple-Selection Lists)

Tratamento de Eventos do Mouse

Classes Adapter

Tratamento de Eventos de Tecla

Gerenciamento de Layout

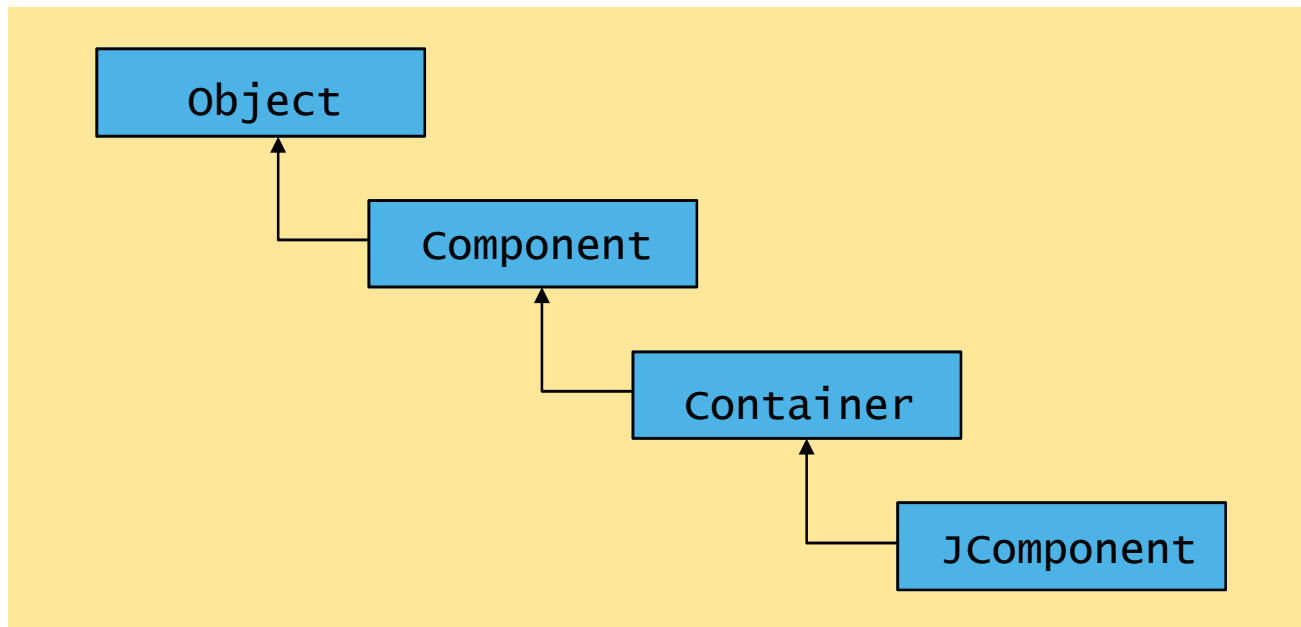
Introdução

- Graphical User Interface (GUI)
 - Swing permite diferentes “estilos” de programa (“look” and “feel”)
 - Provê componentes que fornecem funcionalidades comuns em GUI: botões, 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 Comportes Swing



Overview of Swing Components

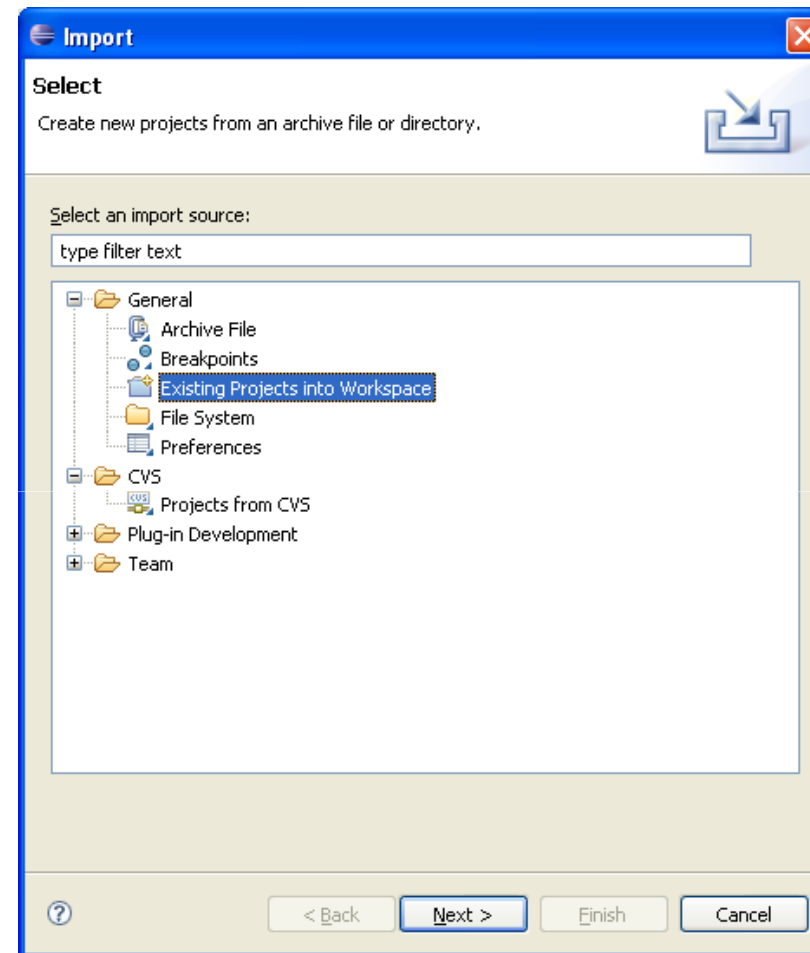
- **Class Component**
 - Contains method `paint` for drawing Component onscreen
- **Class Container**
 - Collection of related components
 - Contains method `add` for adding components
- **Class JComponent**
 - *Pluggable look and feel* for customizing look and feel
 - Shortcut keys (*mnemonics*)
 -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

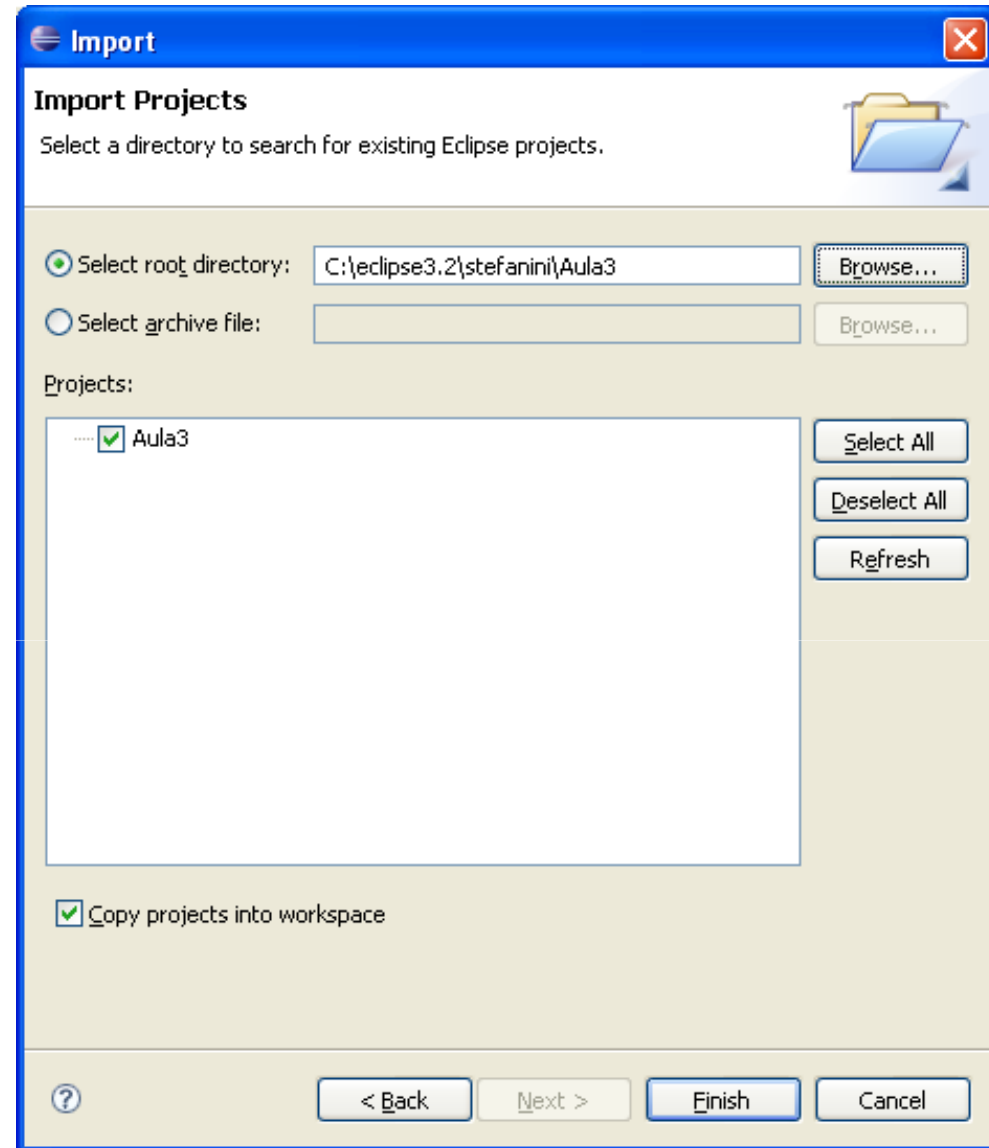
Importando Projetos Eclipse para seu Workspace

