COP 3330: Object-Oriented Programming Summer 2011

Introduction To GUIs and Event-Driven Programming In Java – Part 2

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3330/sum2011

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

Using Panels as Subcontainers

- Suppose that you want to place ten buttons and a text field in a frame. The buttons are placed in a grid formation, but the text field is to be placed on a separate row.
- It would be difficult to achieve this effect by placing all of the components into a single container. With Java GUI programming, you can divide a window into panels.
- Panels act as subcontainers to group user-interface components. We can then add the buttons to one panel and then add the panel into the frame.
- The Swing version of panel is `JPanel`. You can use

  `new JPanel()` to create a panel with a default

  `FlowLayout` manager

  or –

  `new JPanel(LayoutManager)` to create a panel with the specified layout manager.
- The following example illustrates using panels as subcontainers.

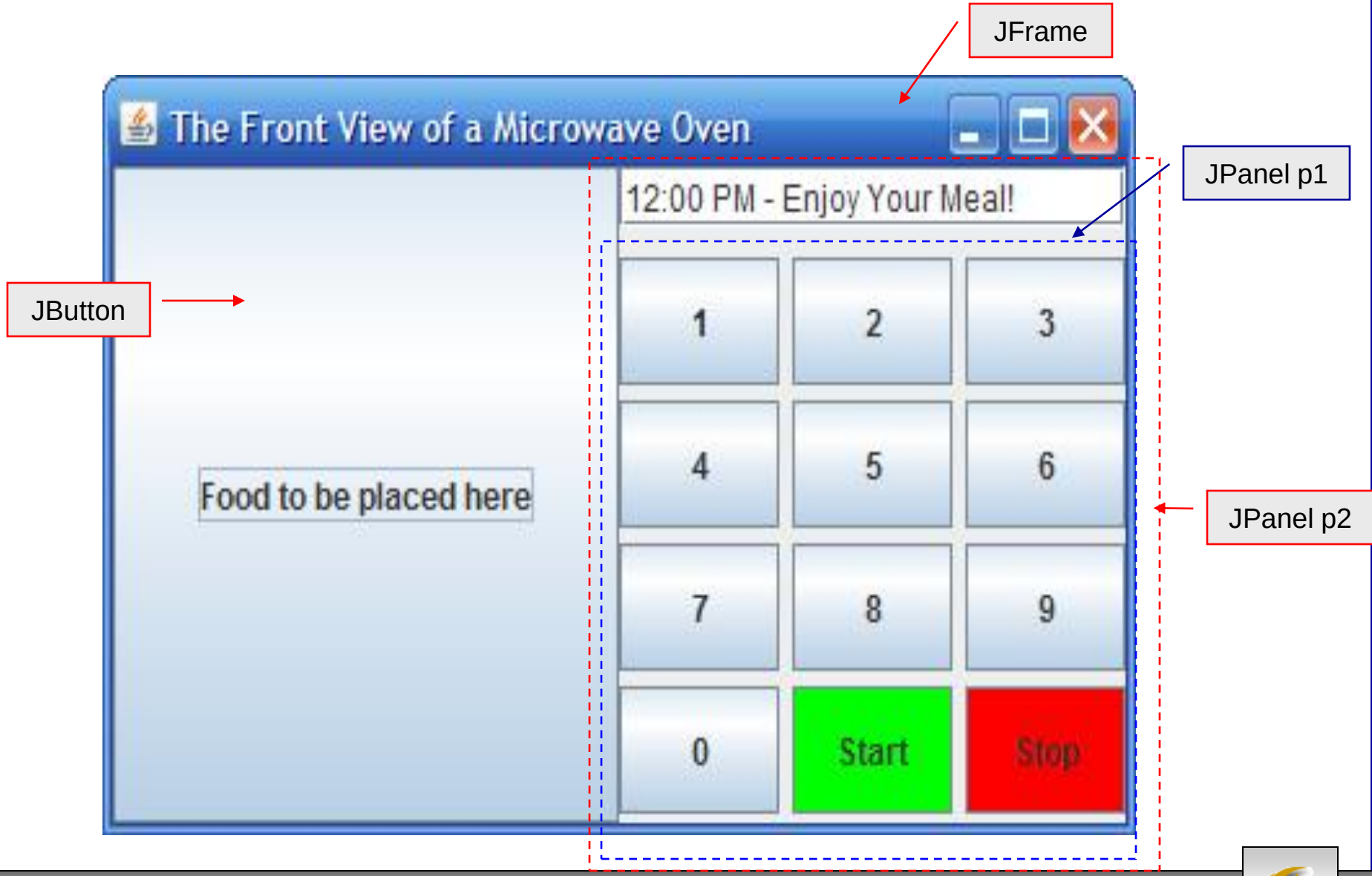
Example – Using Panels

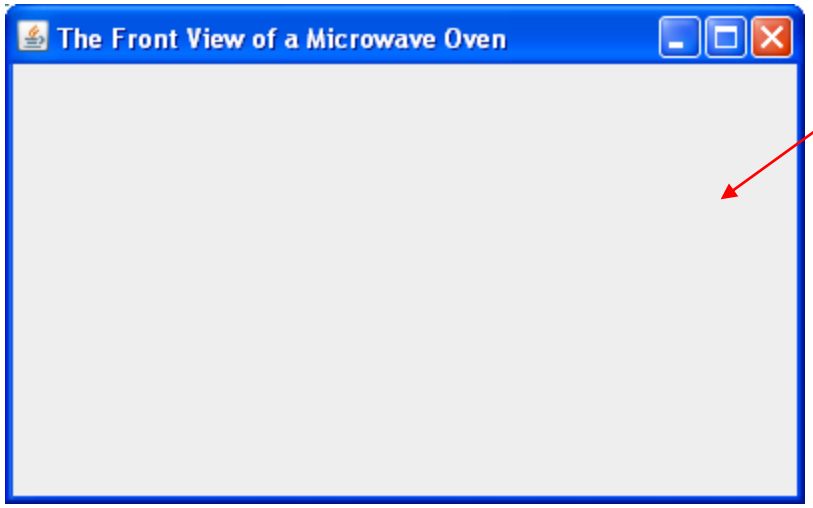
```
import java.awt.*;
import javax.swing.*;

public class MicrowaveOvenFrontPanel extends JFrame {
  public MicrowaveOvenFrontPanel() {
    // Create panel p1 for the buttons and set GridLayout
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(4, 3, 5, 5));
    // Add buttons to the panel
    for (int i = 1; i <= 9; i++) {
      p1.add(new JButton("" + i));
    }//end for stmt
    p1.add(new JButton("" + 0));
    JButton start = new JButton("Start");
    start.setBackground(Color.GREEN);
    p1.add(start);
    JButton stop = new JButton("Stop");
    stop.setBackground(Color.RED);
    p1.add(stop);
    // Create panel p2 to hold a text field and p1
    JPanel p2 = new JPanel(new BorderLayout(10,10));
    p2.add(new JTextField("12:00 PM - Enjoy Your Meal!"),
      BorderLayout.NORTH);
    p2.add(p1, BorderLayout.CENTER);
```


HouseHoldPet.java | ChainedExceptions.ja | *MicrowaveOvenFrontP | 28

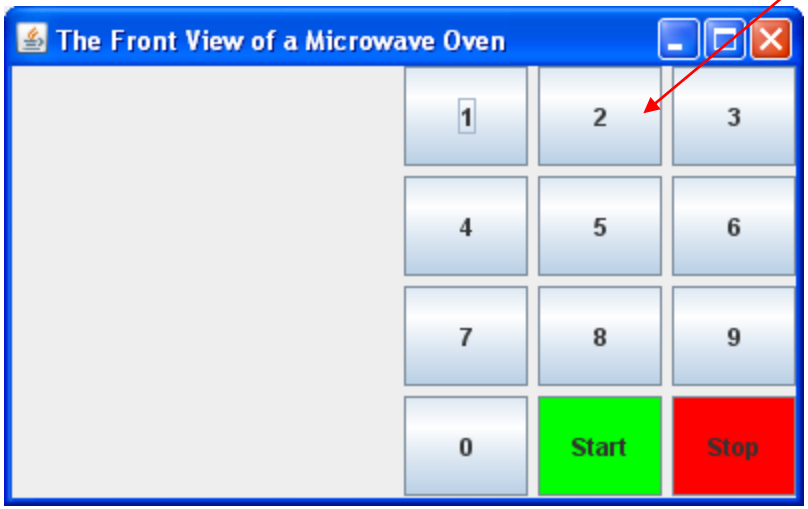
```
    // add contents into the frame
    add(p2, BorderLayout.EAST);
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }//end constructor method

  /** Main method */
  public static void main(String[] args) {
    MicrowaveOvenFrontPanel frame = new MicrowaveOvenFrontPanel
    frame.setTitle("The Front View of a Microwave Oven");
    frame.setSize(400, 250);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }//end main method
}//end class MicrowaveOvenFrontPanel
```

The program uses panel p1 (`GridLayout` manager) to group the number buttons, the Start button, and the Stop button, and panel p2 (`BorderLayout` manager) to hold a text field in the north and the panel p1 in the center. The button representing the food is placed in the center of the frame, and p2 is placed in the east of the frame. See pages 6 and 7.

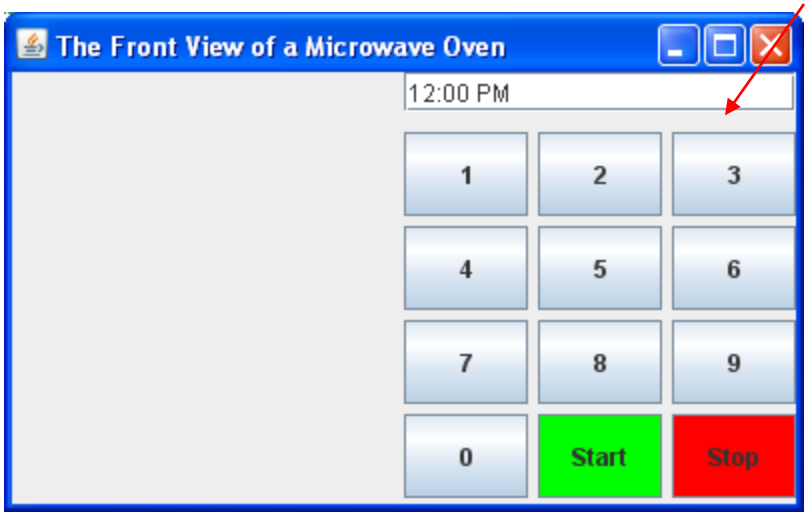

JFrame
The Front View of a Microwave Oven
JPanel p1
12:00 PM - Enjoy Your Meal!
JButton
1
2
3
4
5
6
Food to be placed here
JPanel p2
7
8
9
0
Start
Stop



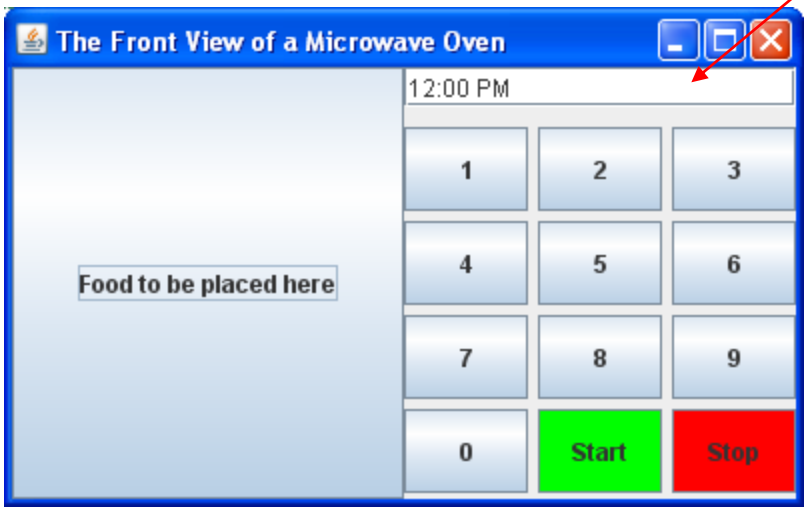
Initial frame – no components added yet.



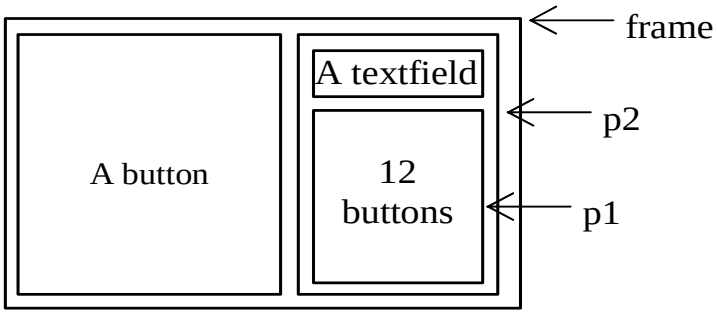
Showing just panel p1 added to the frame.



