CNT 4714: Enterprise Computing
Spring 2010

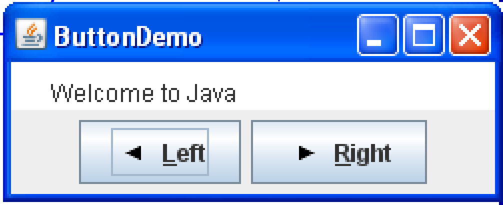
Introduction To GUIs and Event-Driven Programming In Java – Part 3

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cnt4714/spr2010

School of Electrical Engineering and Computer Science
University of Central Florida

```
public ButtonDemo() {
    // Set the background color of messagePanel
    messagePanel.setBackground(Color.white);

    // Create Panel jpButtons to hold two Buttons "<=" and "right =>"
    JPanel jpButtons = new JPanel();
    jpButtons.setLayout(new FlowLayout());
    jpButtons.add(jbtLeft);
    jpButtons.add(jbtRight);

    // Set keyboard mnemonics
    jbtLeft.setMnemonic('L');
    jbtRight.setMnemonic('R');

    // Set icons and remove text
//    jbtLeft.setIcon(new ImageIcon("image/left.gif"));
//    jbtRight.setIcon(new ImageIcon("image/right.gif"));
//    jbtLeft.setText(null);
//    jbtRight.setText(null);
//    jbtRight.setHorizontalTextPosition(SwingConstants.RIGHT);


    // Set tool tip text on the buttons
    jbtLeft.setToolTipText("Move message to left");
    jbtRight.setToolTipText("Move message to right");
```

Uncomment these lines to set an icon image on the button.

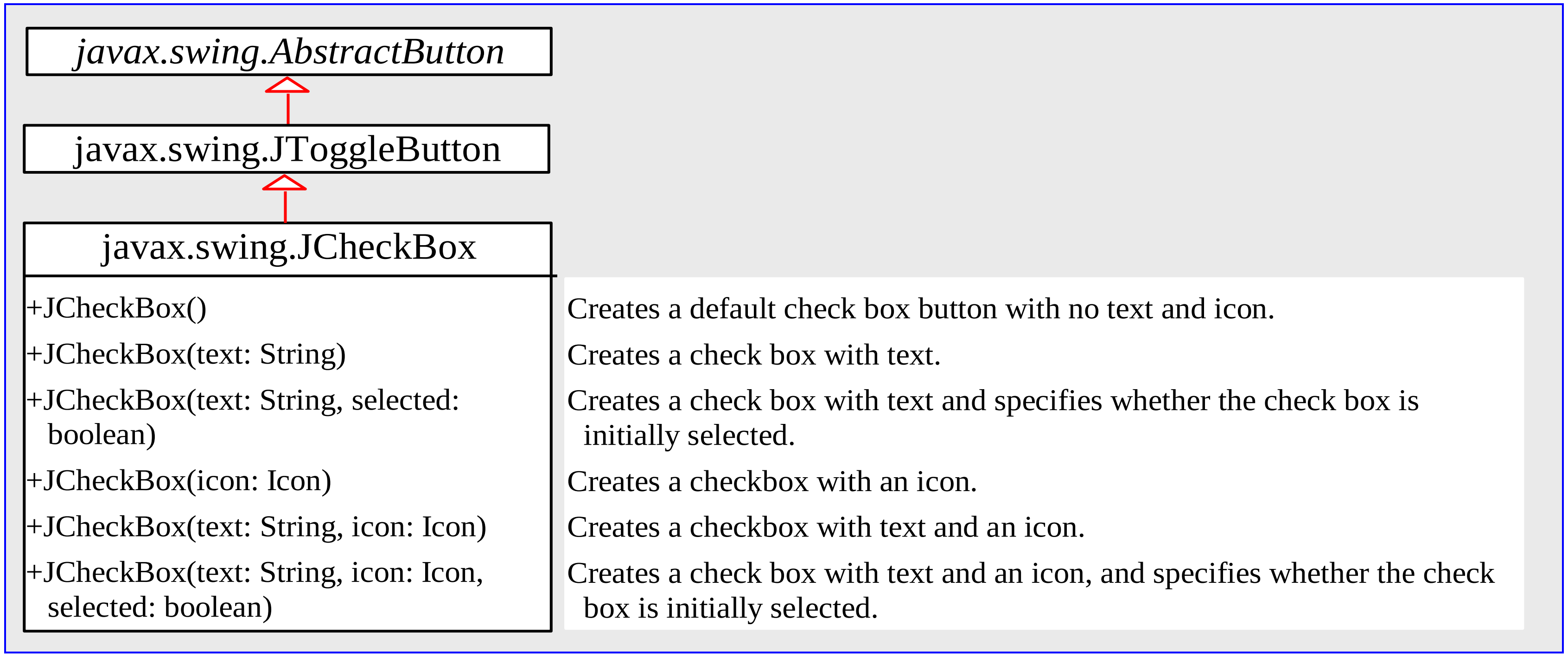


Check Boxes

- A toggle button is a two-state button (like a typical light switch – its either on or off).
- `JToggleButton` inherits `AbstractButton` and implements a toggle button.
- Often one of `JToggleButton`'s subclasses `JCheckBox` and `JRadioButton` are used to enable the user to toggle a choice on or off.
- We'll look at the `JCheckBox` class first.

JCheckBox

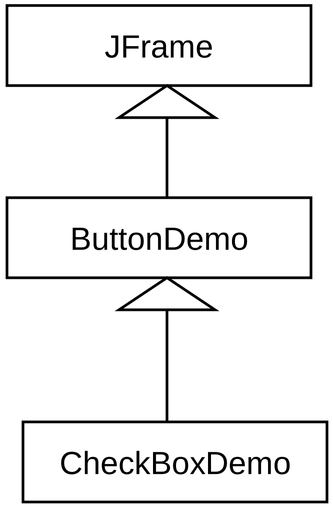
- `JCheckBox` inherits all the properties from AbstractButton, such as `text`, `icon`, `mnemonic`, `verticalAlignment`, `horizontalAlignment`, `horizontalTextPosition`, `verticalTextPosition`, and `selected`, and provides several constructors to create check boxes, as shown below:



+JCheckBox()	Creates a default check box button with no text and icon.
+JCheckBox(text: String)	Creates a check box with text.
+JCheckBox(text: String, selected: boolean)	Creates a check box with text and specifies whether the check box is initially selected.
+JCheckBox(icon: Icon)	Creates a checkbox with an icon.
+JCheckBox(text: String, icon: Icon)	Creates a checkbox with text and an icon.
+JCheckBox(text: String, icon: Icon, selected: boolean)	Creates a check box with text and an icon, and specifies whether the check box is initially selected.