Menu: File | Import



Copie o projeto
para seu
Workspace.

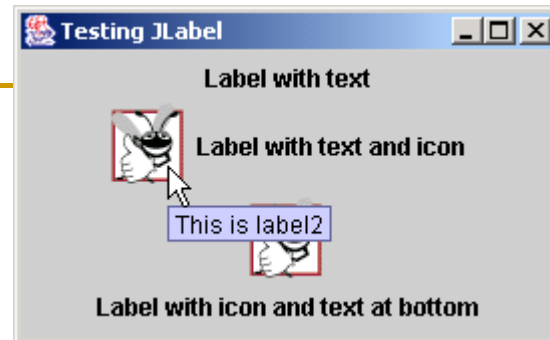
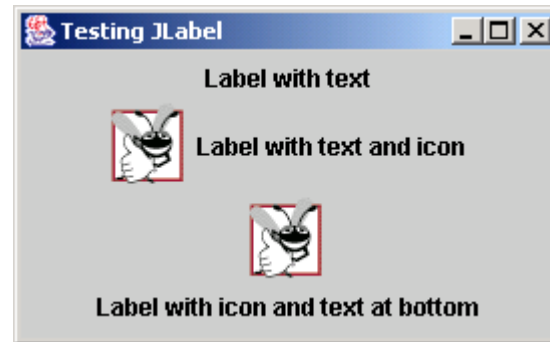
Cuidado por
Default é feita
apenas linkagem