Showing panel p2 added to the frame – panel p2 uses a `BorderLayout` with the `JTextField` placed in the North area and panel p1 placed in the Center area. Other areas on the BorderLayout are not used.



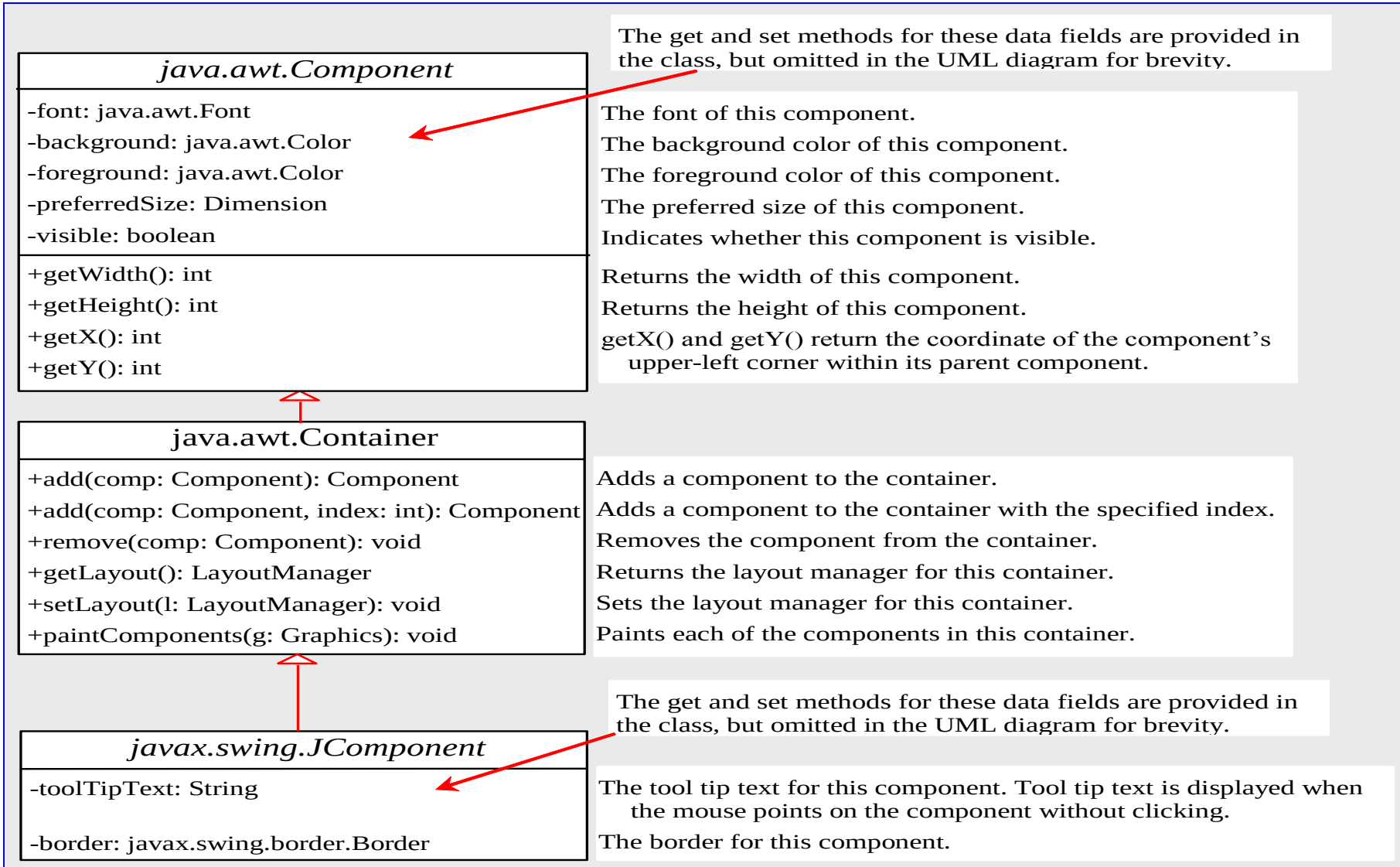
Showing final frame using `BorderLayout`. Added a `JButton` ("Food to be placed here") to the Center area. Added panel p2 to the East area.



Common Features of Swing GUI Components

- We've already used several GUI components (e.g., `JFrame`, `Container`, `JPanel`, `JButton`, `JLabel`, `JTextField`) in the previous example.
- We'll see many more GUI components as we continue on, but it is important to understand the common features of Swing GUI components.
- The `Component` class is the superclass for all GUI components and containers. All Swing GUI components (except `JFrame`, `JApplet`, and `JDialog`) are subclasses of `JComponent` (see Part 1 pages 5 and 9).
- The next page illustrates some of the more commonly used methods in `Component`, `Container`, and `JComponent` for manipulating properties like font, color, size, tool tip text, and border.

Common Features of Swing Components



Common Features of Swing GUI Components

- A tool tip text is text displayed on the component when you move the mouse on the component. It is often used to describe the function of a component.
- You can set the border on any object of the `JComponent` class. Swing has several types of borders.
  - For example, to create a titled border, use:

    ```
    new TitledBorder(String title)
    ```

  - To create a line border use:

    ```
    new LineBorder(Color color, int width)
    ```

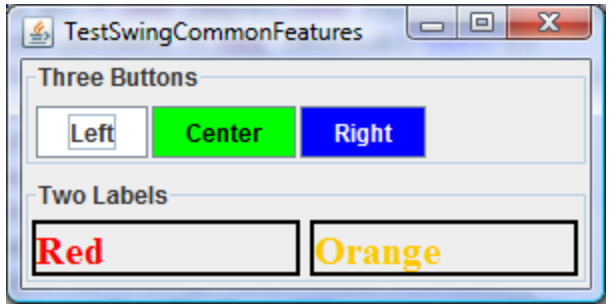
    where width specifies the thickness of the line in pixels.
- The following example illustrates some of the common Swing features.

Example – Common Swing Features

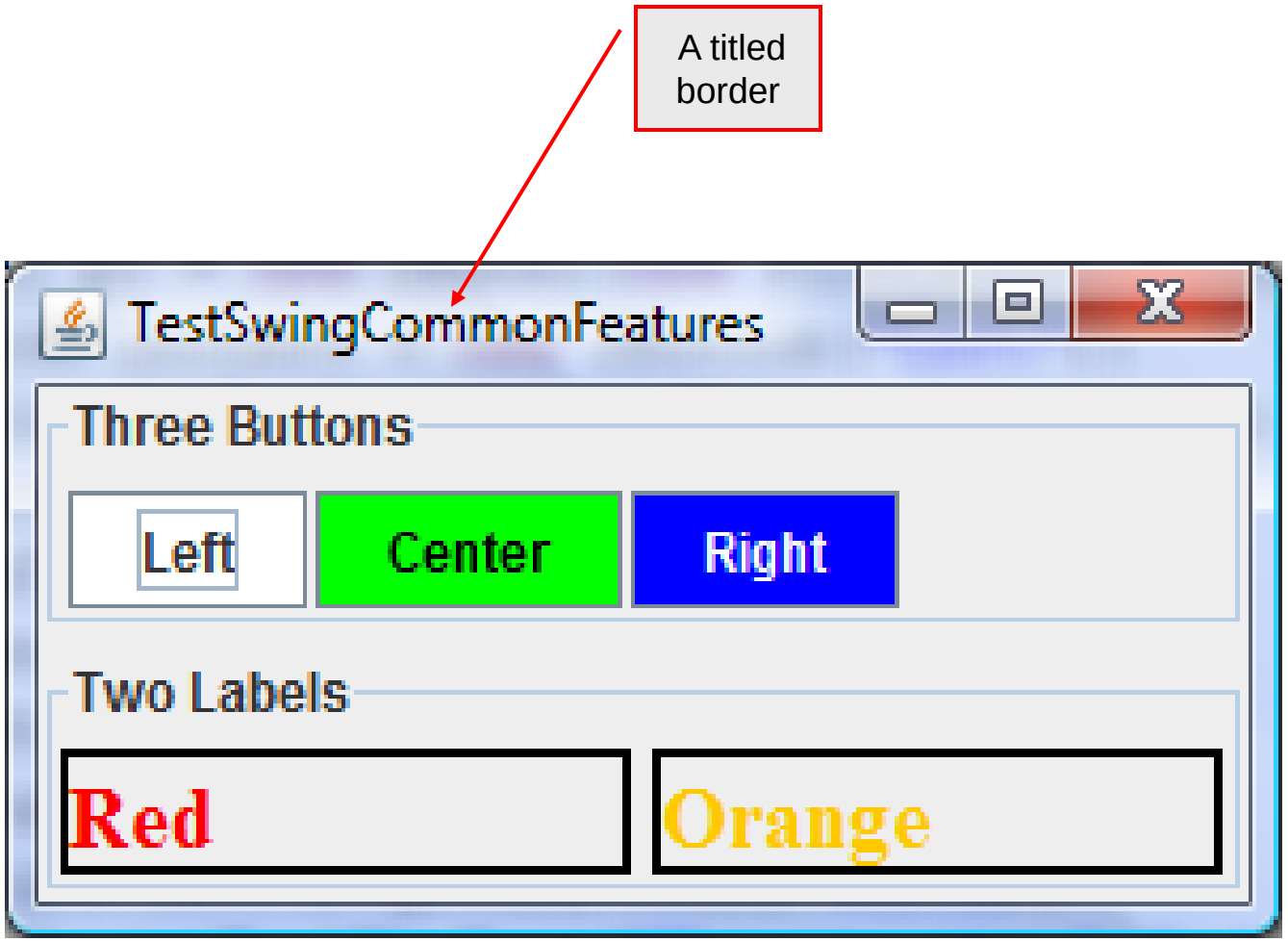
```
// Class: TestSwingCommonFeatures
// GUI Notes - Part 2 - Summer 2011
// MJL 6/28/2011

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

public class TestSwingCommonFeatures extends JFrame {
  public TestSwingCommonFeatures() {
    // Create a panel to group three buttons
    //uncomment this for first example
    JPanel p1 = new JPanel(new FlowLayout(FlowLayout.LEFT, 2, 2));
    //comment this out for first example
    //JPanel p1 = new JPanel(new GridLayout(1, 3, 5, 5));
    JButton jbtLeft = new JButton("Left");
    JButton jbtCenter = new JButton("Center");
    JButton jbtRight = new JButton("Right");
    jbtLeft.setBackground(Color.WHITE);
    jbtCenter.setBackground(Color.GREEN);
    jbtCenter.setForeground(Color.BLACK);
    jbtRight.setBackground(Color.BLUE);
    jbtRight.setForeground(Color.WHITE);
    jbtLeft.setToolTipText("This is the Left button");
    p1.add(jbtLeft);
    p1.add(jbtCenter);
    p1.add(jbtRight);
    p1.setBorder(new TitledBorder("Three Buttons"));
```