Example – Checkboxes

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CheckBoxDemo extends ButtonDemo {
  // Create three check boxes to control the display of
message
  private JCheckBox jchkCentered = new JCheckBox("Centered");
  private JCheckBox jchkBold = new JCheckBox("Bold");
  private JCheckBox jchkItalic = new JCheckBox("Italic");

  public static void main(String[] args) {
    CheckBoxDemo frame = new CheckBoxDemo();
    frame.setTitle("CheckBoxDemo");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(500, 200);
    frame.setVisible(true);
  }

  public CheckBoxDemo() {
    // Set mnemonic keys
    jchkCentered.setMnemonic('C');
    jchkBold.setMnemonic('B');
    jchkItalic.setMnemonic('I');
```

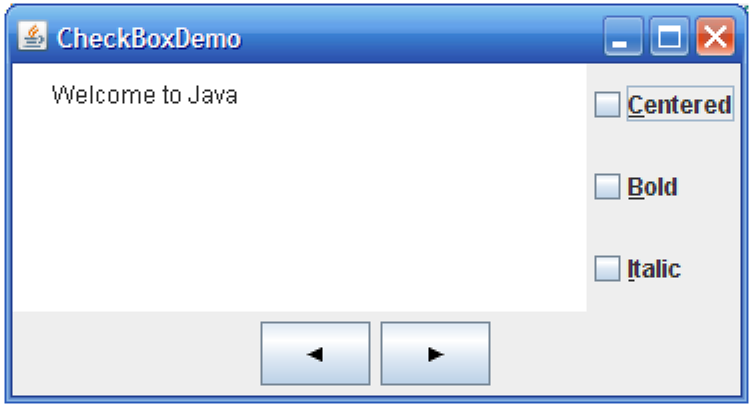


Example – Checkboxes

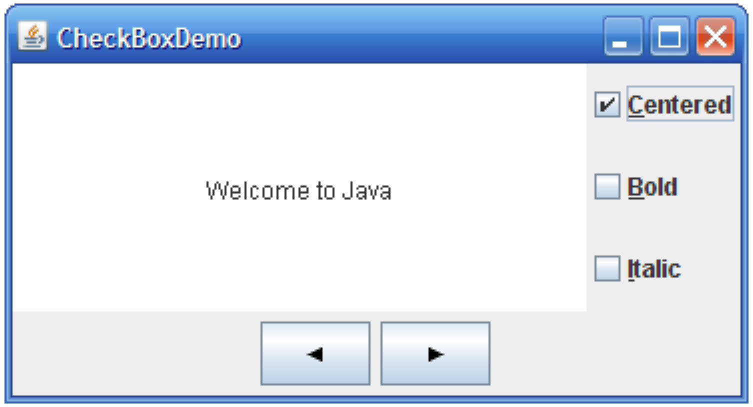
```
// Create a new panel to hold check boxes
    JPanel jpCheckBoxes = new JPanel();
    jpCheckBoxes.setLayout(new GridLayout(3, 1));
    jpCheckBoxes.add(jchkCentered);
    jpCheckBoxes.add(jchkBold);
    jpCheckBoxes.add(jchkItalic);
    add(jpCheckBoxes, BorderLayout.EAST);

    // Register listeners with the check boxes
    jchkCentered.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         messagePanel.setCentered(jchkCentered.isSelected());
      }
    });
    jchkBold.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        setNewFont();
      }
    });
    jchkItalic.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        setNewFont();
      }
    });
  }
```

Example – Checkboxes

```
private void setNewFont() {
    // Determine a font style
    int fontStyle = Font.PLAIN;
    fontStyle += (jchkBold.isSelected() ? Font.BOLD : Font.PLAIN);
    fontStyle += (jchkItalic.isSelected() ? Font.ITALIC :
Font.PLAIN);

    // Set font for the message
    Font font = messagePanel.getFont();
    messagePanel.setFont(
      new Font(font.getName(), fontStyle, font.getSize()));
  }
}
```



Initial GUI



After clicking Centered checkbox

Comments on `CheckBoxDemo`

- The `CheckBoxDemo` class extends `ButtonDemo` and adds three check boxes to control how the message is displayed.
- When a `CheckBoxDemo` is constructed, its superclass's no-arg constructor is invoked, so we did not need to rewrite the code that is already in the constructor of `ButtonDemo`.
- When a check box is checked or unchecked, the listener's `actionPerformed` method is invoked to process the event. When the *Centered* check box is checked or unchecked, the `centered` property of the `MessagePanel` class is set to `true` or `false`.
- The current font name and size used in the `MessagePanel` are obtained from the `MessagePanel.getFont()` using the `getName()` and `getSize()` methods. The font styles are specified in the check boxes. If no font style is selected, the font style is `Font.PLAIN`.

Comments on `CheckBoxDemo`

- The `setFont` method defined in the `Component` class is inherited in the `MessagePanel` class. This method automatically invokes the `repaint` method. Invoking `setFont` in `MessagePanel` automatically repaints the message.
- A check box fires an `ActionEvent` and an `ItemEvent` when it is clicked. You could process either the `ActionEvent` or the `ItemEvent` to redisplay the message. The previous version of the program processes the `ActionEvent`. The following version of the same program processes the `ItemEvent`.
- Run both versions of the check box demo program to convince yourself that both behave the same way even though a different type of event is being handled in each version.

Example – `CheckBoxUsingItemEvent`

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CheckBoxDemoUsingItemEvent extends ButtonDemo
{
  // Create three check boxes to control the display of
message
  private JCheckBox jchkCentered = new
JCheckBox("Centered");
  private JCheckBox jchkBold = new JCheckBox("Bold");
  private JCheckBox jchkItalic = new JCheckBox("Italic");

  public static void main(String[] args) {
    CheckBoxDemo frame = new CheckBoxDemo();
    frame.setTitle("CheckBoxDemo");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(500, 200);
    frame.setVisible(true);
  }

  public CheckBoxDemoUsingItemEvent() {
    // Set mnemonic keys
    jchkCentered.setMnemonic('C');
    jchkBold.setMnemonic('B');
    jchkItalic.setMnemonic('I');
```

Example – CheckBoxUsingItemEvent

```
// Create a new panel to hold check boxes
    JPanel jpCheckBoxes = new JPanel();
    jpCheckBoxes.setLayout(new GridLayout(3, 1));
    jpCheckBoxes.add(jchkCentered);
    jpCheckBoxes.add(jchkBold);
    jpCheckBoxes.add(jchkItalic);
    add(jpCheckBoxes, BorderLayout.EAST);

    // Register listeners with the check boxes
    jchkCentered.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
         messagePanel.setCentered(jchkCentered.isSelected());
      }
    });
    jchkBold.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
        setNewFont();
      }
    });
    jchkItalic.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
        setNewFont();
      }
    });
  }
```