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

Declare three JLabels

Line 21

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

est.java

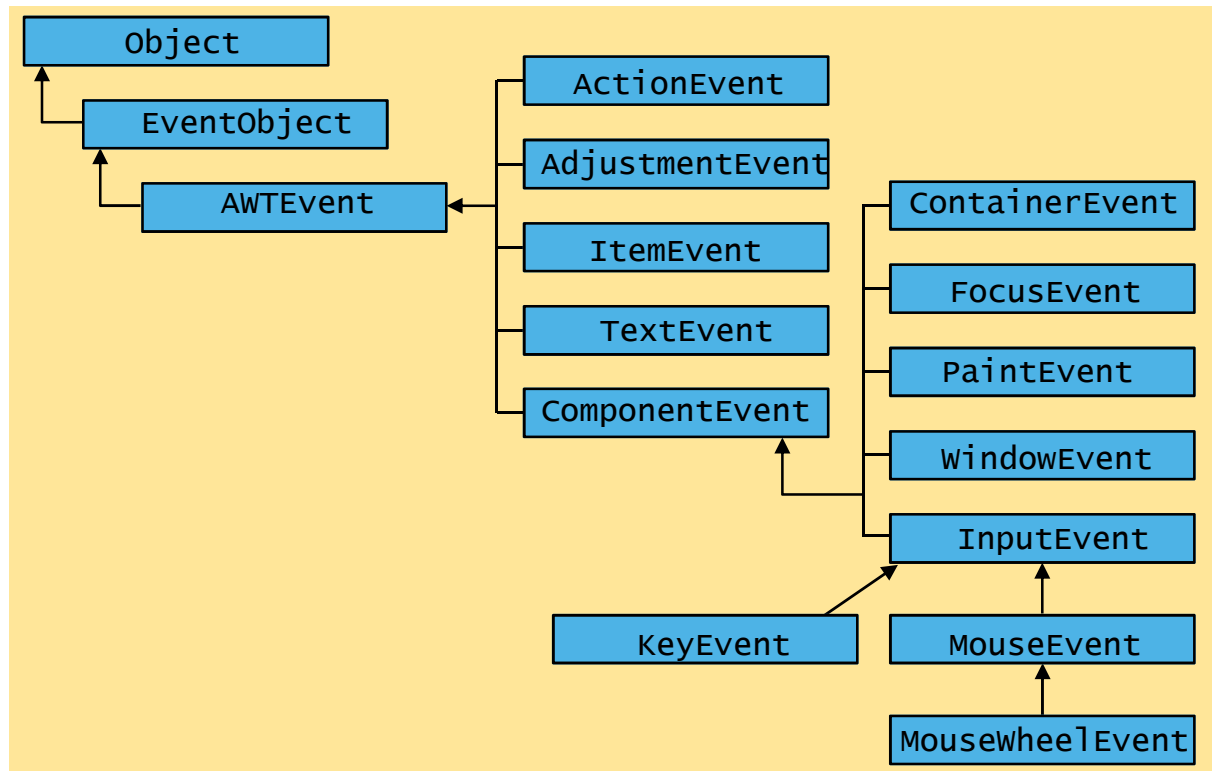
Lines 16-17

Lines 32-37

13.4 Event Handling

- 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`

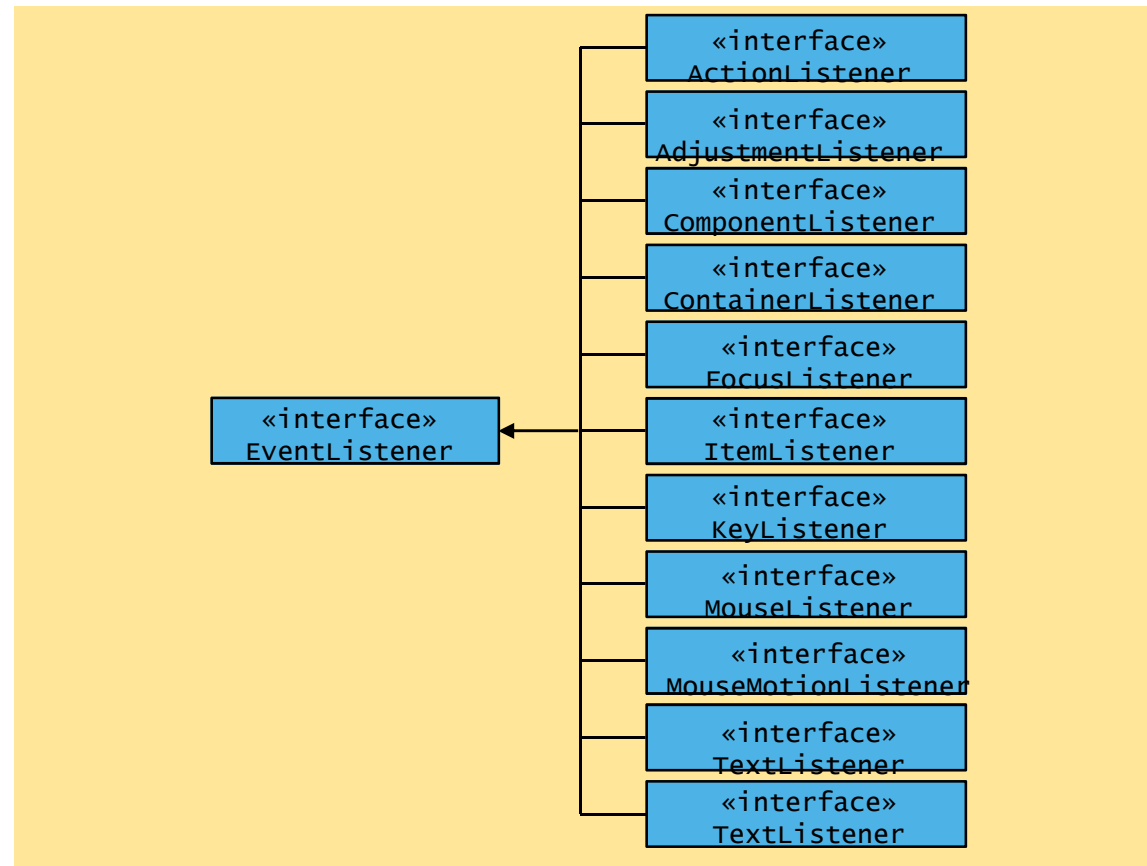
Some event classes of package `java.awt.event`



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)
 - 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`



TextFields

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

TextFieldTest.java



TextFieldTest.java



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
```

Lines 8-9

Declare three
JTextFields and one
JPasswordField

Line 20

Line 24

First JTextField
contains empty string

Second JTextField contains
text "Enter text here"

```

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

Lines 39-42

```

54 // private inner class for event handling
55 private class TextFieldHandler implements ActionListener {
56
57 // process textfield events
58 public void actionPerformed( ActionEvent event )
59 {
60     String string = "";
61
62 // user pressed Enter in JTextField textField1
63 if ( event.getSource() == textField1 )
64     string = "textField1: " + event.getActionCommand();
65
66 // user pressed Enter in JTextField textField2
67 else if ( event.getSource() == textField2 )
68     string = "textField2: " + event.getActionCommand();
69
70 // user pressed Enter in JTextField textField3
71 else if ( event.getSource() == textField3 )
72     string = "textField3: " + event.getActionCommand();
73
74 // user pressed Enter in JTextField passwordField
75 else if ( event.getSource() == passwordField ) {
76     string = "passwordField: " +
77         new String( passwordField.getPassword() );
78 }
79     JOptionPane.showMessageDialog( null, string );
80
81 } // end method actionPerformed
82
83 } // end private inner class TextFieldHandler
84
85 } // end class TextFieldTest
86
87

```

Every TextFieldHandler instance is an ActionListener

Method actionPerformed invoked when user presses Enter in GUI field

TextFieldTest.java

Line 56

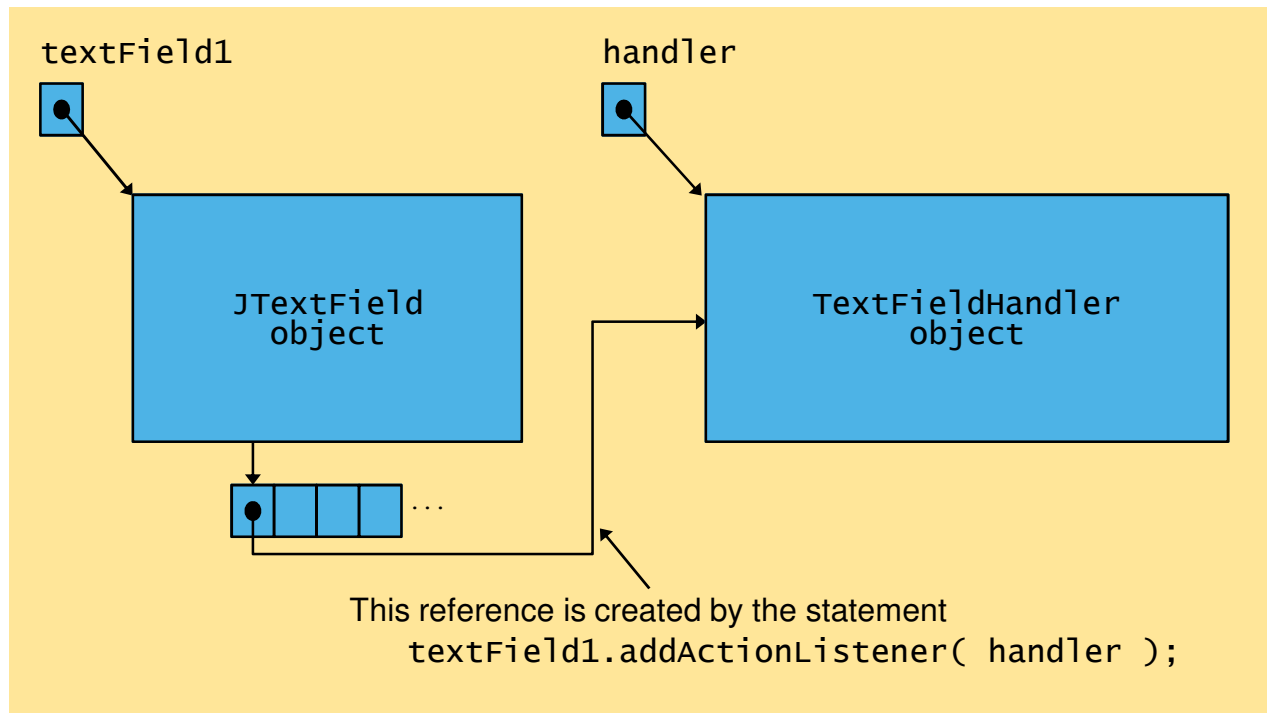
Line 59

Como o tratamento de eventos funciona?