MicrowaveOvenFrontPa | MicrowaveOvenFrontPa | *TestSwingCommonFeat | 19

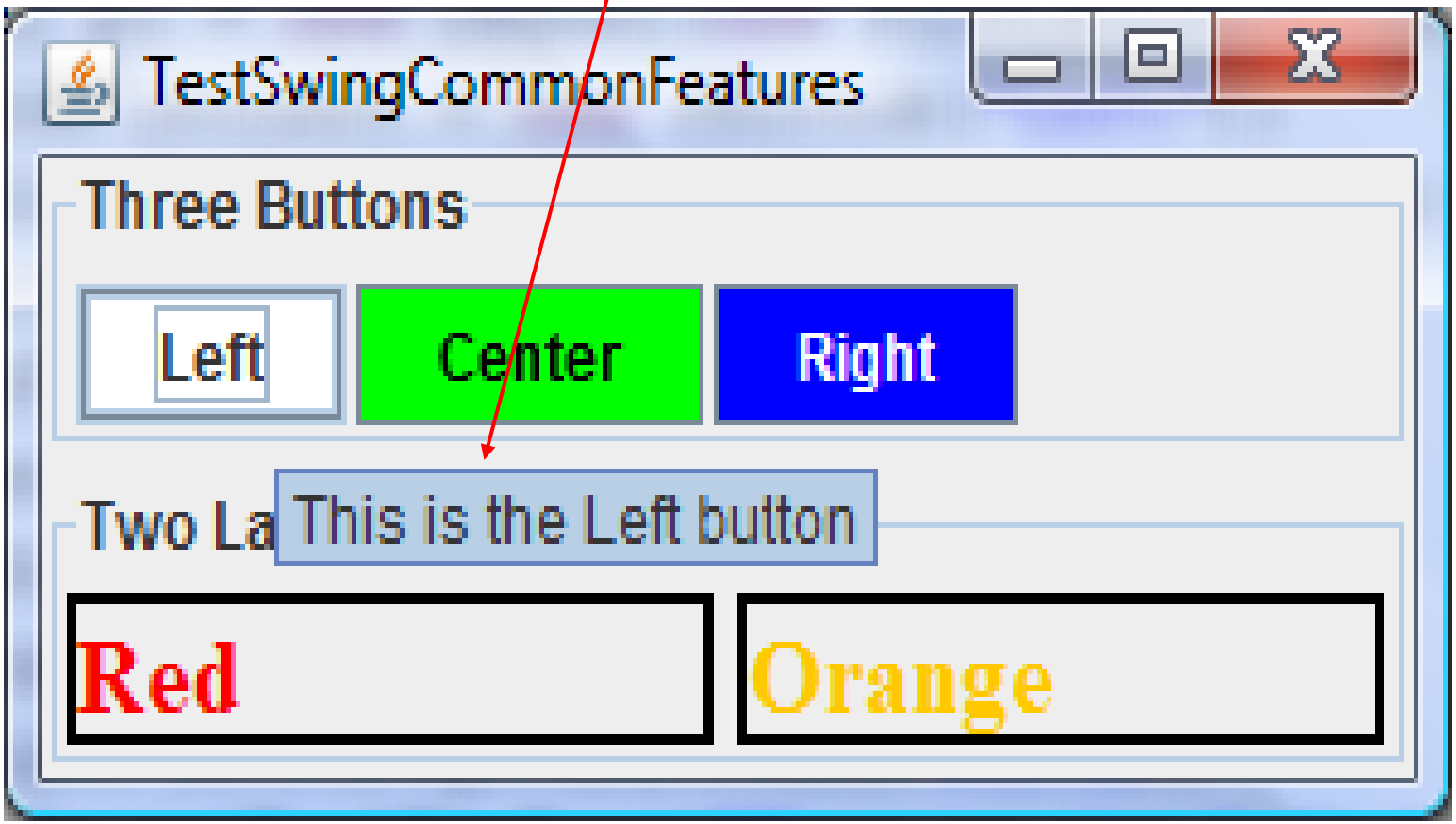
```
    // Create a font and a line border
    Font largeFont = new Font("TimesRoman", Font.BOLD, 20);
    Border lineBorder = new LineBorder(Color.BLACK, 2);
    // Create a panel to group two labels
    JPanel p2 = new JPanel(new GridLayout(1, 2, 5, 5));
    JLabel jlblRed = new JLabel("Red");
    JLabel jlblOrange = new JLabel("Orange");
    jlblRed.setForeground(Color.RED);
    jlblOrange.setForeground(Color.ORANGE);
    jlblRed.setFont(largeFont);
    jlblOrange.setFont(largeFont);
    jlblRed.setBorder(lineBorder);
    jlblOrange.setBorder(lineBorder);
    p2.add(jlblRed);
    p2.add(jlblOrange);
    p2.setBorder(new TitledBorder("Two Labels"));
    // Add two panels to the frame

    setLayout(new GridLayout(2, 1, 5, 5));
    add(p1);
    add(p2);
  }//end constructor

  public static void main(String[] args) {
    // Create a frame and set its properties
    JFrame frame = new TestSwingCommonFeatures();
    frame.setTitle("TestSwingCommonFeatures");
    frame.setSize(300, 150);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }//end main method
```



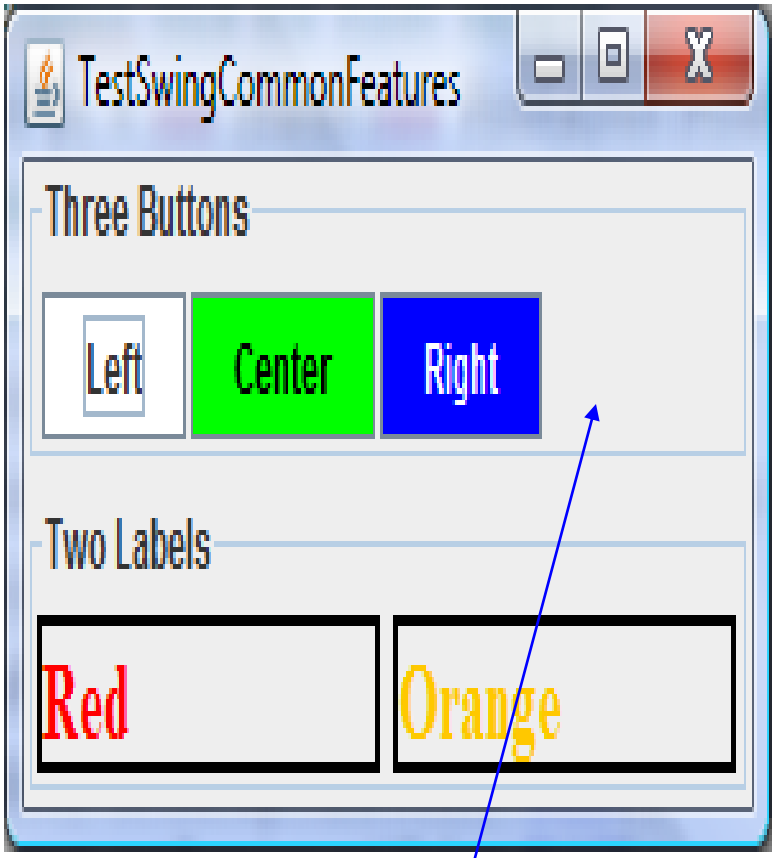
Insertion point for statements shown on page 16.


A titled
border
TestSwingCommonFeatures
Three Buttons
Left
Center
Right
Two Labels
Red
Orange

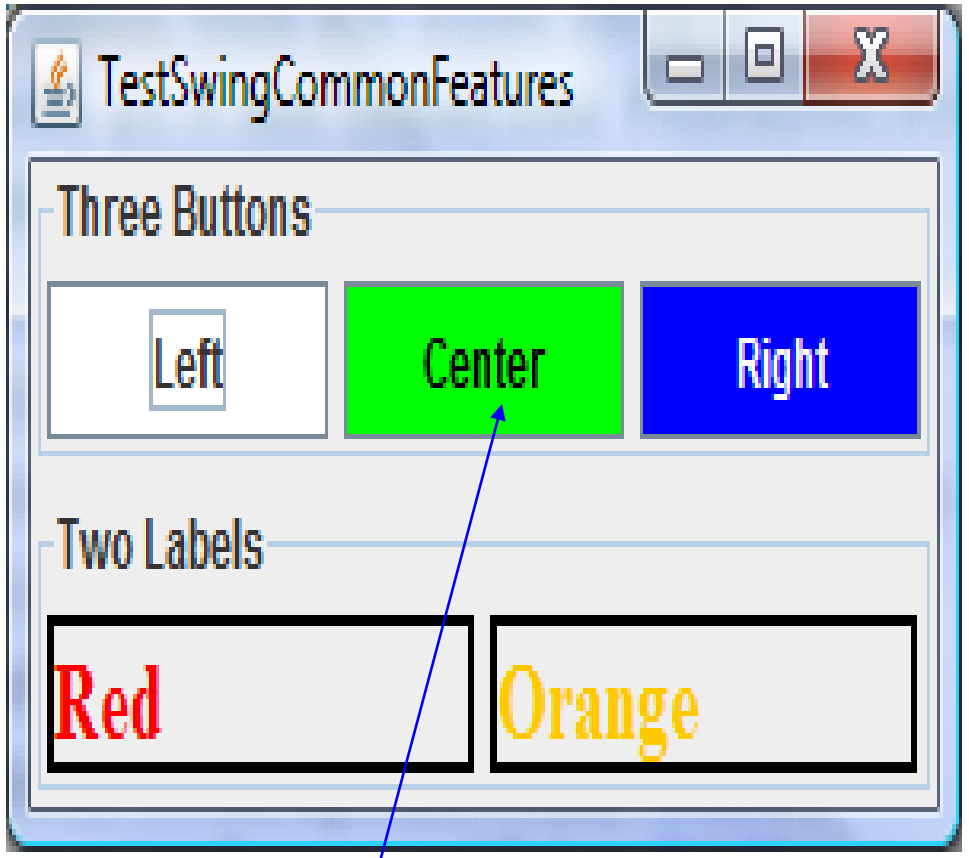
Moving the mouse over the left button causes the tool text tip to display.



Change the layout manager for panel p1 to a `GridLayout`. Notice the difference?



Flow Layout



Grid Layout

NOTE

- The same property may have different default values in different components.
- For example, the visible property in `JFrame` is false by default, but it is true in every instance of `JComponent` (e.g., `JButton` and `JLabel`) by default.
- To display a `JFrame`, you must invoke `setVisible(true)` to set the visible property true, but you don't need to set this property for a `JButton` or a `JLabel` because it is already true.
- To make a `JButton` or a `JLabel` invisible, you need to invoke `setVisible(false)` on the button or label.
- Rerun the `TestSwingCommonFeatures` program again after inserting the two lines, shown below, immediately prior to adding the panels to the frame (see page 12).

```
jbtLeft.setVisible(false);
jlblRed.setVisible(false);
```

The effect of adding these two lines is shown on the next page.

Making button and label invisible

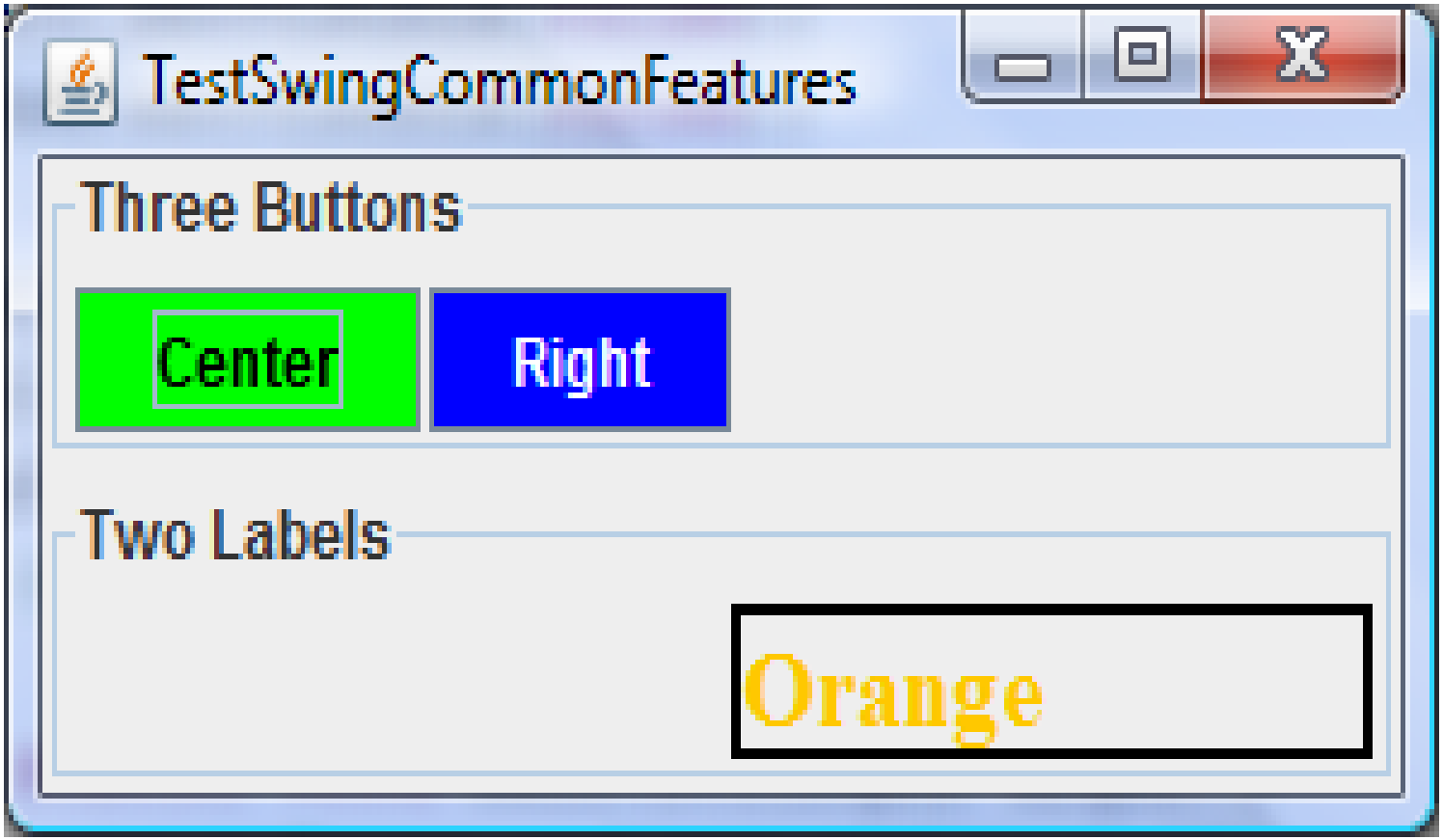


Image Icons

- An icon is a fixed-size picture; typically it is small and used to decorate components.
- Images are stored in image files. Java currently supports three image formats: GIF (Graphics Interchange Format), JPEG (Joint Photographic Experts Group), and PNG (Portable Network Graphics). The image file names for these types end with `.gif`, `.jpg`, and `.png` respectively. If you have a bitmap file or image files in other formats, you can use image-processing utilities to convert them into GIF, JPEG, or PNG formats for use in Java.
- To display an image icon, first create an `ImageIcon` object using `new javax.swing.ImageIcon(filename)`. For example, the following statement creates an icon from an image file `us.gif` in the `image` directory under the current class path: `ImageIcon icon = new ImageIcon("image/us.gif");`

Image Icons

- The back slash (\) is the Windows file path notation. In Unix, the forward slash (/) should be used.
- In Java, the forward slash (/) is used to denote a relative file path under the Java classpath (e.g., `image/us.gif`, as in this example).
- File names are not case sensitive in Windows but are case sensitive in Unix. To enable your programs to run on all platforms, name all image files consistently using only lowercase letters.
- The following example illustrates image icons. This example uses both relative and absolute path names to the image files so that you'll have examples of both types.