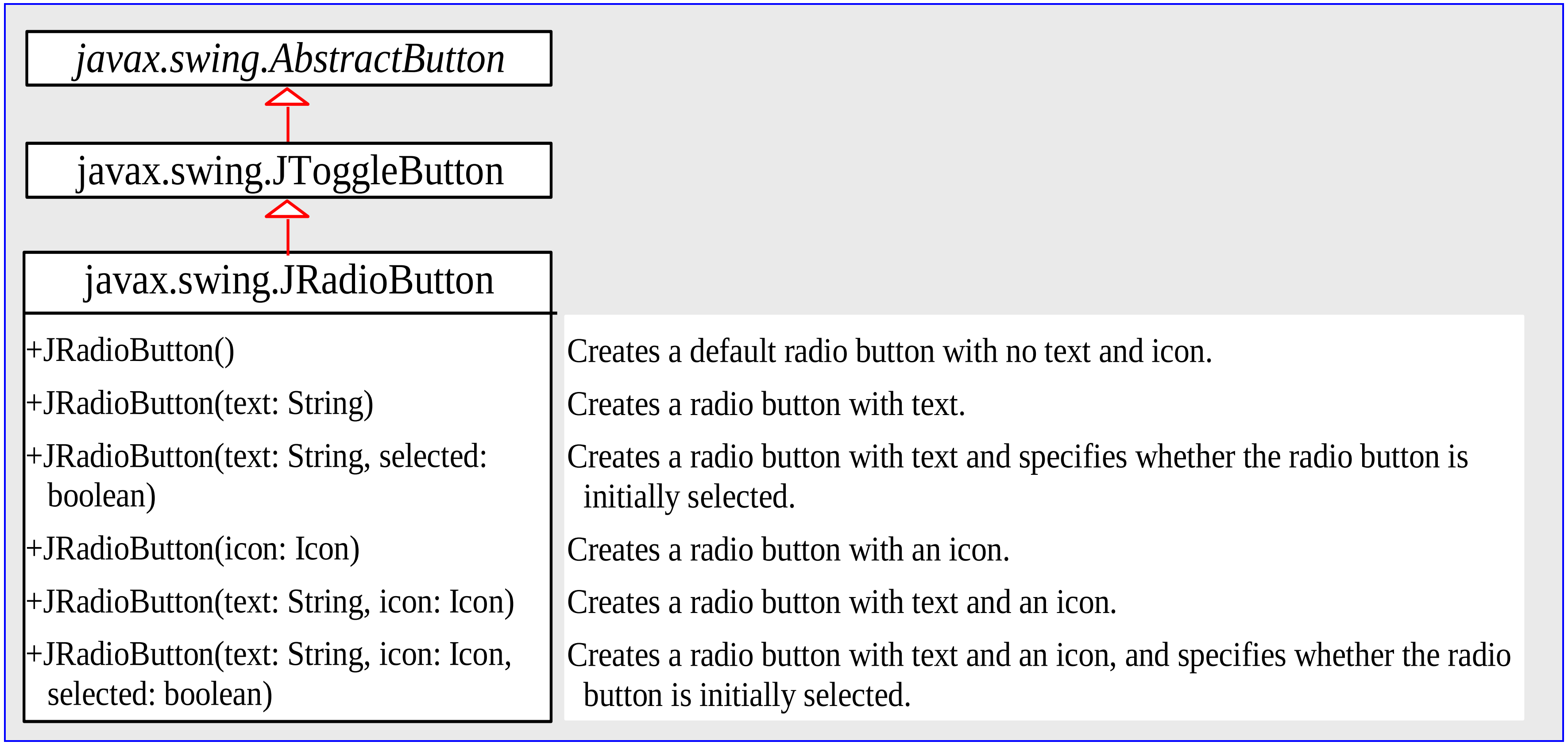
Example – `CheckBoxUsingItemEvent`

```
  private void setNewFont() {
    // Determine a font style
    int fontStyle = Font.PLAIN;
    fontStyle += (jchkBold.isSelected() ? Font.BOLD :
Font.PLAIN);
    fontStyle += (jchkItalic.isSelected() ? Font.ITALIC :
Font.PLAIN);

    // Set font for the message
    Font font = messagePanel.getFont();
    messagePanel.setFont(
      new Font(font.getName(), fontStyle, font.getSize()));
  }
}
```

Radio Buttons

- Radio buttons, also known as option buttons, enable you to choose a single item from a group of choices.
- In appearance radio buttons resemble check boxses, but check boxes display a square that is either checked or unchecked, whereas radio buttons display a circle that is either filled (if selected) or blank (if not selected).
- `JRadioButton` inherits `AbstractButton` and provides several overloaded constructors to create radio buttons. The constructors are similar in nature to those for `JCheckBox`.
- The UML (again a partial UML) for the `JRadioButton` class is shown on the next page.

Radio Buttons



+JRadioButton()	Creates a default radio button with no text and icon.
+JRadioButton(text: String)	Creates a radio button with text.
+JRadioButton(text: String, selected: boolean)	Creates a radio button with text and specifies whether the radio button is initially selected.
+JRadioButton(icon: Icon)	Creates a radio button with an icon.
+JRadioButton(text: String, icon: Icon)	Creates a radio button with text and an icon.
+JRadioButton(text: String, icon: Icon, selected: boolean)	Creates a radio button with text and an icon, and specifies whether the radio button is initially selected.