■ Duas questões

- 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

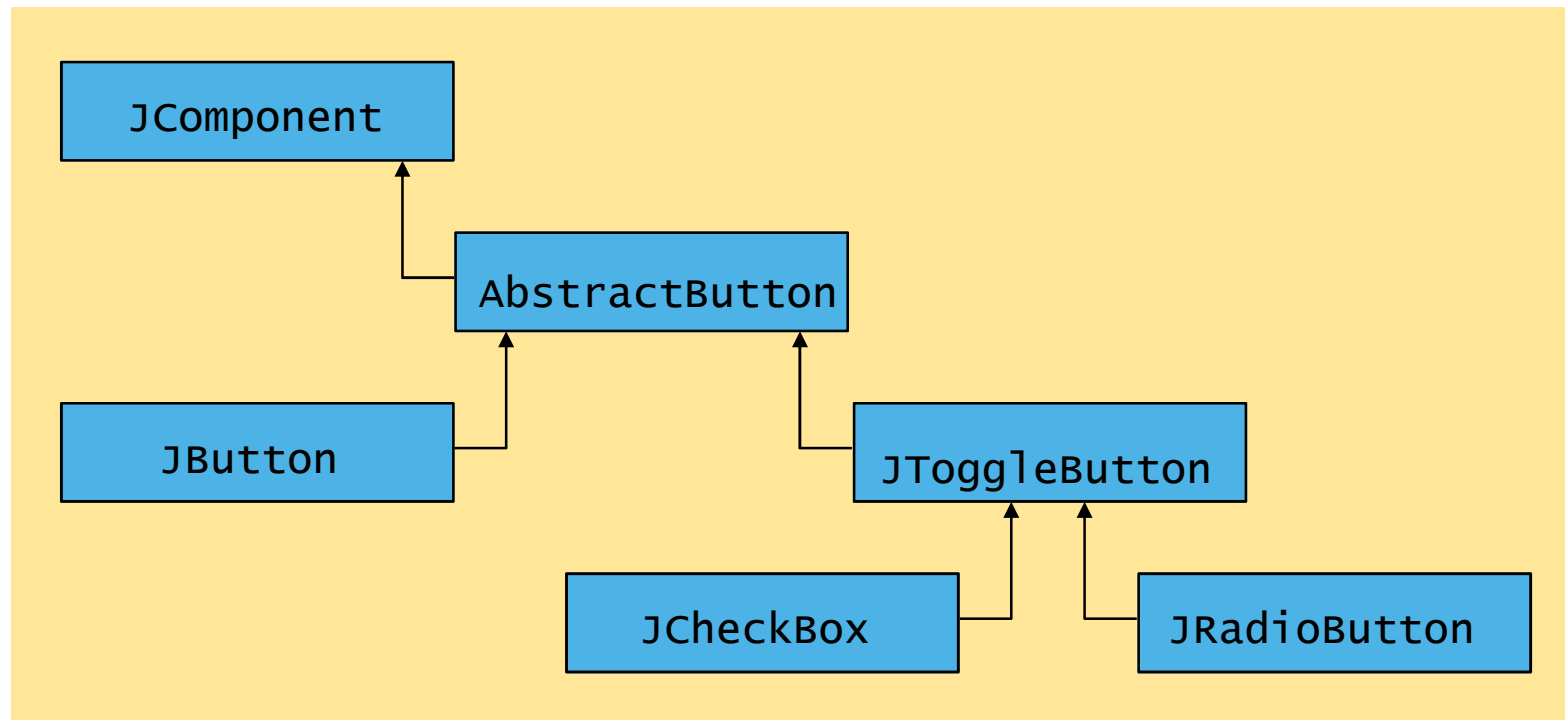
- Altere o programa TextFieldTest para que seja alterado a barra de título da janela, para o valor do campo alterado, qualquer que seja tal campo
 - Dica: O 'handler' precisará de uma referência para a janela.

JButton

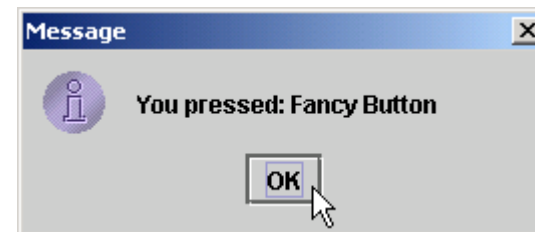
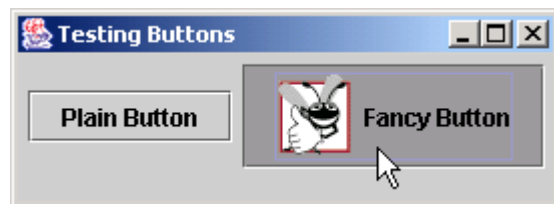
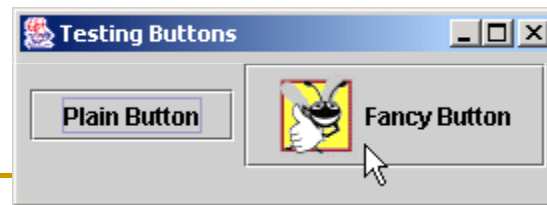
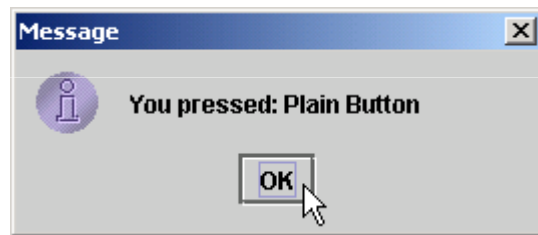
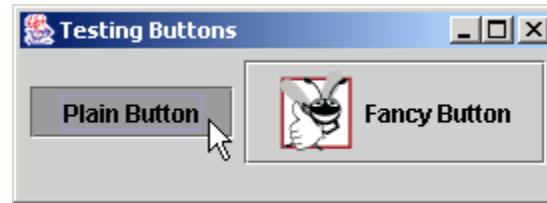
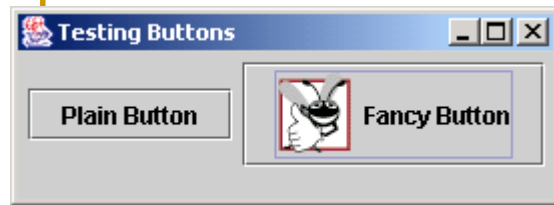
■ Button

- 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 );

```

ButtonTest.java

Line 8

Line 20

Lines 24-26

Create two references to JButton instances

Instantiate JButton with text

Instantiate JButton with image and rollover image

```

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 JButtons to receive
events from ButtonHandler

ButtonTest.java

Line 31

Lines 32-33

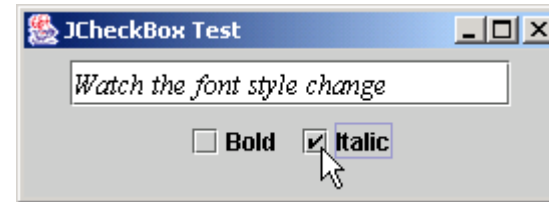
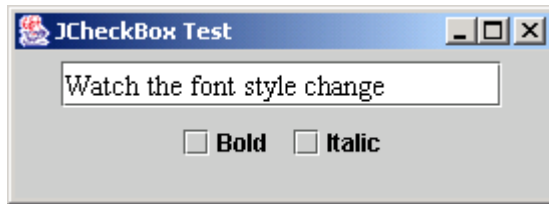
Line 50

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

JCheckBox and JRadioButton

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

CheckBoxTest.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

Line 22

Declare two JCheckBox instances

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 JCheckBoxs for bolding and italicizing JTextField text, respectively

Register JCheckBoxs 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.ITALIC : Font.PLAIN;
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 `itemStateChanged` of all registered listeners

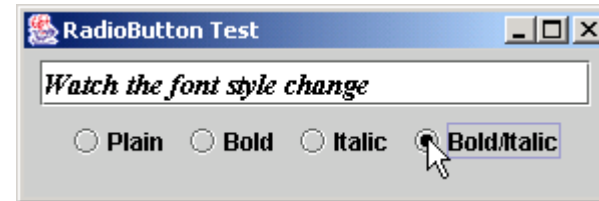
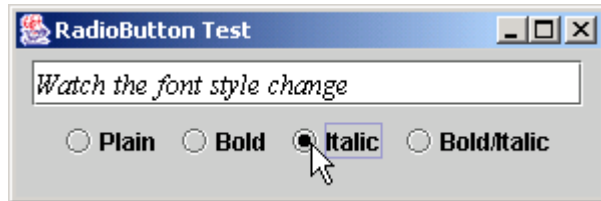
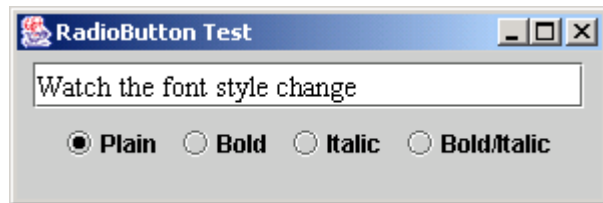
Change JTextField font, depending on which JCheckBox was selected

CheckBoxTest.java

Line 54

Line 65

Exemplo com Radio Button



RadioButtonTest.java

Lines 10-11

```
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
```

Declare four JRadioButton instances

JRadioButtons 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", false );
35 container.add( italicButton );
36
37 boldItalicButton = new JRadioButton( "Bold/Italic", false );
38 container.add( boldItalicButton );
39
40 // create logical relationship between JRadioButtons
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
```

RadioButtonTest
.java

Lines 28-35

Instantiate JRadioButtons for
manipulating JTextField text font

JRadioButtons belong
to ButtonGroup