Example – Using Image Icons

Relative address – in current project directory

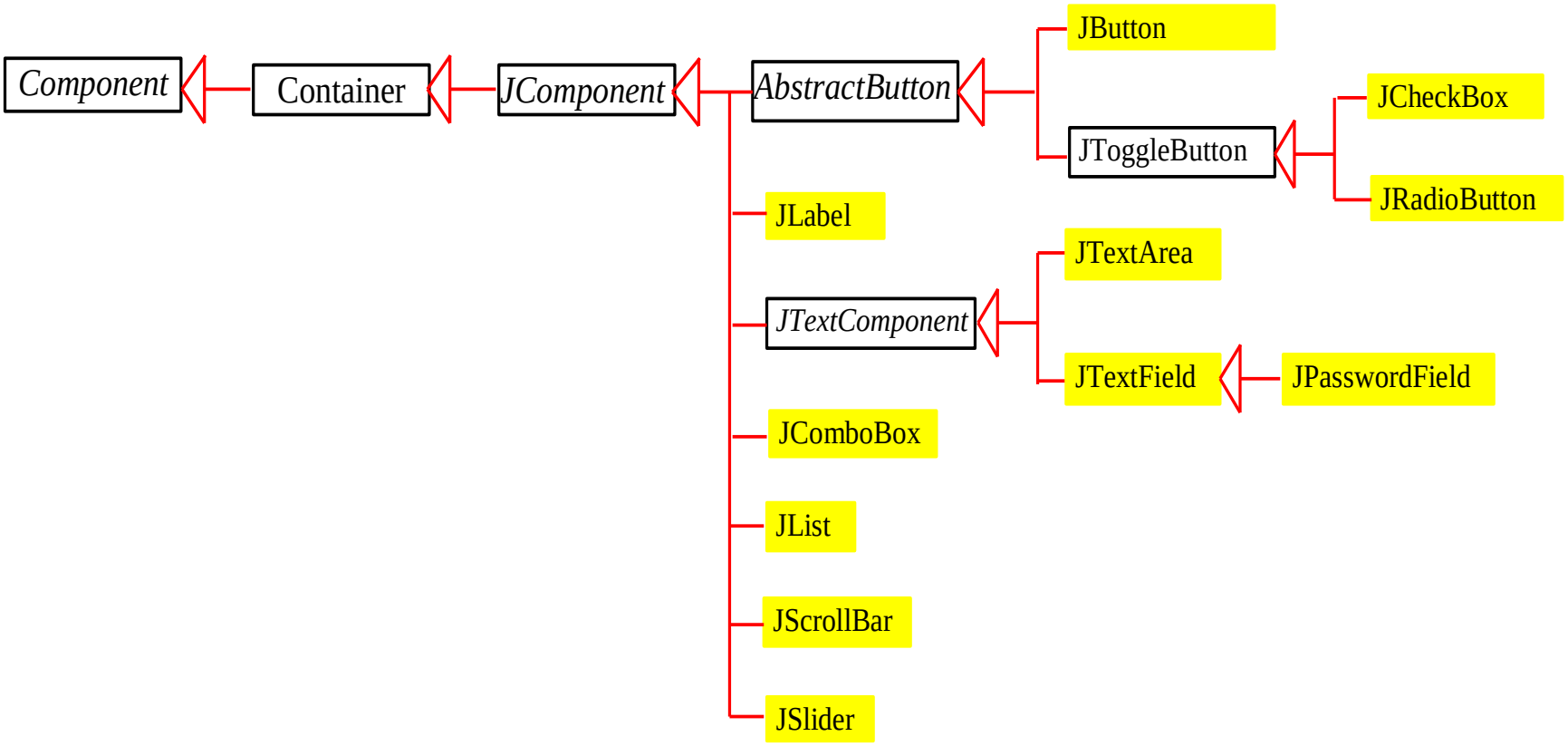
Absolute address – uses fully specified path

```
// Class: TestImageIcons
// GUI Notes - COP 3330 - Summer 2011
// MJL  6/28/2011

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

public class TestImageIcons extends JFrame {
  private ImageIcon usIcon = new ImageIcon("us.gif");
  private ImageIcon myIcon = new ImageIcon("C:/Courses/COP 3330 - OOP/Summer 2011/image/sw-t.gif");
  private ImageIcon frIcon = new ImageIcon("C:/Courses/COP 3330 - OOP/Summer 2011/image/fr.gif");
  private ImageIcon ukIcon = new ImageIcon("C:/Courses/COP 3330 - OOP/Summer 2011/image/uk.gif");

  public TestImageIcons() {
    setLayout(new GridLayout(1, 4, 5, 5));
    add(new JLabel(usIcon));
    add(new JLabel(myIcon));
    add(new JButton(frIcon));
    add(new JButton(ukIcon));
  }//end constructor

  /** Main method */
  public static void main(String[] args) {
    TestImageIcons frame = new TestImageIcons();
    frame.setTitle("TestImageIcon");
    frame.setSize(500, 125);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }//end main method
}//end class TestImageIcons
```

Commonly Used GUI Components

- A graphical user interface (GUI) makes a system user-friendly and easy to use. Creating a GUI requires creativity and knowledge of how GUI components work. Since the GUI components in Java are very flexible and versatile, you can create a wide assortment of useful user interfaces.
- Many Java IDEs provide tools for visually designing and developing GUIs that enable you to rapidly assemble the elements of a user interface for a Java application with minimal coding. However, such tools cannot do everything that you would like and you need to modify the programs that they produce, so you need to be familiar with the basic concepts of Java GUI programming.
- To this end, we'll examine many of the more commonly used GUI components in Java.

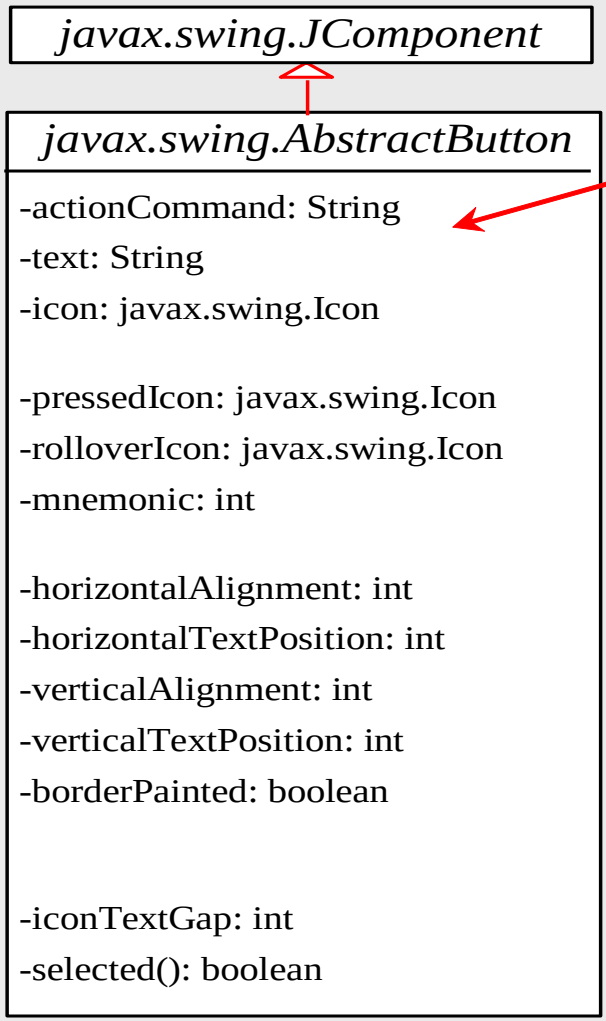
Commonly Used GUI Components



Buttons

- A button is a component that triggers an action event when clicked.
- Swing provides regular buttons, toggle buttons, check box buttons, and radio buttons.
- The common features of these buttons are generalized in `javax.swing.AbstractButton`.
- The UML for this class is shown on page 24.
- Many common buttons are defined in the `JButton` class. The `JButton` class extends `AbstractButton` and its UML is shown on page 25.

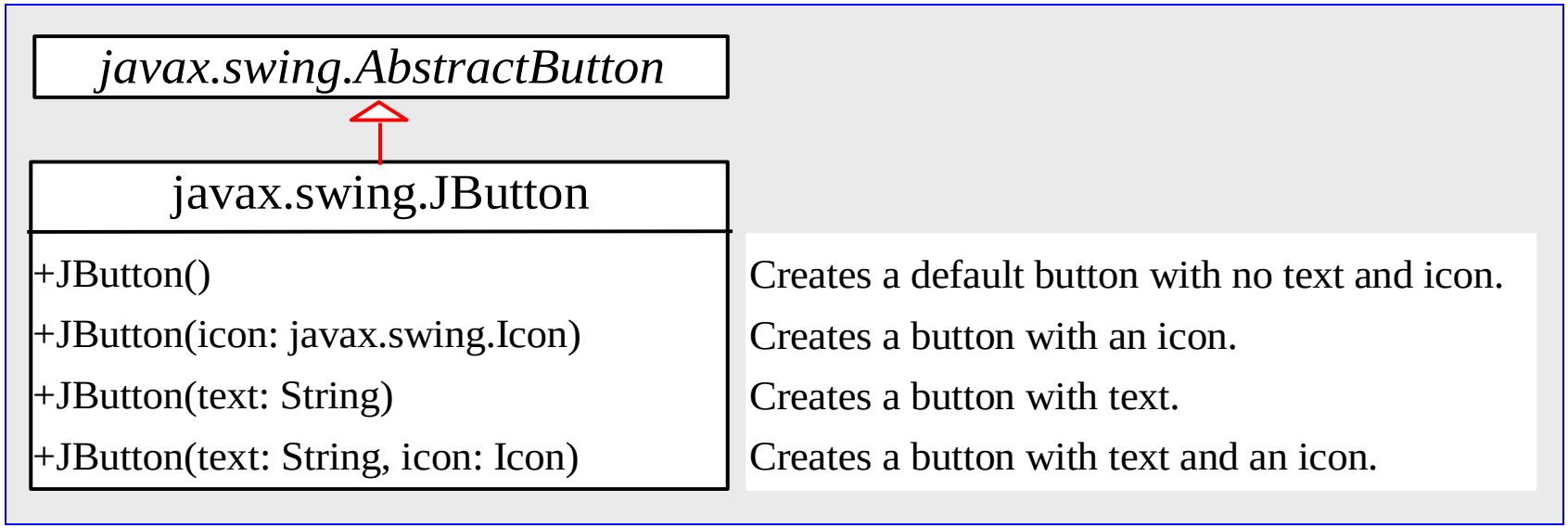
javax.swing.AbstractButton



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

-actionCommand: String	The action command of this button.
-text: String	The button's text (i.e., the text label on the button).
-icon: javax.swing.Icon	The button's default icon. This icon is also used as the "pressed" and "disabled" icon if there is no explicitly set pressed icon.
-pressedIcon: javax.swing.Icon	The pressed icon (displayed when the button is pressed).
-rolloverIcon: javax.swing.Icon	The rollover icon (displayed when the mouse is over the button).
-mnemonic: int	The mnemonic key value of this button. You can select the button by pressing the ALT key and the mnemonic key at the same time.
-horizontalAlignment: int	The horizontal alignment of the icon and text (default: CENTER).
-horizontalTextPosition: int	The horizontal text position relative to the icon (default: RIGHT).
-verticalAlignment: int	The vertical alignment of the icon and text (default: CENTER).
-verticalTextPosition: int	The vertical text position relative to the icon (default: CENTER).
-borderPainted: boolean	Indicates whether the border of the button is painted. By default, a regular button's border is painted, but the borders for a check box and a radio button is not painted.
-iconTextGap: int	The gap between the text and the icon on the button (JDK 1.4).
-selected(): boolean	The state of the button. True if the check box or radio button is selected, false if it's not.

javax.swing.JButton



Icons, Pressed Icons, and Rollover Icons

- A regular button has a default icon, a pressed icon, and a rollover icon.
- Normally, you use the default icon. The other icons are for special effects. A pressed icon is displayed when a button is pressed, and a rollover icon is displayed when the mouse is positioned over the button but not pressed.
- The example on the next page, displays the American flag as a regular icon, the Canadian flag as a pressed icon and the British flag as a rollover icon.