Example – RadioButtonDemo

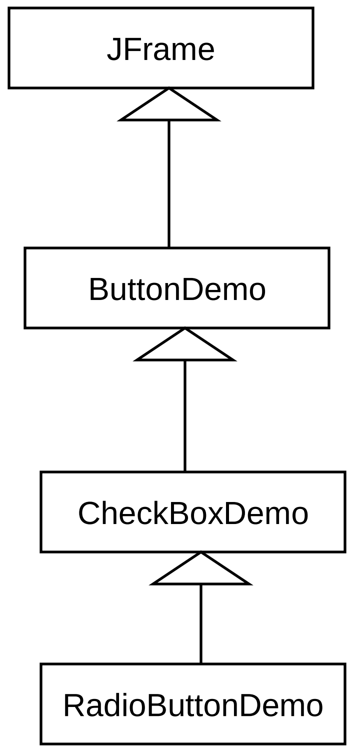
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class RadioButtonDemo extends CheckBoxDemo {
  // Declare radio buttons
  private JRadioButton jrbRed, jrbGreen, jrbBlue;
  public static void main(String[] args) {
    RadioButtonDemo frame = new RadioButtonDemo();
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setTitle("RadioButtonDemo");
    frame.setSize(500, 200);
    frame.setVisible(true);
  }
  public RadioButtonDemo() {
    // Create a new panel to hold check boxes
    JPanel jpRadioButtons = new JPanel();
    jpRadioButtons.setLayout(new GridLayout(3, 1));
    jpRadioButtons.add(jrbRed = new JRadioButton("Red"));
    jpRadioButtons.add(jrbGreen = new JRadioButton("Green"));
    jpRadioButtons.add(jrbBlue = new JRadioButton("Blue"));
    add(jpRadioButtons, BorderLayout.WEST);
    // Create a radio button group to group three buttons
    ButtonGroup group = new ButtonGroup();
    group.add(jrbRed);
    group.add(jrbGreen);
    group.add(jrbBlue);
```

Our approach here is to extend the CheckBoxDemo class by adding the radio button features. We could have also add the code directly to the CheckBoxDemo, This approach is somewhat classier, since the CheckBoxDemo class can always be reused to implement just check boxes.

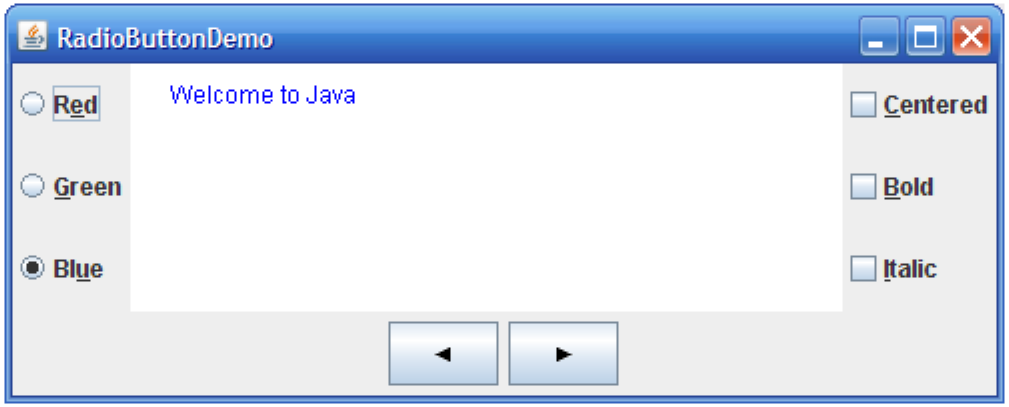
See Note on page 18

Example – RadioButtonDemo

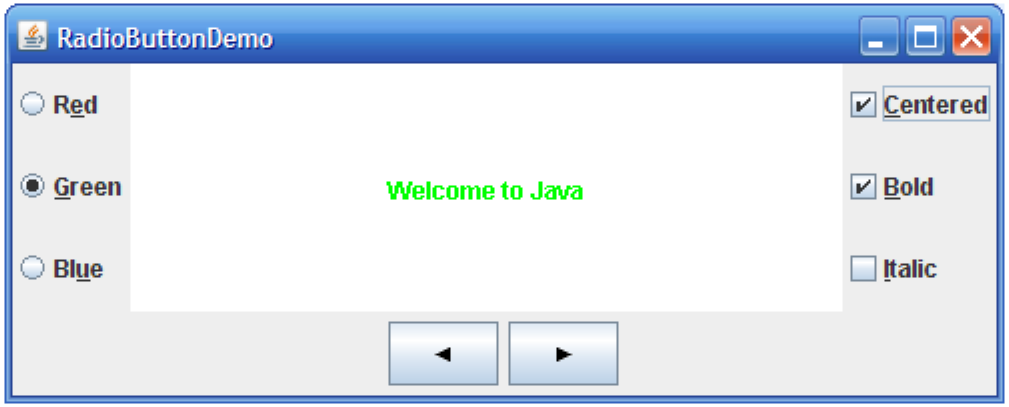
```
// Set keyboard mnemonics
    jrbRed.setMnemonic('E');
    jrbGreen.setMnemonic('G');
    jrbBlue.setMnemonic('U');

    // Register listeners for check boxes
    jrbRed.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        messagePanel.setForeground(Color.red);
      }
    });
    jrbGreen.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        messagePanel.setForeground(Color.green);
      }
    });
    jrbBlue.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        messagePanel.setForeground(Color.blue);
      }
    });