```

54 // register events for JRadioButtons
55 plainButton.addItemListener( new RadioButtonHandler( plainFont ) );
56 boldButton.addItemListener( new RadioButtonHandler( boldFont ) );
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

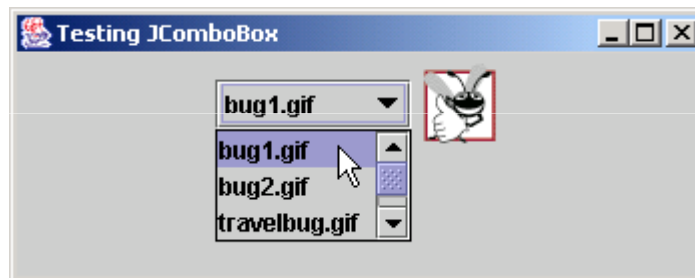
Exercício

- 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*



ComboBoxTest.java

```
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
```

```

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 event )
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 Strings 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

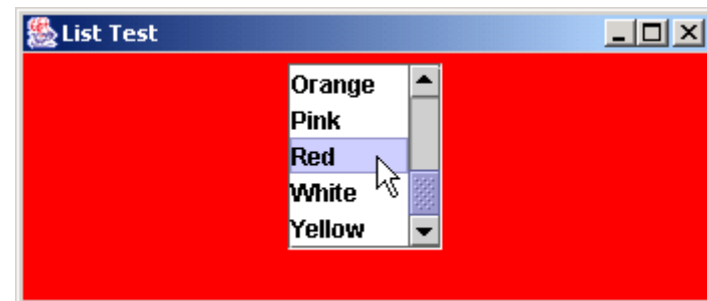
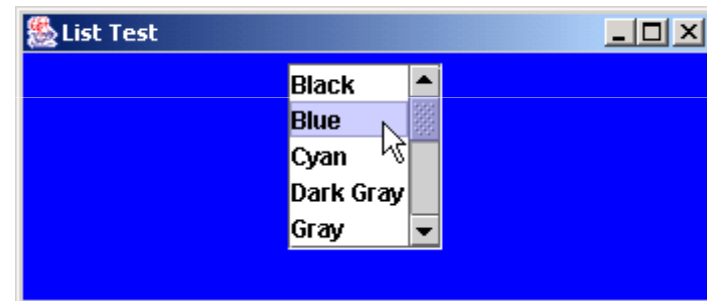
```
52     setSize( 350, 100 );
53     setVisible( true );
54
55     } // end ComboBoxTest constructor
56
57     public static void main( String args[] )
58     {
59         ComboBoxTest application = new ComboBoxTest();
60         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
61    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



ListTest.java

```
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() );
```

28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

```
// create a list with items in colorNames array  
colorList = new JList( colorNames );  
colorList.setVisibleRowCount( 5 );
```

Use colorNames array
to populate JList

```
// do not allow multiple selections  
colorList.setSelectionMode( ListSelectionMode.SINGLE_SELECTION );
```

ListTest.java

```
// add a JScrollPane containing JList to content pane  
container.add( new JScrollPane( colorList ) );  
colorList.addListSelectionListener(
```

JList allows single selections

```
new ListSelectionListener() { // anonymous
```

Register JList to receive events from
anonymous ListSelectionListener

```
    // handle list selection events  
    public void valueChanged( ListSelectionEvent event )  
    {  
        container.setBackground(  
            colors[ colorList.getSelectedIndex() ] );  
    }  
} // end anonymous inner class
```

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

```
); // end call to addListSelectionListener
```

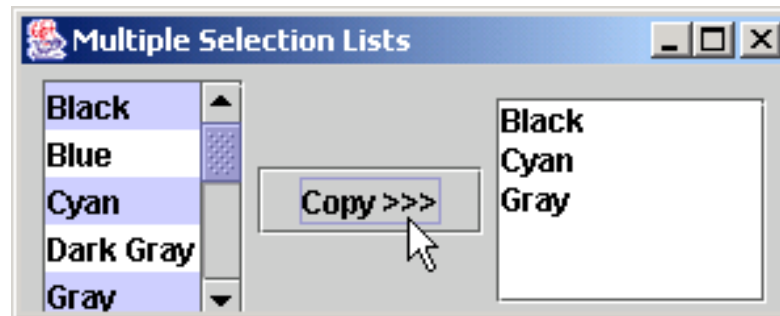
Set appropriate background
depending on user selection

```
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 a list
 - 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            ListSelectionMode.MULTIPLE_INTERVAL_SELECTION );
28        container.add( new JScrollPane( colorList ) );

```

MultipleSelectionTest.java

Lines 10-12 and 24

Lines 26-27

Use colorNames array to populate JList

JList colorList allows multiple selections

29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

```
// create copy button and register its listener
copyButton = new JButton( "Copy >>>" );
copyButton.addActionListener(

    new ActionListener() { // anonymous inner class

        // handle button event
        public void actionPerformed((ActionEvent event) )
        {
            // place selected values in copyList
            copyList.setListData( colorList.getSelectedValues() );
        }

    } // end anonymous inner class

); // end call to addActionListener

container.add( copyButton );

// set up JList copyList
copyList = new JList( );
copyList.setVisibleRowCount( 5 );
copyList.setFixedCellwidth( 100 );
copyList.setFixedCellHeight( 15 );
copyList.setSelectionMode(
    ListSelectionMode.SINGLE_INTERVAL_SELECTION );
container.add( new JScrollPane( copyList ) );
```

MultipleSelecti
onTest.java

Line 40

Lines 54-55

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

JList colorList
allows single selections

57
58
59
60
61
62
63
64
65
66
67
68
69

```
setSize( 300, 130 );  
setVisible( true );
```

MultipleSelectionTest.java

```
} // end constructor MultipleSelectionTest
```

```
public static void main( String args[] )
```

```
{
```

```
MultipleSelectionTest application = new MultipleSelectionTest();  
application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