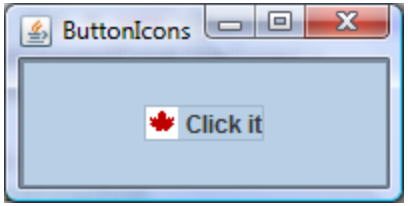
Icons, Pressed Icons, and Rollover Icons

MicrowaveOvenFrontPa | TestSwingCommonFeatu | TestImageIcons.java | TestButtonIcons.java | 20

```
// Class: TestButtonIcons
// GUI Notes - COP 3330 - Summer 2011
// MJL 6/28/2011

import javax.swing.*;

public class TestButtonIcons extends JFrame  {

    public TestButtonIcons() {
        ImageIcon usIcon = new ImageIcon("C:/Courses/COP 3330 - OOP/Summer 2011/image/usIcon.gif");
        ImageIcon caIcon = new ImageIcon("C:/Courses/COP 3330 - OOP/Summer 2011/image/caIcon.gif");
        ImageIcon ukIcon = new ImageIcon("C:/Courses/COP 3330 - OOP/Summer 2011/image/ukIcon.gif");

        JButton jbt = new JButton("Click it", usIcon);
        jbt.setPressedIcon(caIcon);
        jbt.setRolloverIcon(ukIcon);

        add(jbt);
      }//end constructor

  public static void main(String[] args) {
    // Create a frame and set its properties
    JFrame frame = new TestButtonIcons();
    frame.setTitle("ButtonIcons");
    frame.setSize(200, 100);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }//end main method
}//end class TestButtonIcons
```

default icon

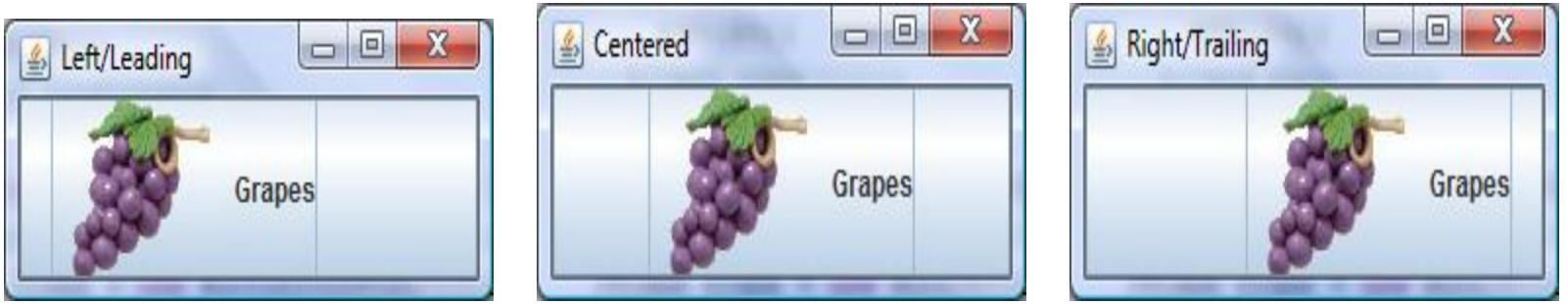
rollover icon



pressed icon

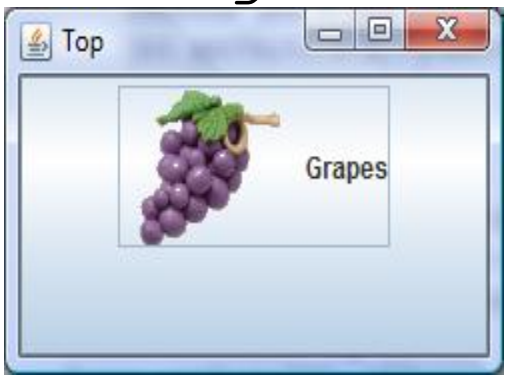
Alignments

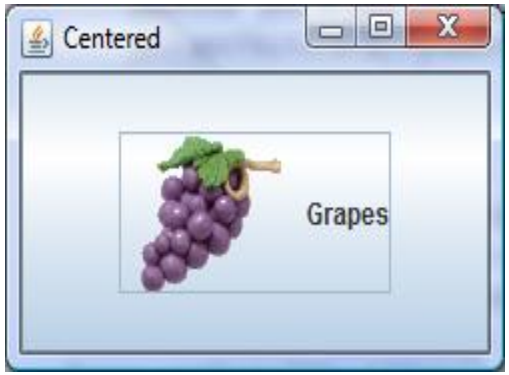
- Horizontal alignment specifies how the icon and text are placed horizontally on a button.
- You can set the horizontal alignment using one of the five constants: `LEADING`, `LEFT`, `CENTER`, `RIGHT`, `TRAILING`.
  - At present, `LEADING` and `LEFT` are the same and `TRAILING` and `RIGHT` are the same. Future implementation may distinguish them.
- The default horizontal alignment is `SwingConstants.TRAILING`.

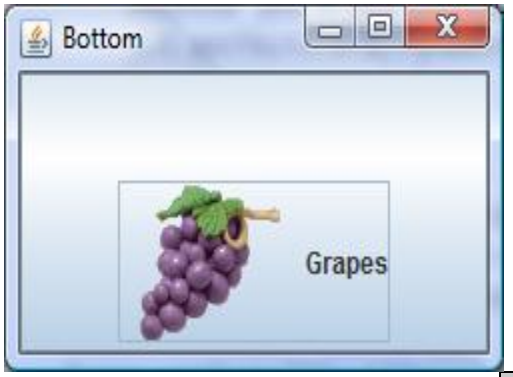


Alignments

- Vertical alignment specifies how the icon and text are placed vertically on a button.
- You can set the vertical alignment using one of the three constants: `TOP`, `CENTER`, `BOTTOM`.
- The default vertical alignment is `SwingConstants.CENTER`.

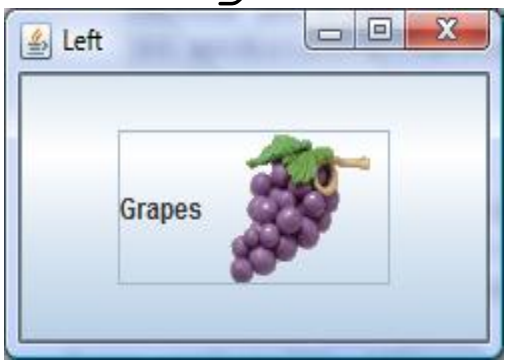


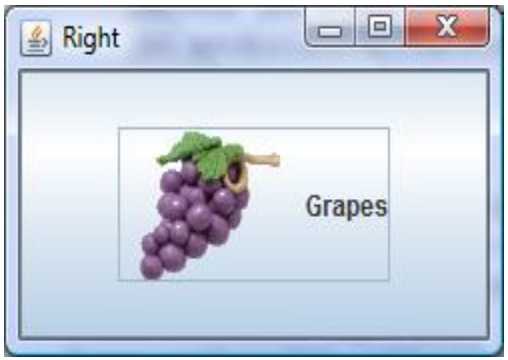




Text Positions

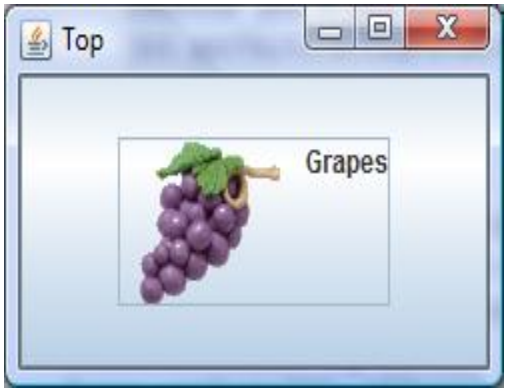
- Horizontal text position specifies the horizontal position of the text relative to the icon.
- You can set the horizontal text position using one of the five constants: `LEADING`, `LEFT`, `CENTER`, `RIGHT`, `TRAILING`.
- The default horizontal text position is `SwingConstants.RIGHT`.

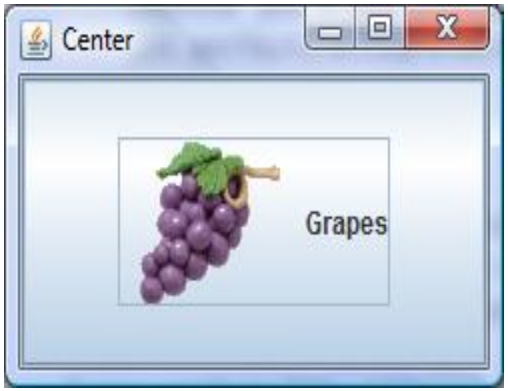


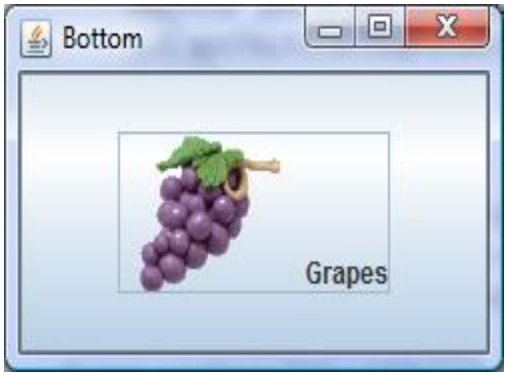


Text Positions

- Vertical text position specifies the vertical position of the text relative to the icon.
- You can set the vertical text position using one of the three constants: `TOP`, `CENTER`.
- The default vertical text position is `SwingConstants.CENTER`.