    // Set initial message color to blue
    jrbBlue.setSelected(true);
    messagePanel.setForeground(Color.blue);
  }
}
```



Example – `RadioButtonDemo`



Initial GUI



After Radio and Check Buttons set

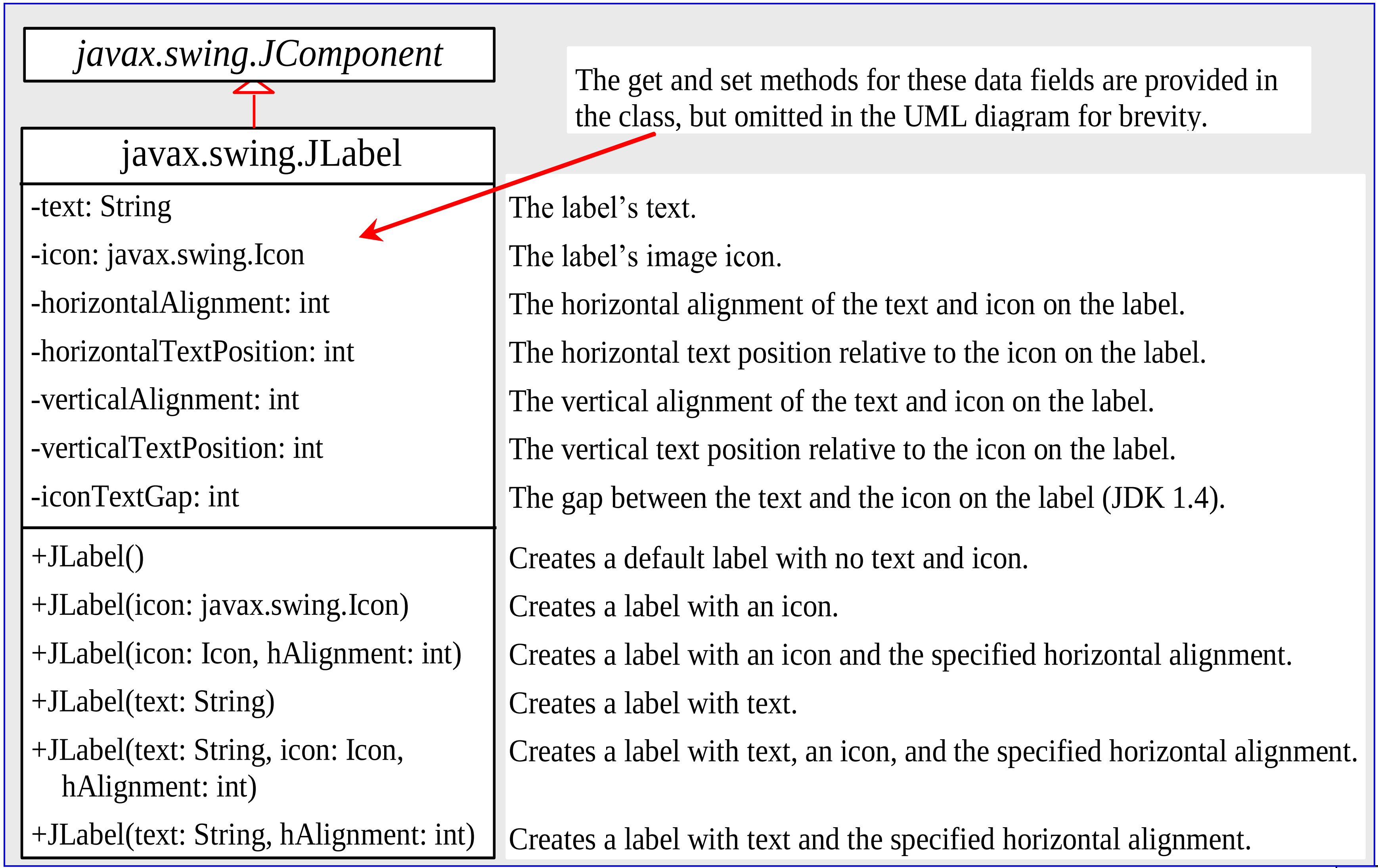
Grouping Radio Buttons

- To group radio buttons, you need to create an instance of `java.swing.ButtonGroup` and use the add method to add them to it as shown in the code on page 15.
- Without putting radio buttons into a group, the buttons can be selected independently of one another. The act of placing the buttons into the group is what makes the buttons within that group mutually exclusive. To see this yourself, remove the statements from the program that create the button group and add the buttons to it and then re-run the program and you will be able to select all three radio buttons simultaneously.
- When a radio button is changed (selected or deselected), it fires an `ItemEvent` and then an `ActionEvent`.
- To see if a radio button is selected, use the `isSelected()` method.

Labels

- A label is a display area for a short text message, an image, or both. It is often used to label other components (usually text fields, to indicate what the user is to enter in the field).
- `JLabel` inherits all the properties of the `JComponent` class and contains many properties similar to the ones in the `JButton` class.
- The UML for the `JLabel` class is shown on the next page.

JLabel



javax.swing.JComponent

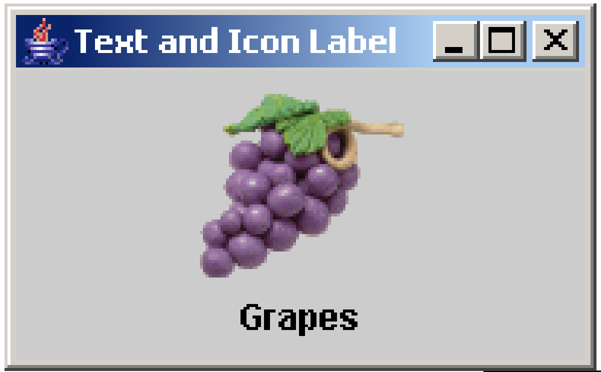
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

javax.swing.JLabel

-text: String	The label's text.
-icon: javax.swing.Icon	The label's image icon.
-horizontalAlignment: int	The horizontal alignment of the text and icon on the label.
-horizontalTextPosition: int	The horizontal text position relative to the icon on the label.
-verticalAlignment: int	The vertical alignment of the text and icon on the label.
-verticalTextPosition: int	The vertical text position relative to the icon on the label.
-iconTextGap: int	The gap between the text and the icon on the label (JDK 1.4).
+JLabel()	Creates a default label with no text and icon.
+JLabel(icon: javax.swing.Icon)	Creates a label with an icon.
+JLabel(icon: Icon, hAlignment: int)	Creates a label with an icon and the specified horizontal alignment.
+JLabel(text: String)	Creates a label with text.
+JLabel(text: String, icon: Icon, hAlignment: int)	Creates a label with text, an icon, and the specified horizontal alignment.
+JLabel(text: String, hAlignment: int)	Creates a label with text and the specified horizontal alignment.

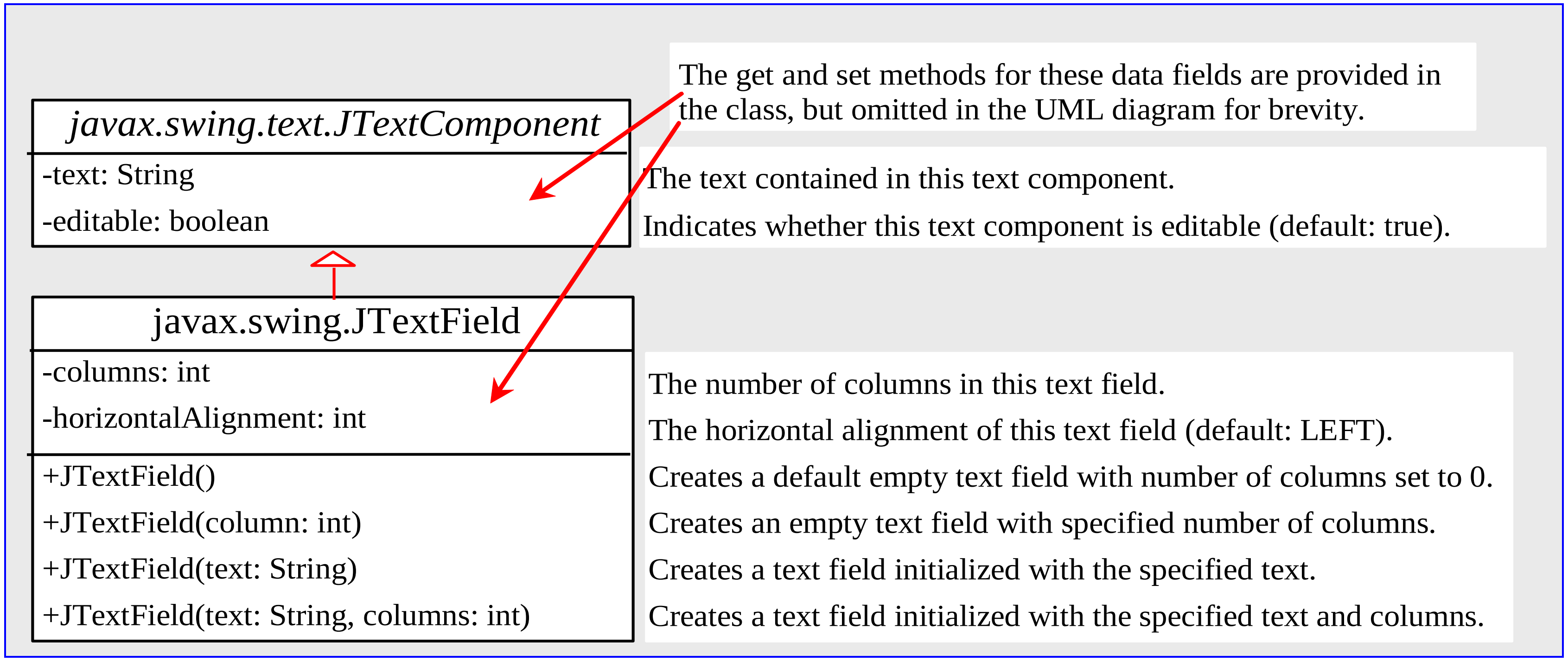
JLabel

```
// Create an image icon from image file
ImageIcon icon = new ImageIcon("E:/image/grapes.gif");

// Create a label with text, an icon,
// with centered horizontal alignment
JLabel jlbl = new JLabel("Grapes", icon,
SwingConstants.CENTER);