```
}
```

```
} // end class MultipleSelectionTest
```

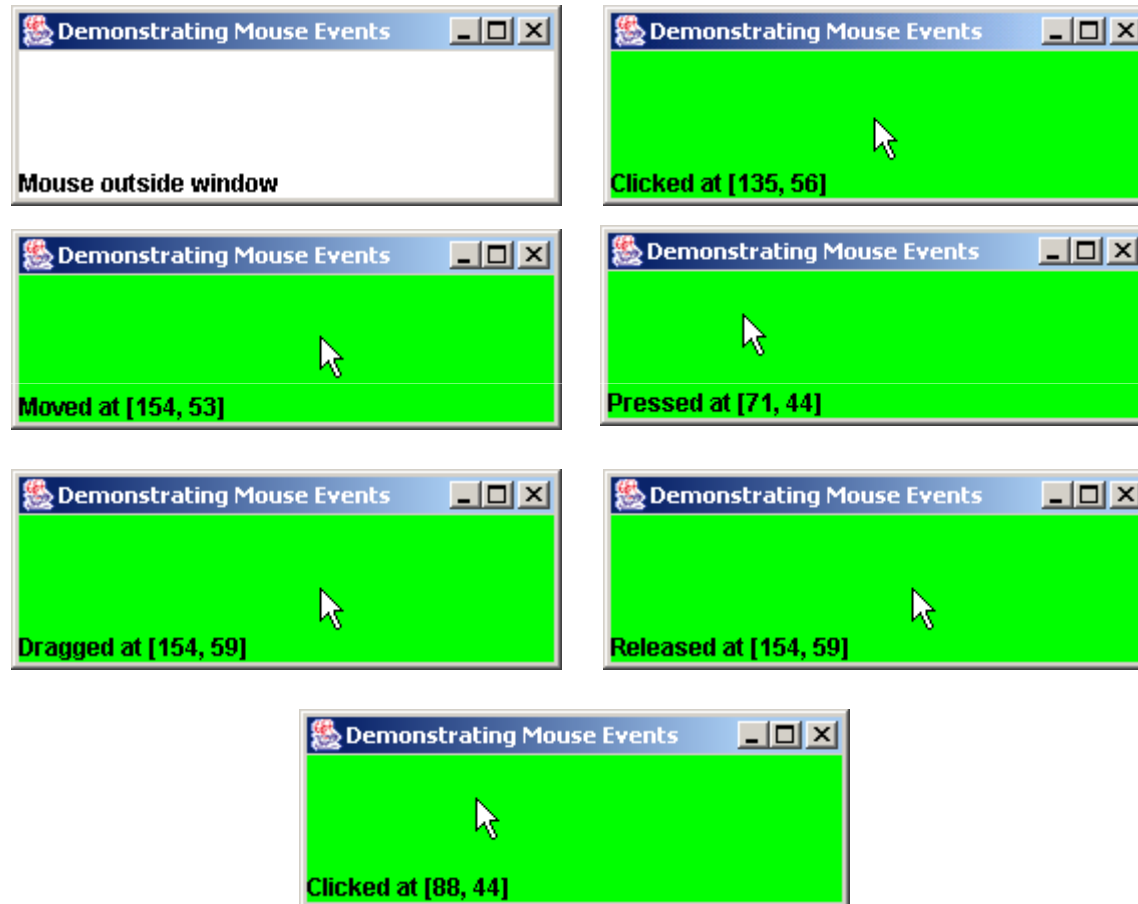
13.12 Mouse Event Handling

- Event-listener interfaces for mouse events
 - `MouseListener`
 - `MouseMotionListener`
 - Listen for `MouseEvent`s

MouseListener and MouseMotionListener interface methods

MouseListener and MouseMotionListener interface methods	
<i>Methods of interface MouseListener</i>	
<code>public void mousePressed(MouseEvent event)</code>	Called when a mouse button is pressed while the mouse cursor is on a component.
<code>public void mouseClicked(MouseEvent event)</code>	Called when a mouse button is pressed and released while the mouse cursor remains stationary on a component.
<code>public void mouseReleased(MouseEvent event)</code>	Called when a mouse button is released after being pressed. This event is always preceded by a mousePressed event.
<code>public void mouseEntered(MouseEvent event)</code>	Called when the mouse cursor enters the bounds of a component.
<code>public void mouseExited(MouseEvent event)</code>	Called when the mouse cursor leaves the bounds of a component.
<i>Methods of interface MouseMotionListener</i>	
<code>public void mouseDragged(MouseEvent event)</code>	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.
<code>public void mouseMoved(MouseEvent event)</code>	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-motion
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

MouseListener.java

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 }
```

MouseListener.java

```
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 }
```

Invoked when mouse
cursor exits JFrame

```
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 }
```

Invoked when user
drags mouse cursor

```
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 }
```

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

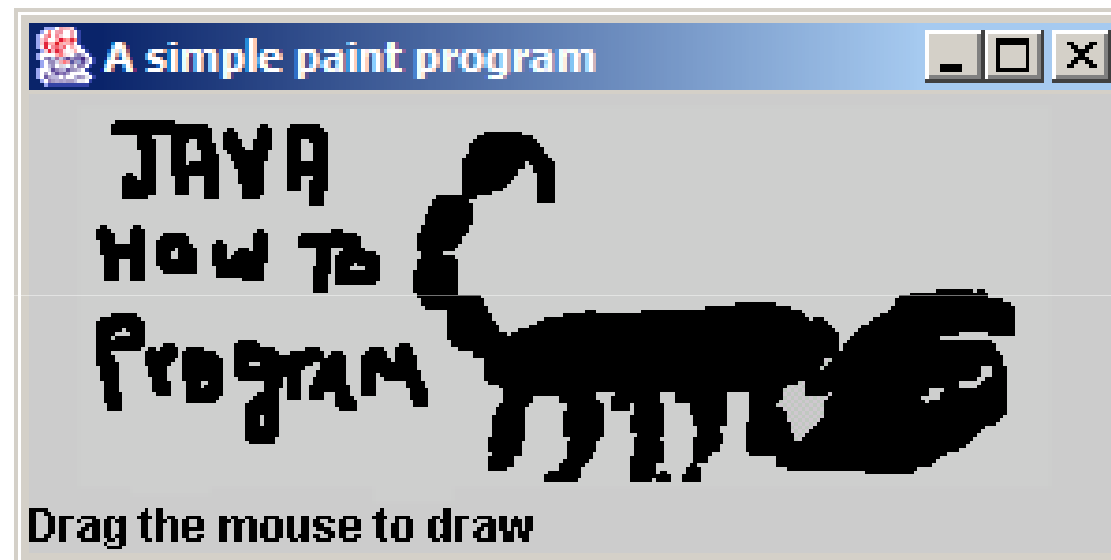
Classes Adaptadoras - Adapter Classes

- Adapter class
 - Implementam uma interface
 - Provê implementação default para cada método de interface method
 - 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
<code>ComponentAdapter</code>	<code>ComponentListener</code>
<code>ContainerAdapter</code>	<code>ContainerListener</code>
<code>FocusAdapter</code>	<code>FocusListener</code>
<code>KeyAdapter</code>	<code>KeyListener</code>
<code>MouseAdapter</code>	<code>MouseListener</code>
<code>MouseMotionAdapter</code>	<code>MouseMotionListener</code>
<code>WindowAdapter</code>	<code>WindowListener</code>

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         addMouseListener(
23             new MouseMotionAdapter() { // anonymous inner class
24
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      }

```

Painter.java

Override method mouseDragged, but not method mouseMoved

Store coordinates where mouse was dragged, then repaint JFrame

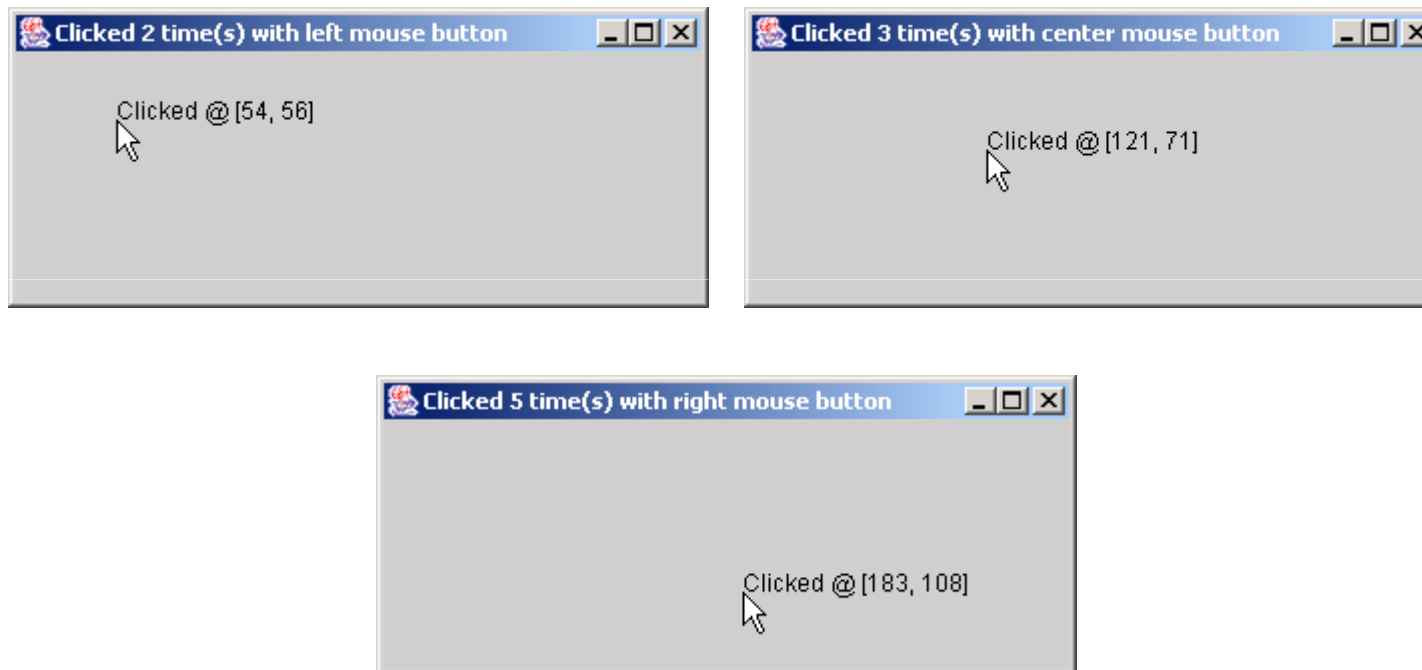
Draw circle of diameter 4 where user dragged cursor

53
54
55
56
57
58
59
60