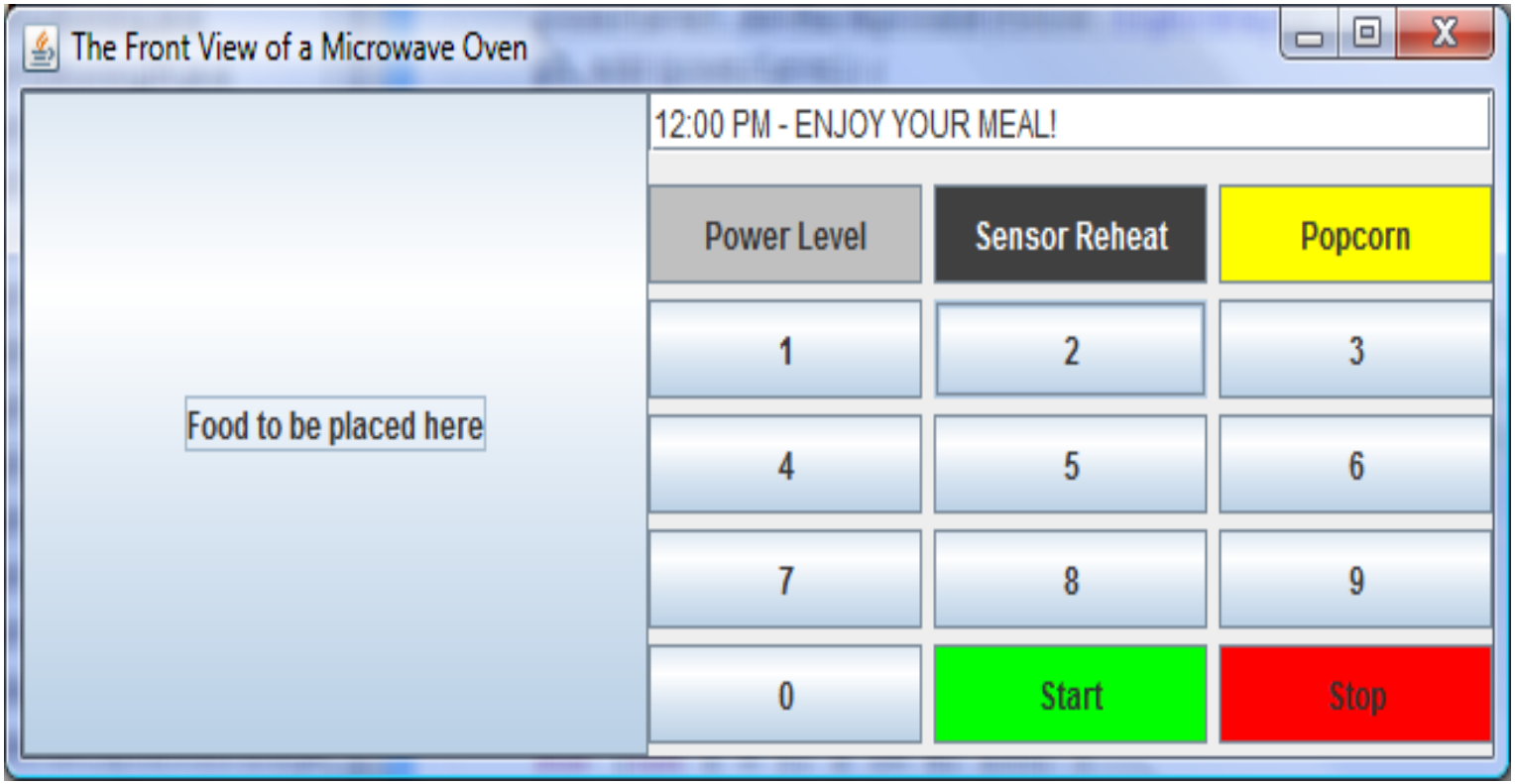


NOTE

- The constants `LEFT`, `CENTER`, `RIGHT`, `LEADING`, `TRAILING`, `TOP`, and `BOTTOM` used in `AbstractButton` are also used in many other Swing components. These constants are centrally defined in the `javax.swing.SwingConstants` interface.
- Since all Swing GUI components implement `SwingConstants`, you can reference the constants through `SwingConstants` (class reference) or a GUI component (instance reference). For example, `SwingConstants.CENTER` is the same as `JButton.CENTER`.
- `JButton` can generate many types of events (as we'll see later), but often you need to respond to an `ActionEvent`. When a button is pressed, it generates an `ActionEvent`.

Practice Problem

- Modify the microwave oven front panel so that the GUI looks like the following.

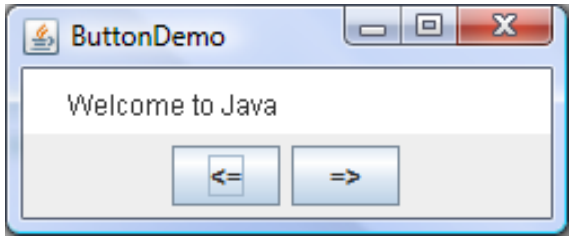


Using Buttons

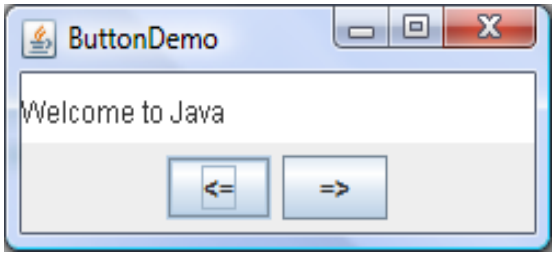
- As a brief introduction to event-driven programming, the next example, creates a message panel that displays a message and then allows the user, through the use of buttons, to move the message either left or right in the panel.
- The major steps in the program are:

1. Create the user interface.
2. Create a `MessagePanel` object to display the message. (The `MessagePanel` class is separate from this program and we'll use it again later. In this case the `messagePanel` object is deliberately declared `protected` so that it can be referenced by a subclass in future examples.) Place it in the center of the frame, and create two buttons on a panel and place the panel in the south area of the frame.
3. Process the event. Create and register listeners for processing the action event to move the message left or right depending on which button was clicked (pressed).

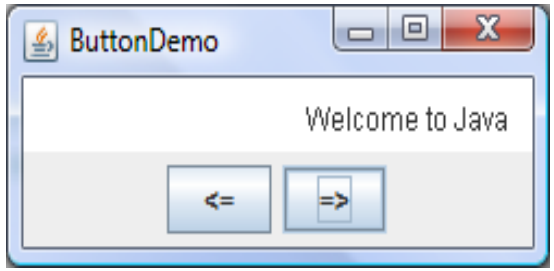
Using Buttons



Initial frame



Frame after user has clicked the left button a few times



Frame after user has clicked the right button a few times

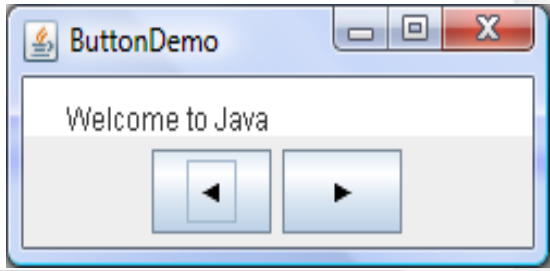
ButtonDemo.java | HeartRateZones.java | TemperatureConverter | 27

```
// Class: ButtonDemo
// GUI Notes - COP 3330 - Part 2 - Summer 2011
// MJL 6/28/2011

import java.awt.*;

public class ButtonDemo extends JFrame {
  // Create a panel for displaying message
  protected MessagePanel messagePanel = new MessagePanel("Welcome to Java");

  // Declare two buttons to move the message left and right
  private JButton jbtLeft = new JButton("<=");
  private JButton jbtRight = new JButton("=>");

  public static void main(String[] args) {
    ButtonDemo frame = new ButtonDemo();
    frame.setTitle("ButtonDemo");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(250, 100);
    frame.setVisible(true);
  }
```


ButtonDemo.java | TemperatureConverter | HeartRateZones.java | TemperatureConverter | 26

```
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("C:/Courses/COP 3330 - OOP/Summer 2011/image/left.gif"));
        //jbtRight.setIcon(new ImageIcon("C:/Courses/COP 3330 - OOP/Summer 2011/image/right.gif"
        //jbtLeft.setText(null);
        //jbtRight.setText(null);

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

        // Place panels in the frame
        setLayout(new BorderLayout());
        add(messagePanel, BorderLayout.CENTER);
        add(jpButtons, BorderLayout.SOUTH);
```

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



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

        // Place panels in the frame
        setLayout(new BorderLayout());
        add(messagePanel, BorderLayout.CENTER);
        add(jpButtons, BorderLayout.SOUTH);

        // Register listeners with the buttons
        jbtLeft.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            messagePanel.moveLeft();
          }
        });
        jbtRight.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            messagePanel.moveRight();
          }
        });
    }
}
```

Register listener for left button and set `actionPerformed()`

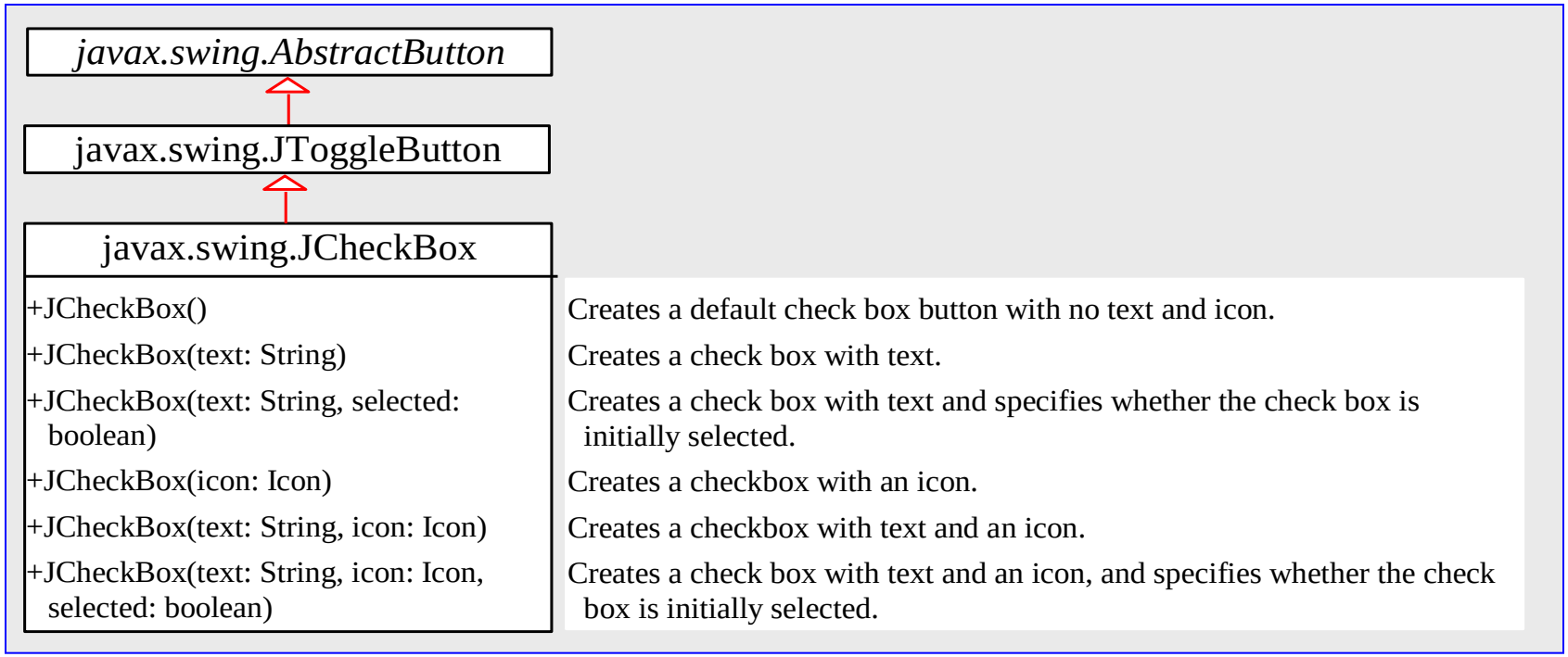
Register listener for right button and set `actionPerformed()`

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.

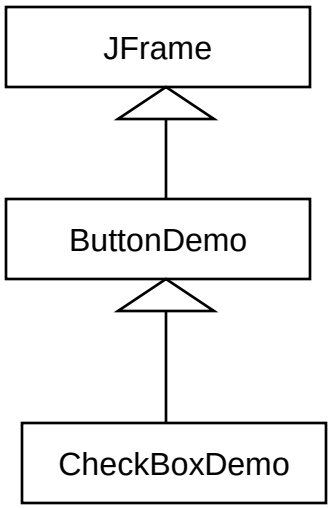
JCheckBox

- 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:



Example – CheckBoxDemo

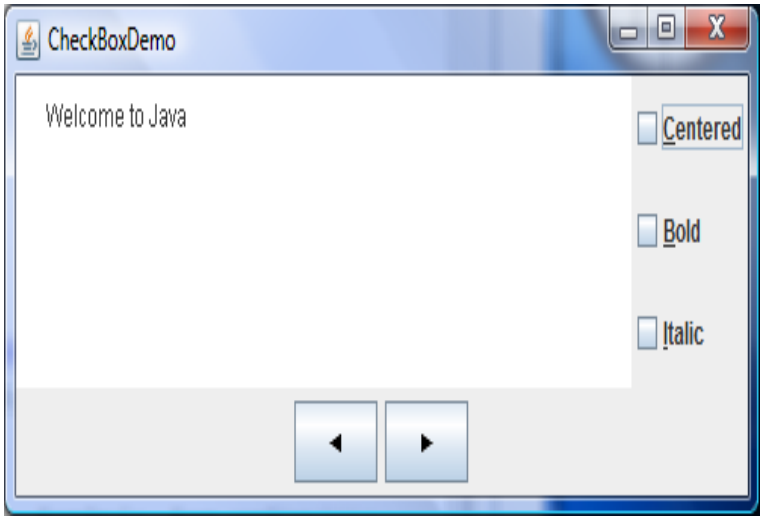
```
//Class: CheckBoxDemo