// Set label's text alignment and gap between text and
icon
jlbl.setHorizontalTextPosition(SwingConstants.CENTER);
jlbl.setVerticalTextPosition(SwingConstants.BOTTOM);
jlbl.setIconTextGap(5);
```



Text Fields

- A text field can be used to enter or display a string. `JTextField` is a subclass of `JTextComponent`.
- The UML for the `JTextField` class is shown below.


javax.swing.text.JTextComponent
-text: String
-editable: boolean
javax.swing.JTextField
-columns: int
-horizontalAlignment: int
+JTextField()
+JTextField(column: int)
+JTextField(text: String)
+JTextField(text: String, columns: int)

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The text contained in this text component.

Indicates whether this text component is editable (default: true).

The number of columns in this text field.

The horizontal alignment of this text field (default: LEFT).

Creates a default empty text field with number of columns set to 0.

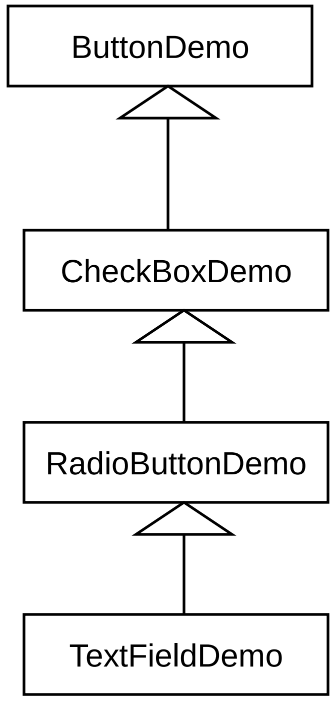
Creates an empty text field with specified number of columns.

Creates a text field initialized with the specified text.

Creates a text field initialized with the specified text and columns.

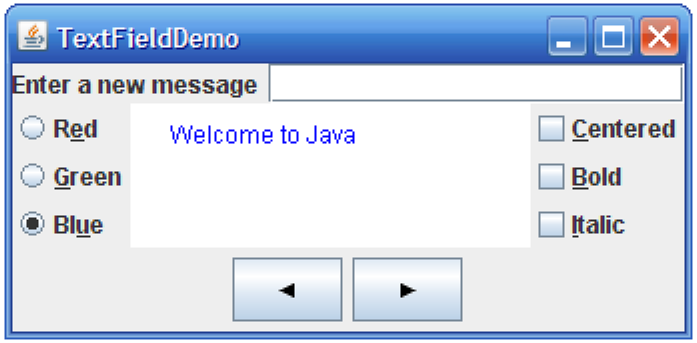
Example – TextFieldDemo

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TextFieldDemo extends RadioButtonDemo {
  private JTextField jtfMessage = new JTextField(10);

  /** Main method */
  public static void main(String[] args) {
    TextFieldDemo frame = new TextFieldDemo();
    frame.pack();
    frame.setTitle("TextFieldDemo");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }

  public TextFieldDemo() {
    // Create a new panel to hold label and text field
    JPanel jpTextField = new JPanel();
    jpTextField.setLayout(new BorderLayout(5, 0));
    jpTextField.add(
      new JLabel("Enter a new message"), BorderLayout.WEST);
    jpTextField.add(jtfMessage, BorderLayout.CENTER);
    add(jpTextField, BorderLayout.NORTH);

    jtfMessage.setHorizontalAlignment(JTextField.RIGHT);
```

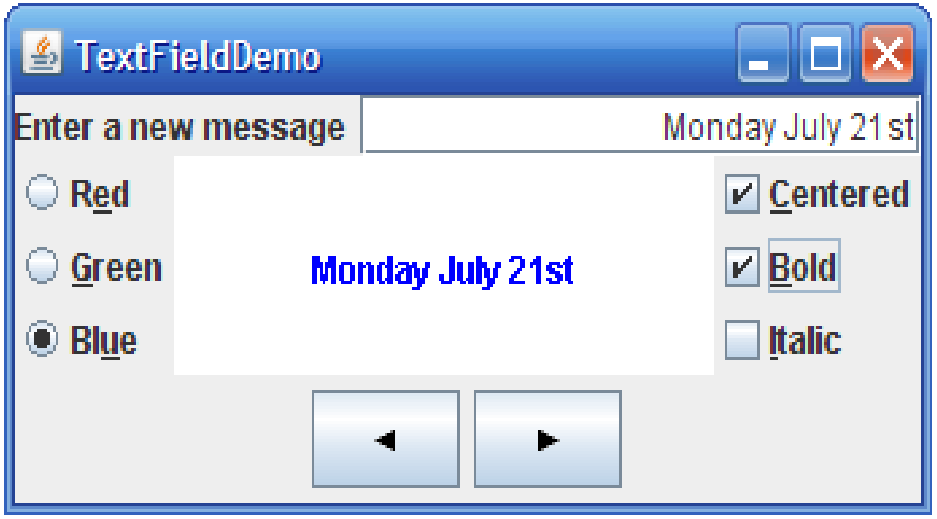


Example – `TextFieldDemo`

```
// Register listener
    jtfMessage.addActionListener(new ActionListener() {
      /** Handle ActionEvent */
      public void actionPerformed(ActionEvent e) {
        messagePanel.setMessage(jtfMessage.getText());
        jtfMessage.requestFocusInWindow();
      }
    });
  }
}
```



Initial GUI



GUI after user input

Comments on Text Fields

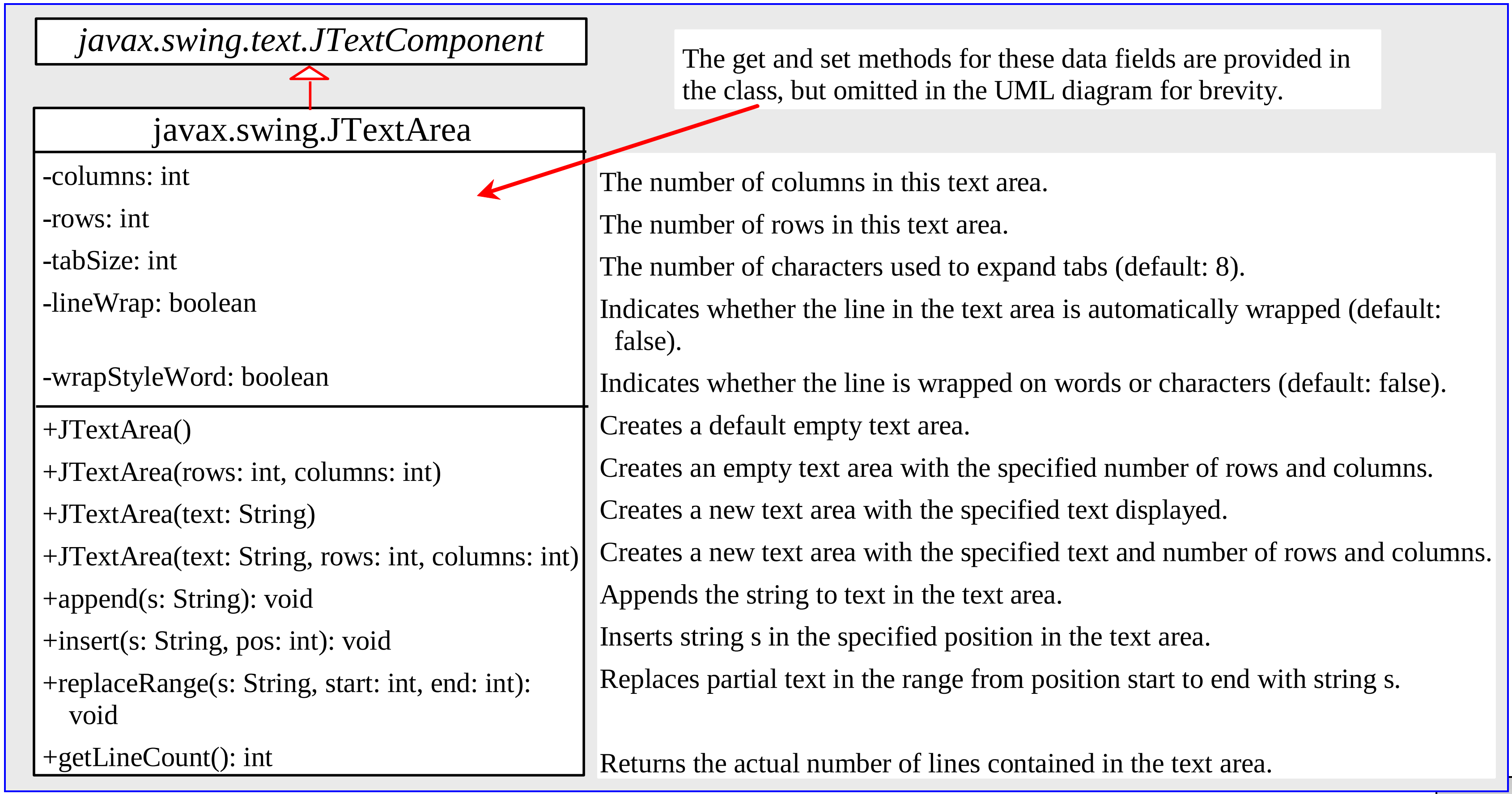
- When you move the cursor into a text field and press the Enter key, it fires an `ActionEvent`.
- In this example program, the `actionPerformed` method sets the new message into `messagePanel`.
- The `pack()` method automatically sizes the frame according to the size of the components placed in it.
- The `requestFocusInWindow()` method is defined in the `Component` class and requests the component to receive input focus. Thus, `jtfMessage.requestFocusInWindow()` requests the input focus on `jtfMessage`. You will see that the cursor is placed on the `jtfMessage` object after the `actionPerformed` method is invoked.