```
public static void main( String args[] )  
{  
    Painter application = new Painter();  
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );  
}  
} // 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
```

MouseDetails.java

Line 15

Register mouse listener

```

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() + " times",
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";

```

MouseDetails.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

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

InputEvent method	Description
<code>isMetaDown()</code>	Returns <code>true</code> 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 <i>Meta</i> key on the keyboard and click the mouse button.
<code>isAltDown()</code>	Returns <code>true</code> 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 <i>Alt</i> key on the keyboard and click the only- or left-mouse button, respectively.

Tratamento de Eventos de Teclado

- Interface `KeyListener`

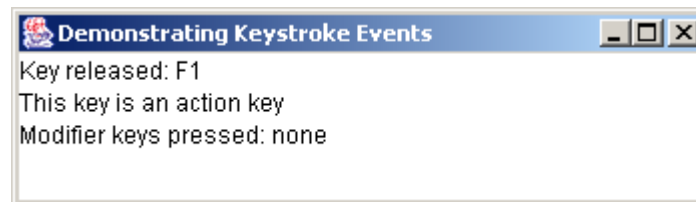
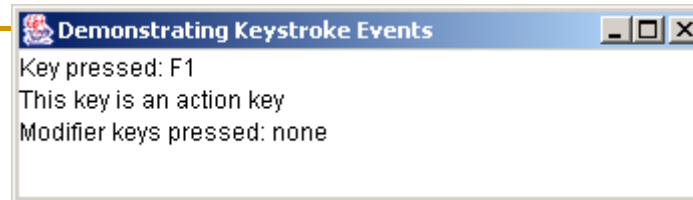
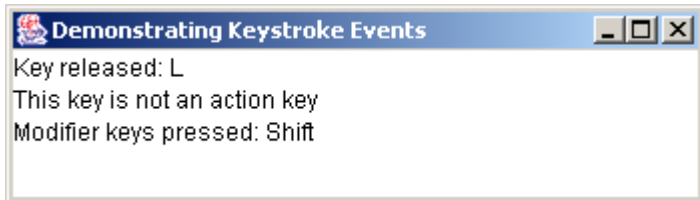
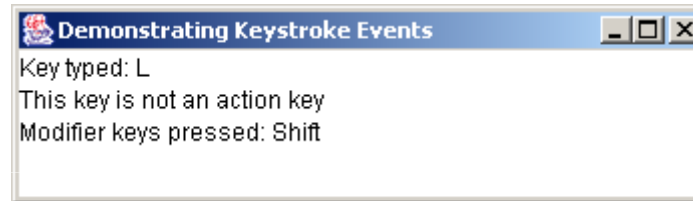
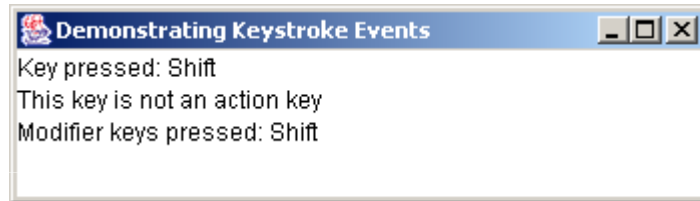
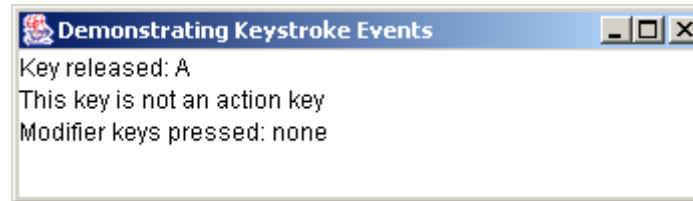
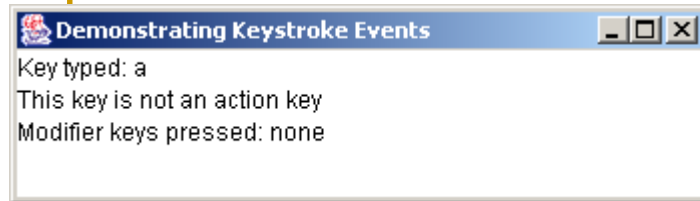
- Handles *key events*

- 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
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

```
} // end KeyDemo constructor
```

```
// handle press of any key
```

```
public void keyPressed( KeyEvent event )
```

```
{
```

```
    line1 = "key pressed: " + event.getKeyText( event.getKeyCode() );
```

```
    setLines2and3( event );
```

```
}
```

```
// handle release of any key
```

```
public void keyReleased( KeyEvent event )
```

```
{
```

```
    line1 = "key released: " + event.getKeyText( event.getKeyCode() );
```

```
    setLines2and3( event );
```

```
}
```

```
// handle press of an action key
```

```
public void keyTyped( KeyEvent event )
```

```
{
```

```
    line1 = "key typed: " + event.getKeyChar();
```

```
    setLines2and3( event );
```

```
}
```

```
// set second and third lines of output
```

```
private void setLines2and3( KeyEvent event )
```

```
{
```

KeyDemo.java

Called when user presses key

Return virtual key code

Called when user releases key

Called when user types key

```

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" + line3 );
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

```

KeyDemo.java

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

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

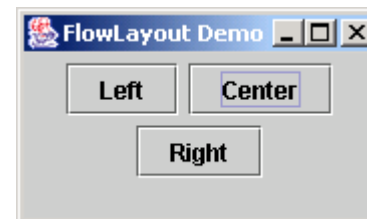
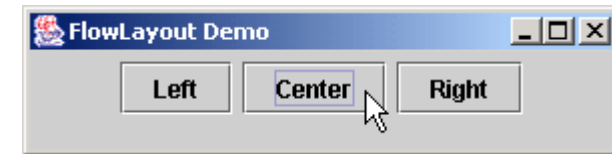
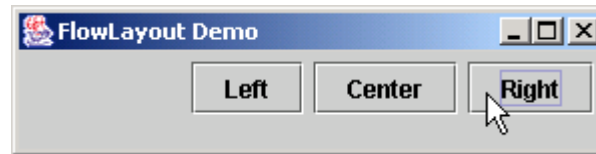
Alguns Exemplos de Layout managers

Layout manager	Description
FlowLayout	Default for <code>java.awt.Applet</code> , <code>java.awt.Panel</code> and <code>javax.swing.JPanel</code> . 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 <code>Container</code> method <code>add</code> , which takes a <code>Component</code> and an integer index position as arguments.
BorderLayout	Default for the content panes of <code>JFrames</code> (and other windows) and <code>JApplets</code> . Arranges the components into five areas: <code>NORTH</code> , <code>SOUTH</code> , <code>EAST</code> , <code>WEST</code> and <code>CENTER</code> .
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

Lines 17 and 21

Set layout as FlowLayout