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);
 }//end main method

  public CheckBoxDemo() {
    // Set mnemonic keys
    jchkCentered.setMnemonic('C');
    jchkBold.setMnemonic('B');
    jchkItalic.setMnemonic('I');
    // Create a new panel to hold check boxes
    JPanel jpCheckBoxes = new JPanel();
    jpCheckBoxes.setLayout(new GridLayout(3, 1));
```



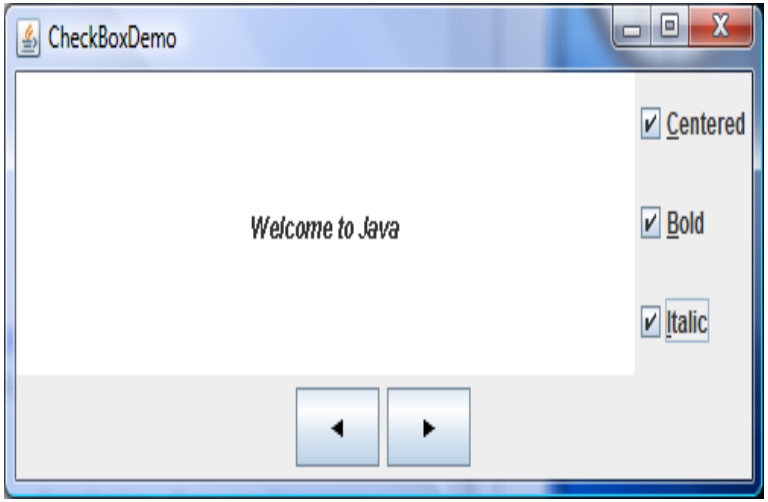
MicrowaveOvenFrontPa | MicrowaveOvenFrontPa | TemperatureConverter | CheckBoxDemo.java

Example – CheckBoxDemo

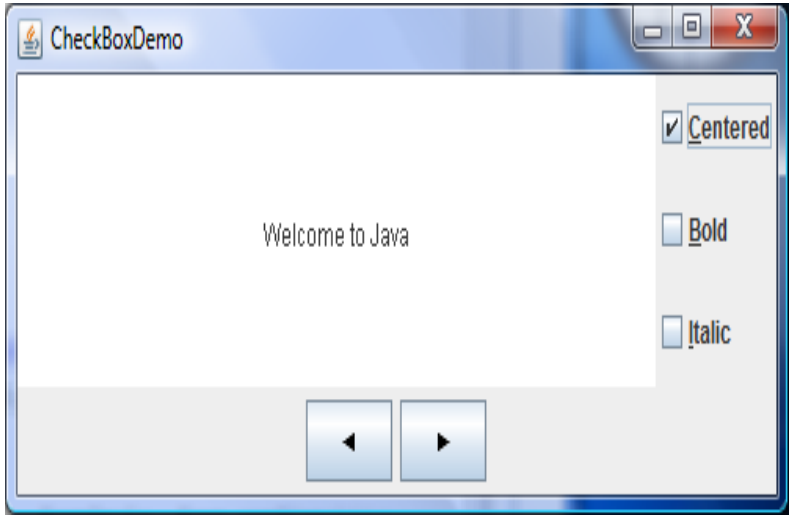
```
    jpCheckBoxes.add(jchkCentered);
    jpCheckBoxes.add(jchkBold);
    jpCheckBoxes.add(jchkItalic);
    add(jpCheckBoxes, BorderLayout.EAST);

    // 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();
      }
    });
  }
    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()));
    }//end method setNewFont
}//end class CheckBoxDemo
```

Initial GUI



After clicking all three checkboxes



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.

MicrowaveOvenFrontPa | MicrowaveOvenFrontPa | CheckBoxDemo.java | CheckBoxUsingItemEve

Example – CheckBoxUsingItemEvent

```
//Class: CheckBoxUsingItemEvent

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

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


MicrowaveOvenFrontPa | MicrowaveOvenFrontPa | CheckBoxDemo.java | CheckBoxUsingItemEve

```
    // 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();
      }
    });
  }

  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()));
  }//end method setNewFont
}//end class CheckBoxUsingItemEvent
```

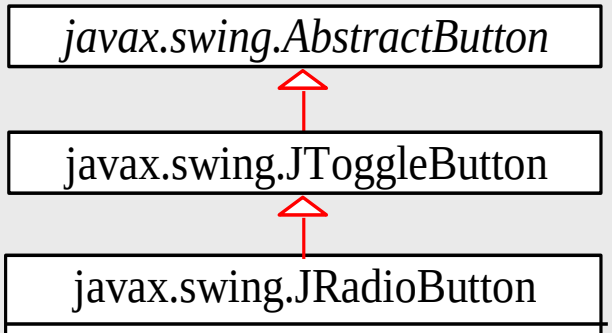
Example – `CheckBoxUsingItemEvent`

This method is the same as in the previous example

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.

MicrowaveOvenFrontPa | CheckBoxDemo.java | CheckBoxUsingItemEve | RadioButtonDemo.java

Example – RadioButtonDemo

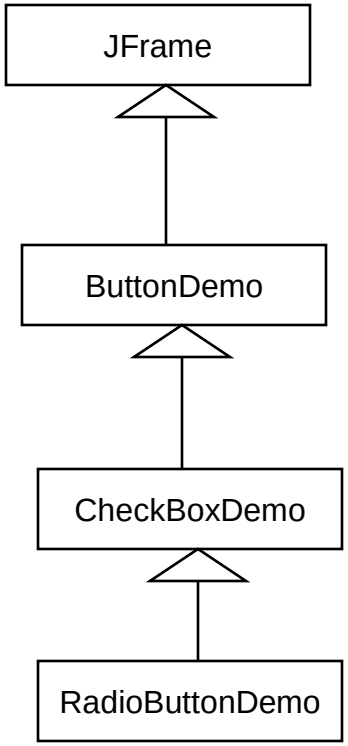
```
//Class: RadioButtonDemo

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.

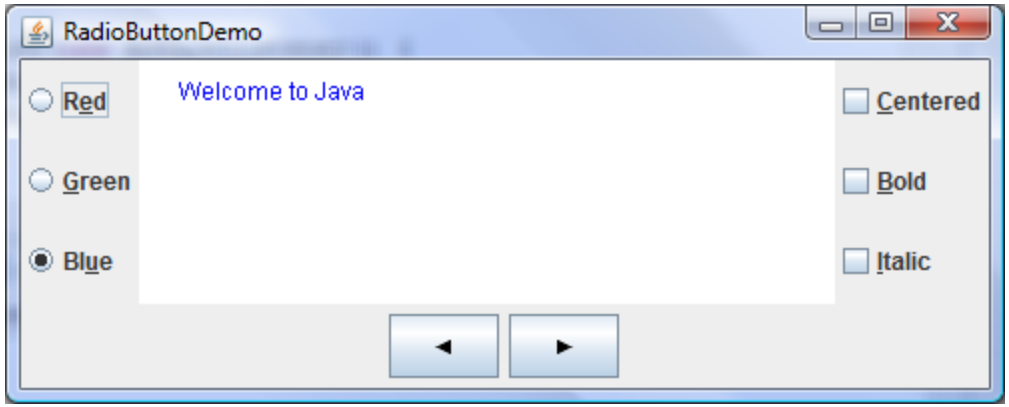
See Note on page 53

Example – RadioButtonDemo

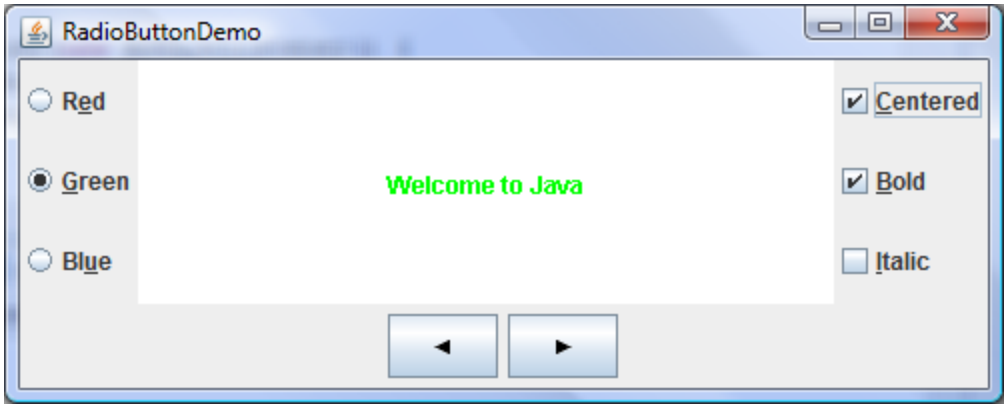
```
    // Set keyboard mnemonics
    jrbRed.setMnemonic('E');
    jrbGreen.setMnemonic('G');
    jrbBlue.setMnemonic('U');

    // 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);
  }
}//end class RadioButtonDemo
```



Example – RadioButtonDemo



Initial GUI



After Radio and Check Buttons set

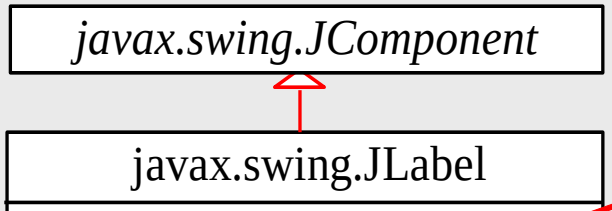
Grouping Radio Buttons

- 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 50.
- 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

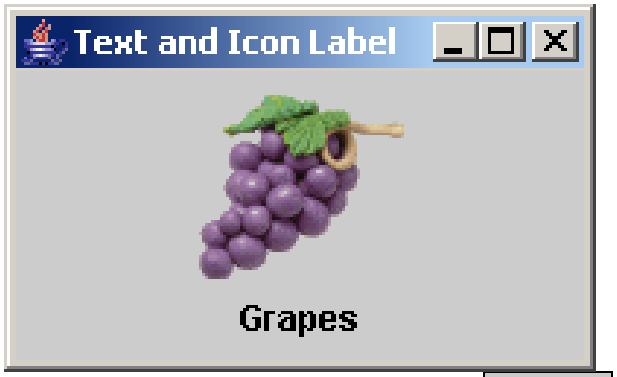


The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

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.

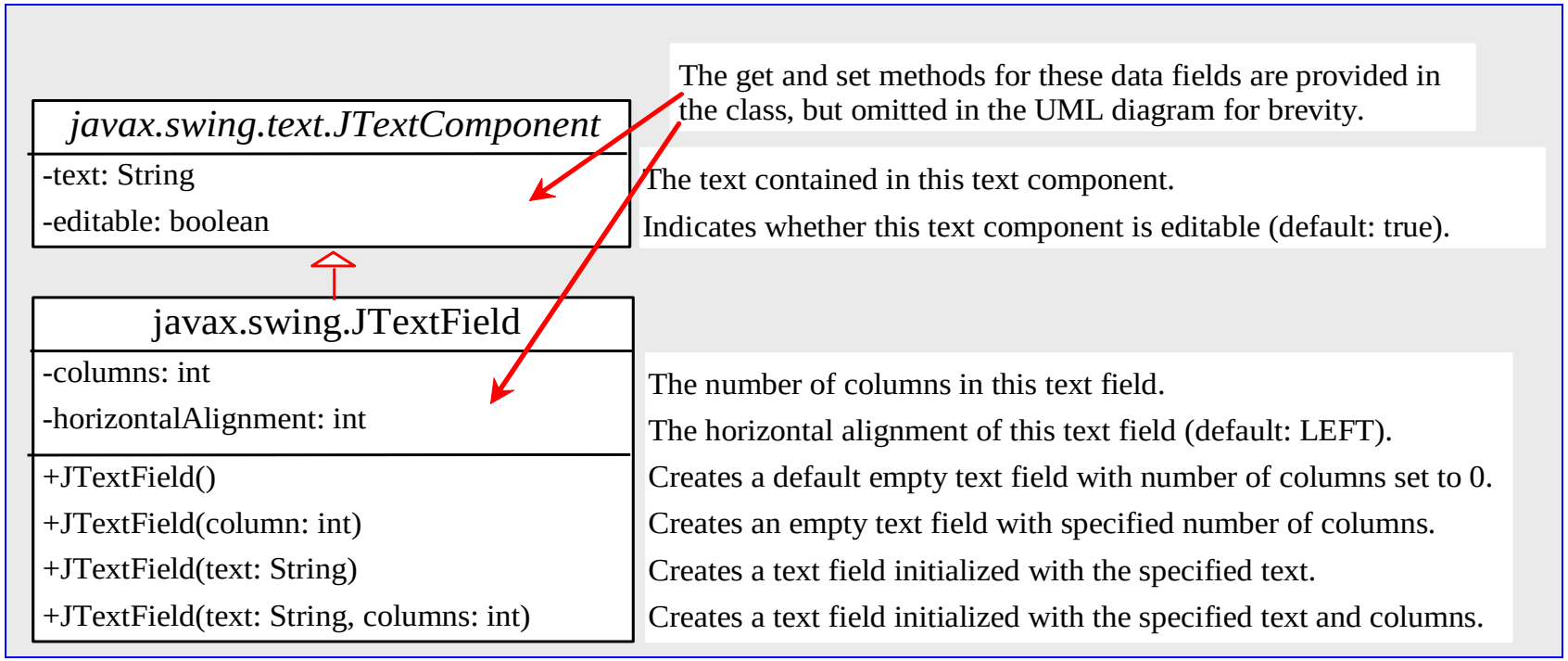
JLabel

```
// Create an image icon from image file
ImageIcon icon = new ImageIcon("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

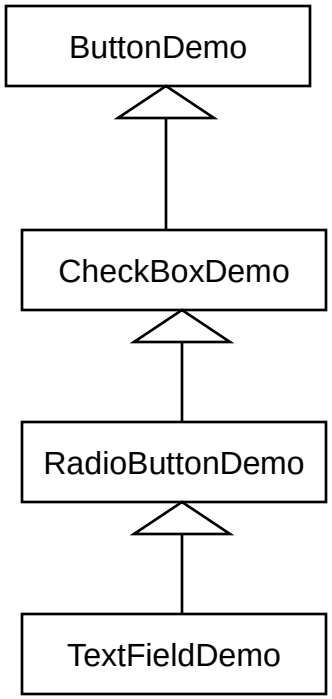
- A text field can be used to enter or display a string. `JTextField` is a subclass of `JTextComponent`.
- The UML for the `JTextField` class is shown below.