Comments on Text Fields

- If a text field is used for entering a password, use `JPasswordField` to replace `JTextField`. `JPasswordField` extends `JTextField` and hides the input with echo characters (e.g., ****). By default, the echo character is *. You can specify a new echo character using the `setEchoChar(char)` method.

Text Areas

- If you would like to allow the user to enter multiple lines of text, you would need to create several instances of `JTextField.` A better alternative is to use JTextArea, which enables the user to enter multiple lines of text.
- The UML for the J`TextArea` class is shown below.

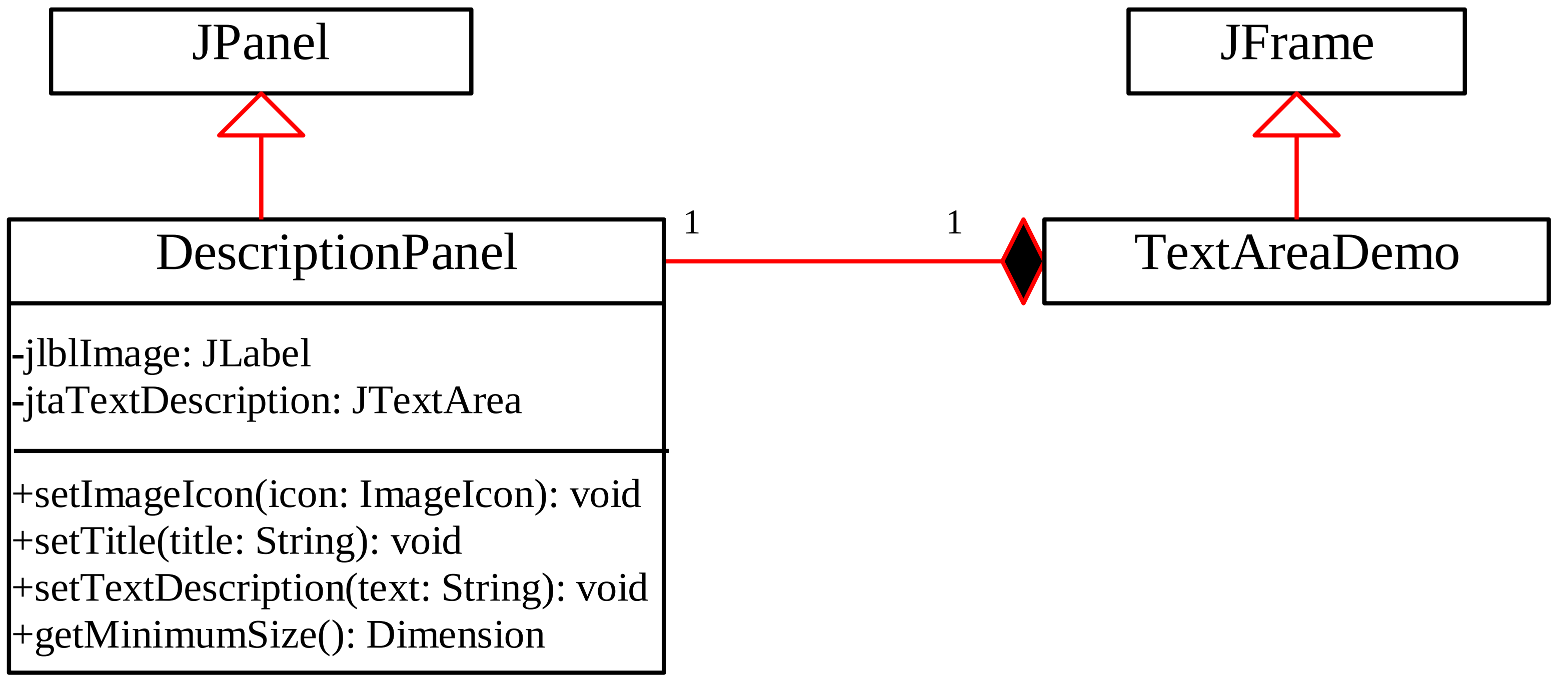


javax.swing.text.JTextComponent

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

javax.swing.JTextArea	
-columns: int	The number of columns in this text area.
-rows: int	The number of rows in this text area.
-tabSize: int	The number of characters used to expand tabs (default: 8).
-lineWrap: boolean	Indicates whether the line in the text area is automatically wrapped (default: false).
-wrapStyleWord: boolean	Indicates whether the line is wrapped on words or characters (default: false).
+JTextArea()	Creates a default empty text area.
+JTextArea(rows: int, columns: int)	Creates an empty text area with the specified number of rows and columns.
+JTextArea(text: String)	Creates a new text area with the specified text displayed.
+JTextArea(text: String, rows: int, columns: int)	Creates a new text area with the specified text and number of rows and columns.
+append(s: String): void	Appends the string to text in the text area.
+insert(s: String, pos: int): void	Inserts string s in the specified position in the text area.
+replaceRange(s: String, start: int, end: int): void	Replaces partial text in the range from position start to end with string s.
+getLineCount(): int	Returns the actual number of lines contained in the text area.