```

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     } // end anonymous inner class
39
40 ); // end call to addActionListener
41
42 // set up centerButton and register listener
43 centerButton = new JButton( "Center" );
44 container.add( centerButton );
45 centerButton.addActionListener(
46
47     new ActionListener() { // anonymous inner class
48
49         // process centerButton event
50         public void actionPerformed( ActionEvent event )
51         {
52             layout.setAlignment( FlowLayout.CENTER );
53         }
54

```

FlowLayoutDemo.java

When user presses left JButton, left align components

When user presses center JButton, center components

```
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
```

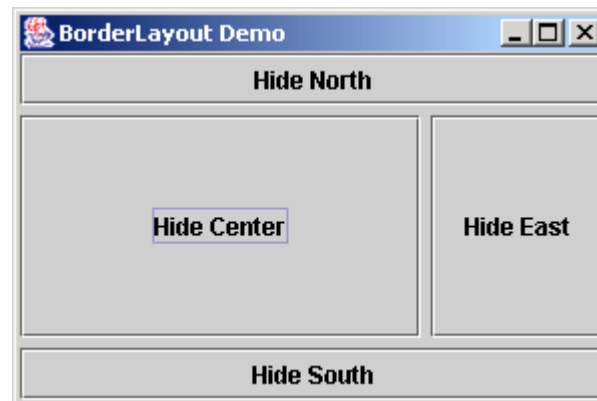
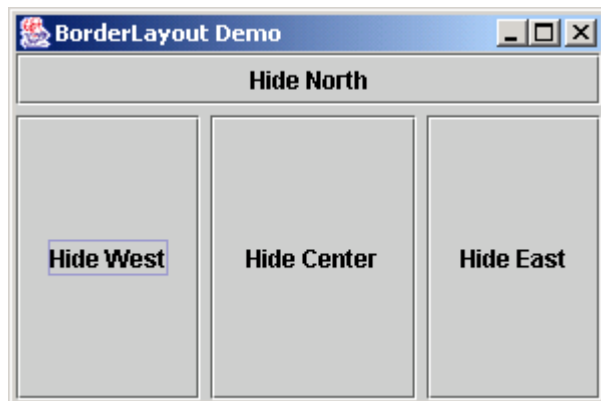
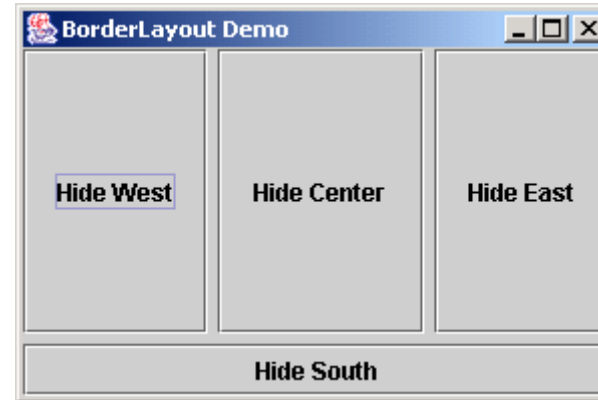
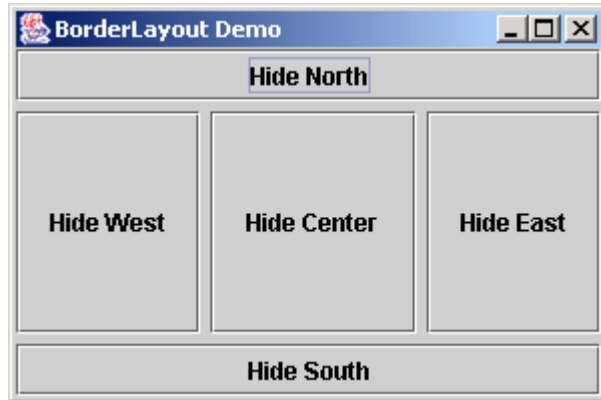
FlowLayoutDemo.java

13.15.2 BorderLayout

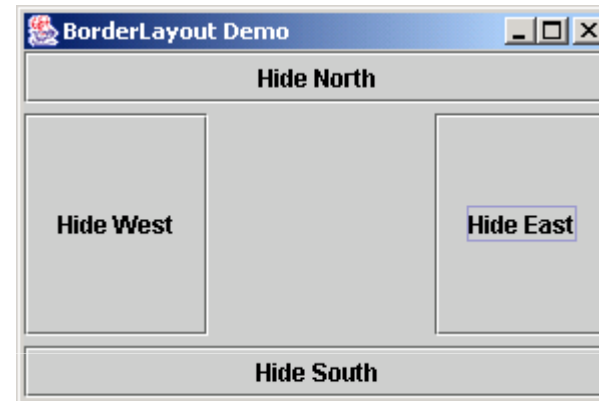
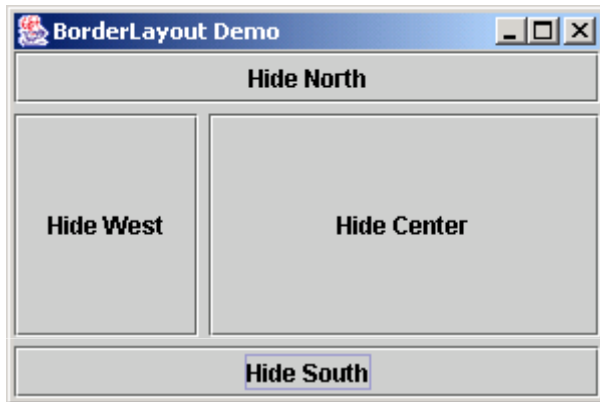
■ BorderLayout

- 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



BorderLayoutDemo.java

```
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
```

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 );
```

Place JButtons in regions specified by BorderLayout

```
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 );
```

When JButtons are "invisible," they are not displayed on screen, and BorderLayout rearranges

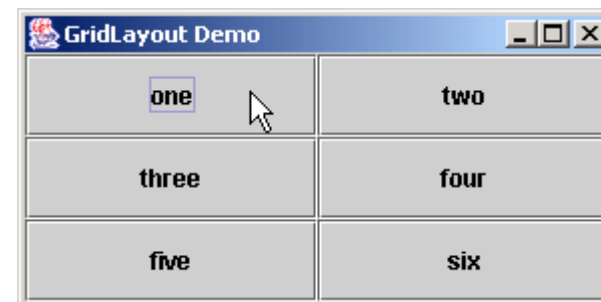
```
53
54     // re-layout the content pane
55     layout.layoutContainer( getContentPane() );
56 }
57
58 public static void main( String args[] )
59 {
60     BorderLayoutDemo application = new BorderLayoutDemo();
61     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
62 }
63
64 } // end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

BorderLayoutDemo.java

GridLayout

- GridLayout

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



GridLayoutDemo.java

```
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 );
```

Create GridLayout grid1
with 2 rows and 3 columns

Create GridLayout grid2
with 3 rows and 2 columns

GridLayoutDemo.java

```
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 }
```

Toggle current
GridLayout when
user presses JButton

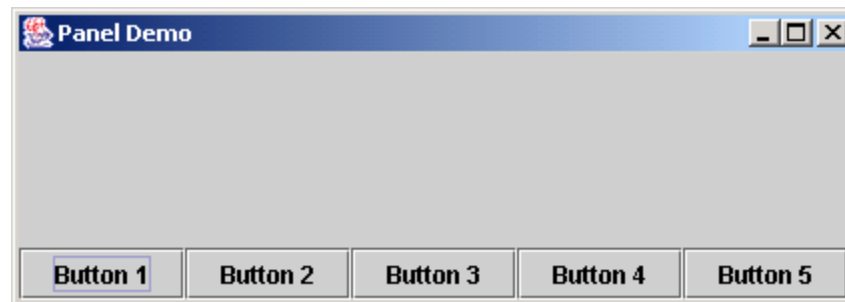
53
54
55
56
57
58
59
60

```
public static void main( String args[] )  
{  
    GridLayoutDemo application = new GridLayoutDemo();  
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );  
}  
} // 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 JButtons

```

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 JButton to JPanel
ava

Line 29

Add JPanel to SOUTH region of Container

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