CheckBoxDemo.java | CheckBoxUsingItemEve | RadioButtonDemo.java | TextFieldDemo.java

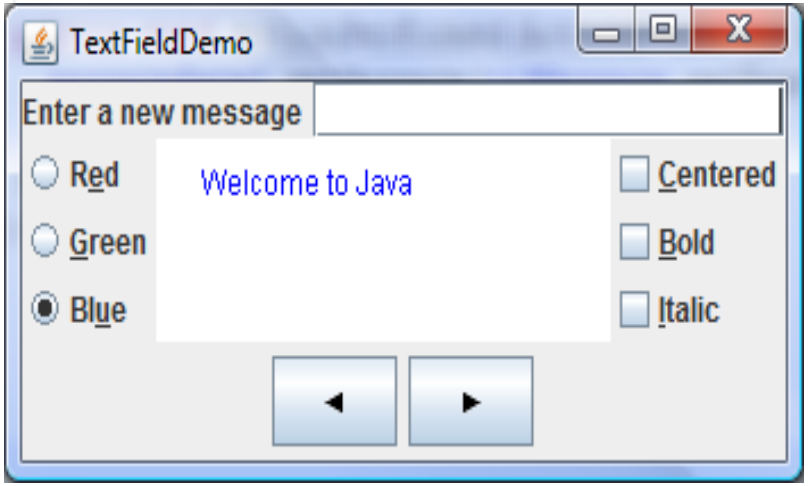
Example – TextFieldDemo

```
//Class: TextFieldDemo
// GUIs Part 3 - COP 3330 - Summer 2011
//MJL 7/6/2011

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);
```

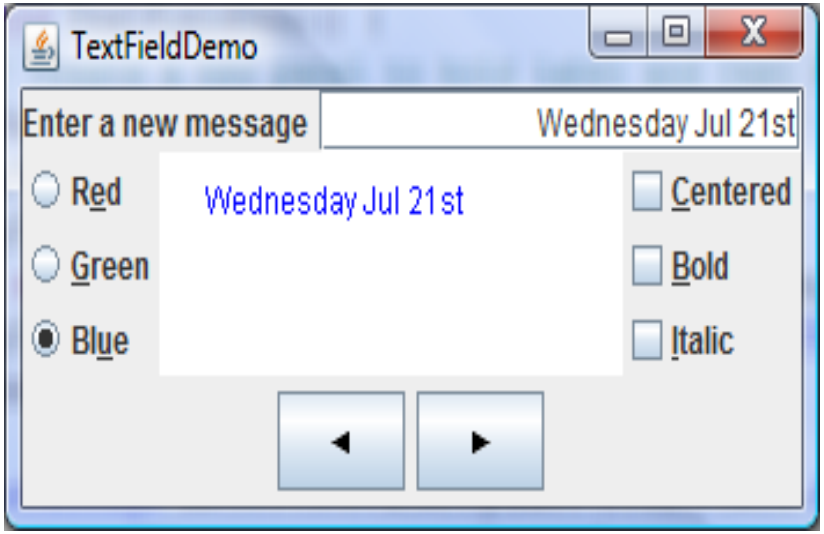


CheckBoxDemo.java | CheckBoxUsingItemEve | RadioButtonDemo.java | TextFieldDemo.java

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

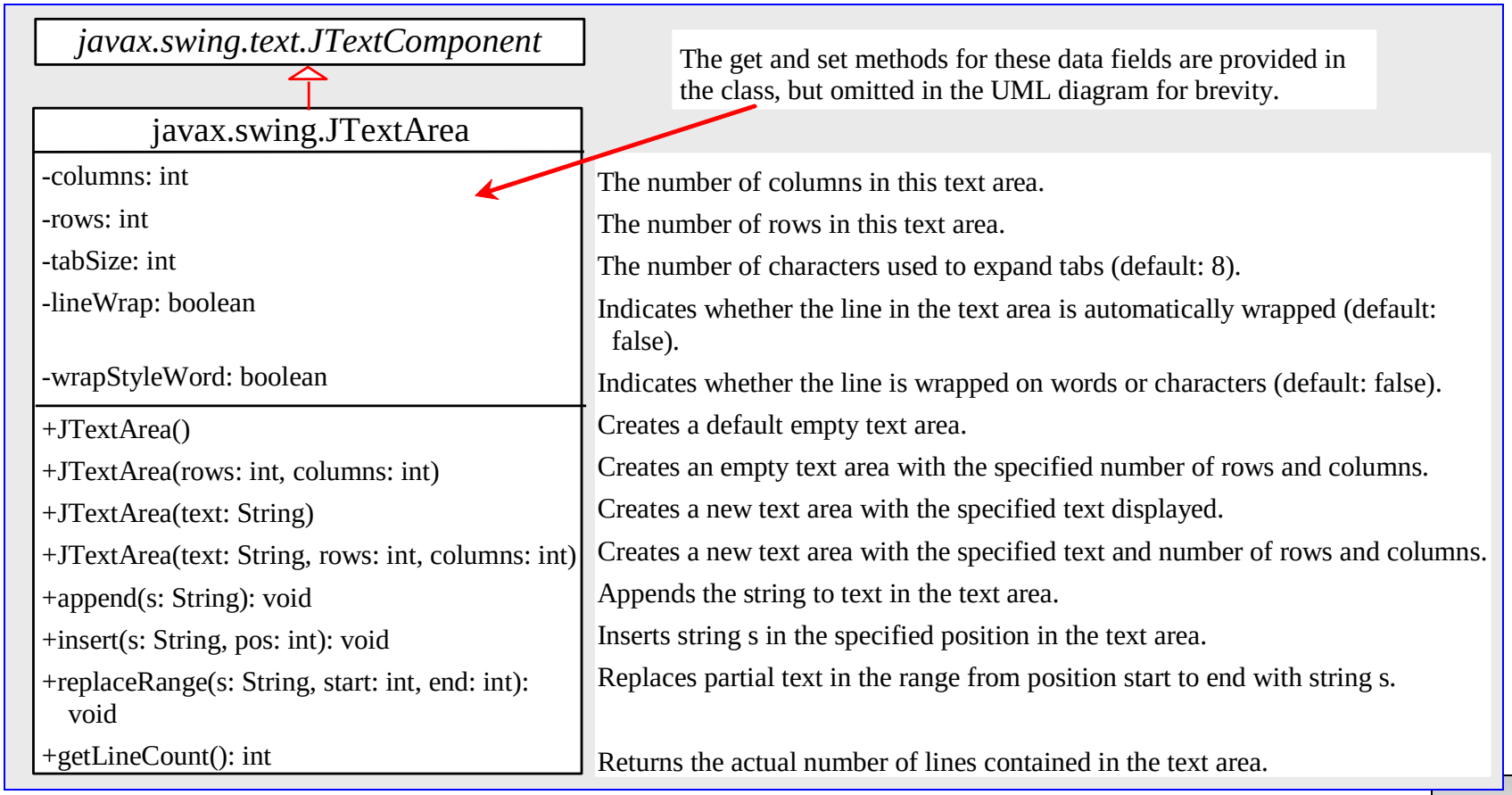
- When you move the cursor into a text field and press the Enter key, it fires an `ActionEvent`.
- In this example program, the `actionPerformed` method sets the new message into `messagePanel`.
- The `pack()` method automatically sizes the frame according to the size of the components placed in it.
- 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.

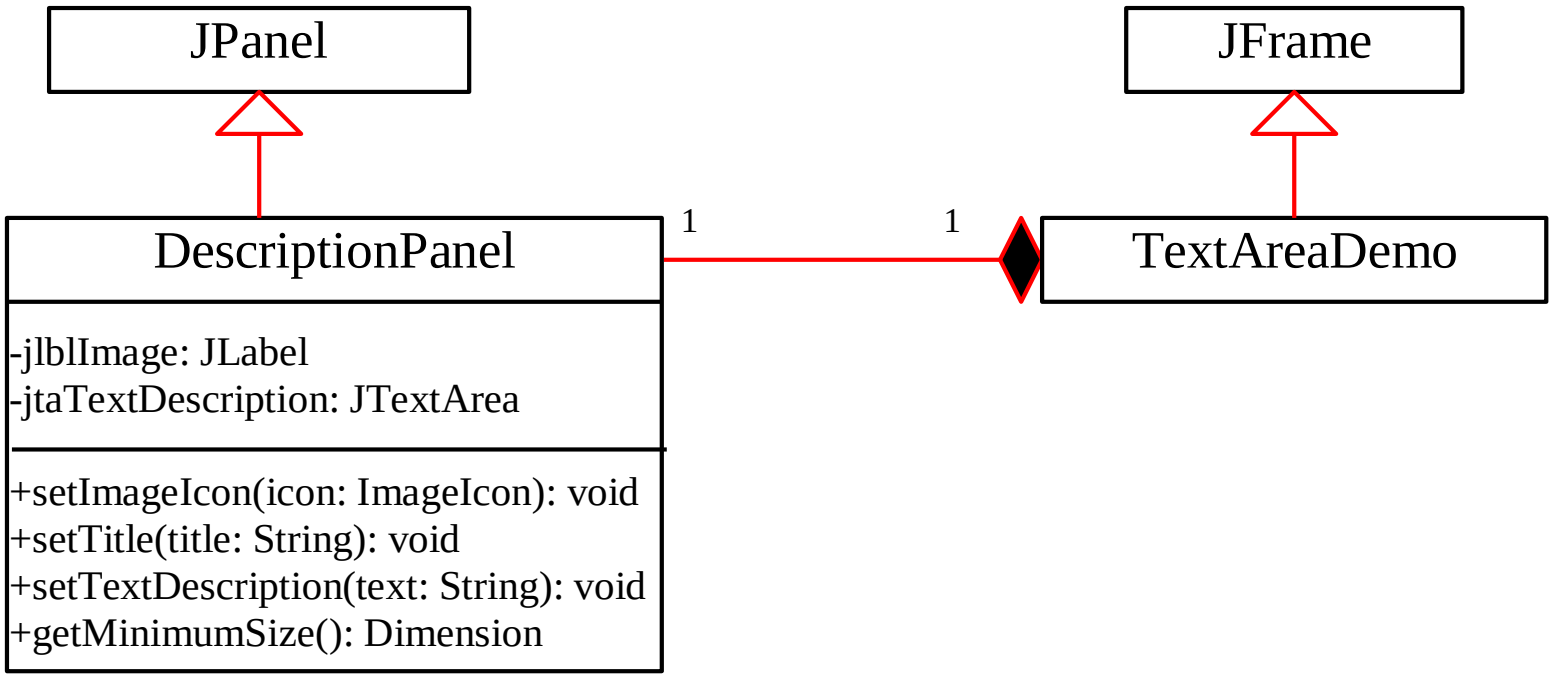
Comments on Text Fields

- 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 `JTextArea` class is shown below.




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

CheckBoxDemo.java | RadioButtonDemo.java | TextFieldDemo.java | *DescriptionPanel.ja

```
//Class: DescriptionPanel
//GUIs - Part 3 - COP 3330 - Summer 2011
//MJL 7/6/2011

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);
```


Example – `DescriptionPanel` Class

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

```
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("C:/Courses/COP 3330 - OOP/Summer 2011/image
    // Add the description panel to the frame
    setLayout(new BorderLayout());
    add(descriptionPanel, BorderLayout.CENTER);
  }
}
```


Example – TextAreaDemo

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.

GUI after re-sizing – notice absence of slider bar