Example – `DescriptionPanel` Class


JPanel
JFrame
DescriptionPanel
-jlblImage: JLabel
-jtaTextDescription: JTextArea
+setImageIcon(icon: ImageIcon): void
+setTitle(title: String): void
+setTextDescription(text: String): void
+getMinimumSize(): Dimension
1
1
TextAreaDemo

Example – DescriptionPanel Class

```
import javax.swing.*;
import java.awt.*;

public class DescriptionPanel extends JPanel {
  /** Label for displaying an image icon and a text */
  private JLabel jlblImageTitle = new JLabel();

  /** Text area for displaying text */
  private JTextArea jtaDescription = new JTextArea();

  public DescriptionPanel() {
    // Center the icon and text and place the text under the icon
    jlblImageTitle.setHorizontalAlignment(JLabel.CENTER);
    jlblImageTitle.setHorizontalTextPosition(JLabel.CENTER);
    jlblImageTitle.setVerticalTextPosition(JLabel.BOTTOM);

    // Set the font in the label and the text field
    jlblImageTitle.setFont(new Font("SansSerif", Font.BOLD, 16));
    jtaDescription.setFont(new Font("Serif", Font.PLAIN, 14));

    // Set lineWrap and wrapStyleWord true for the text area
    jtaDescription.setLineWrap(true);
    jtaDescription.setWrapStyleWord(true);
    jtaDescription.setEditable(false);

    // Create a scroll pane to hold the text area
    JScrollPane scrollPane = new JScrollPane(jtaDescription);
```

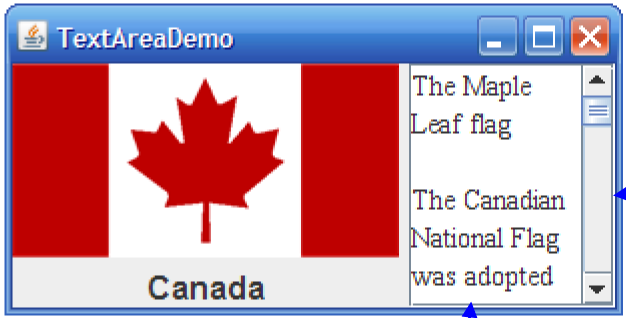
Example – `DescriptionPanel` Class

```
// Set BorderLayout for the panel, add label and scrollpane
    setLayout(new BorderLayout(5, 5));
    add(scrollPane, BorderLayout.CENTER);
    add(jlblImageTitle, BorderLayout.WEST);
  }

  /** Set the title */
  public void setTitle(String title) {
    jlblImageTitle.setText(title);
  }

  /** Set the image icon */
  public void setImageIcon(ImageIcon icon) {
    jlblImageTitle.setIcon(icon);
  }

  /** Set the text description */
  public void setDescription(String text) {
    jtaDescription.setText(text);
  }
}
```

Example – TextAreaDemo

```java
import java.awt.*;
import javax.swing.*;

public class TextAreaDemo extends JFrame {
  // Declare and create a description panel
  private DescriptionPanel descriptionPanel = new DescriptionPanel();
  public static void main(String[] args) {
    TextAreaDemo frame = new TextAreaDemo();
    frame.pack();
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setTitle("TextAreaDemo");
    frame.setVisible(true);
  }
  public TextAreaDemo() {
    // Set title, text and image in the description panel
    descriptionPanel.setTitle("Canada");
    String description = "The Maple Leaf flag \n\n" +
      "The Canadian National Flag was adopted by the Canadian " +
      "Parliament on October 22, 1964 and was proclaimed into law " +
      "by Her Majesty Queen Elizabeth II (the Queen of Canada) on " +
      "February 15, 1965. The Canadian Flag (colloquially known " +
      "as The Maple Leaf Flag) is a red flag of the proportions " +
      "two by length and one by width, containing in its center a " +
      "white square, with a single red stylized eleven-point " +
      "mapleleaf centered in the white square.";
    descriptionPanel.setDescription(description);
    descriptionPanel.setImageIcon(new ImageIcon("L:/COP 3330 - Summer 2008/images/ca.gif"));
    // Add the description panel to the frame
    setLayout(new BorderLayout());
    add(descriptionPanel, BorderLayout.CENTER);
  }
}
```

Example – `TextAreaDemo`



The Maple Leaf flag

The Canadian National Flag was adopted

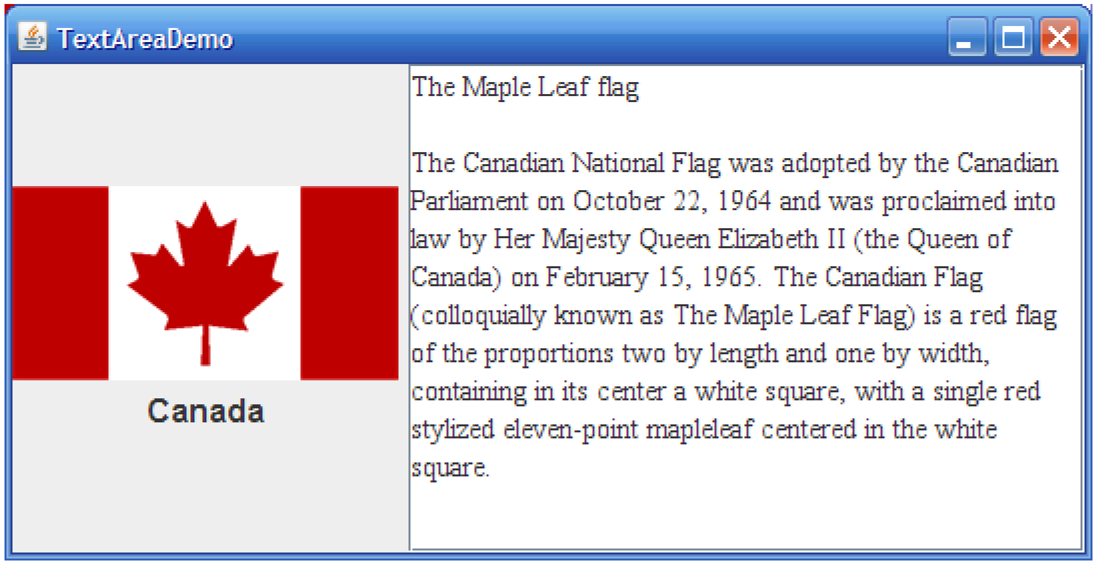
Initial GUI

The text area is inside a `JScrollPane`, which provides scrolling functions for the text area. Scroll bars automatically appear if there is more text than the physical size of the text area.

The `lineWrap` property is set to `true` so that the line is automatically wrapped when the text cannot fit in one line.

The `wrapStyleWord` property is set to `true` so that the line is wrapped on words rather than characters.

The text area is set non-editable so you cannot edit the description in the text area.



The Maple Leaf flag

The Canadian National Flag was adopted by the Canadian Parliament on October 22, 1964 and was proclaimed into law by Her Majesty Queen Elizabeth II (the Queen of Canada) on February 15, 1965. The Canadian Flag (colloquially known as The Maple Leaf Flag) is a red flag of the proportions two by length and one by width, containing in its center a white square, with a single red stylized eleven-point mapleleaf centered in the white square.

GUI after re-sizing – notice absence of slider bar