CNT 4714: Enterprise Computing Spring 2010

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/cnt4714/spr2010

School of Electrical Engineering and Computer Science
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 –

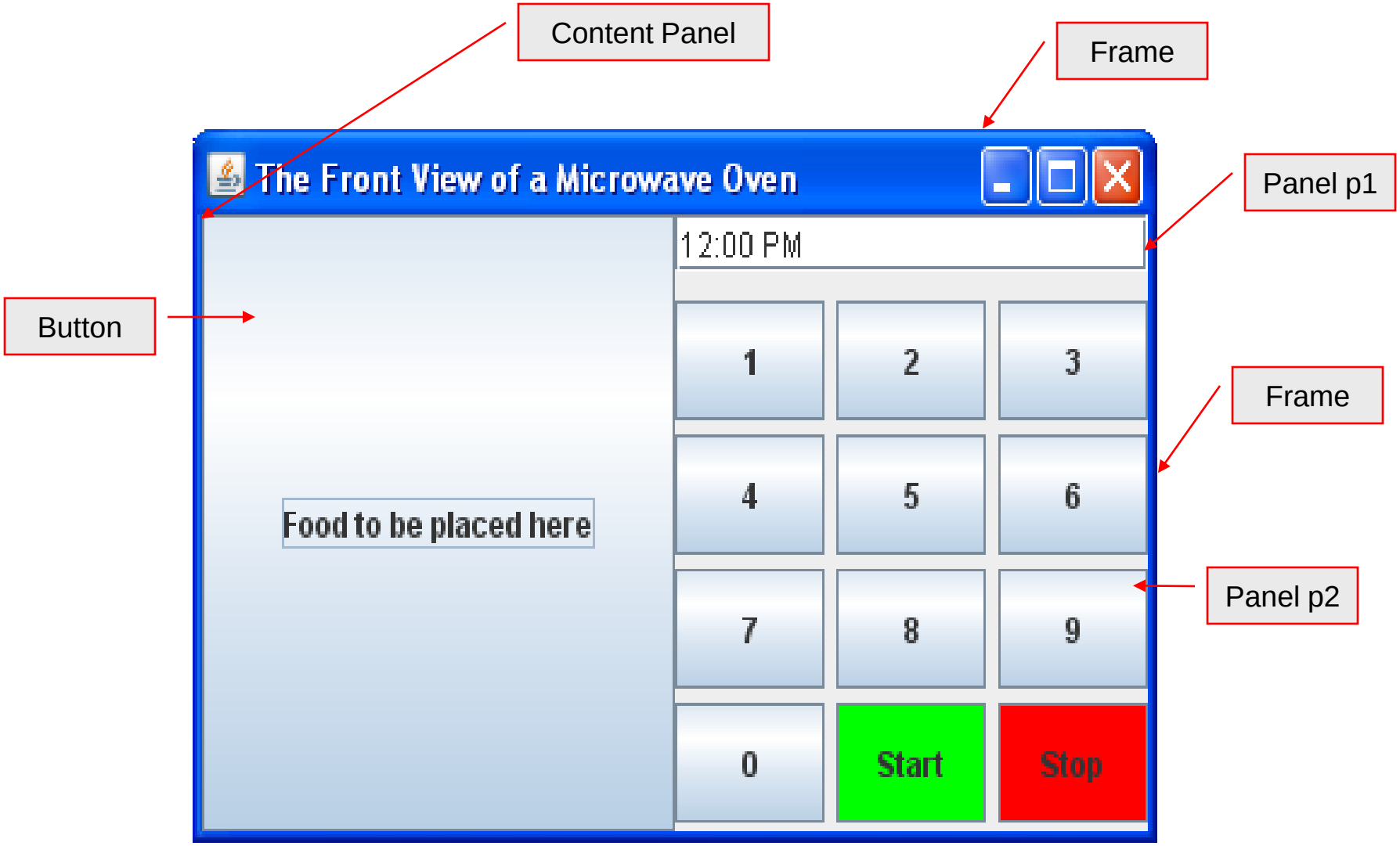
  `new JPanel(LayoutManager)` to create a panel with the specified layout manager.
- The following example illustrates using panels as subcontainers.

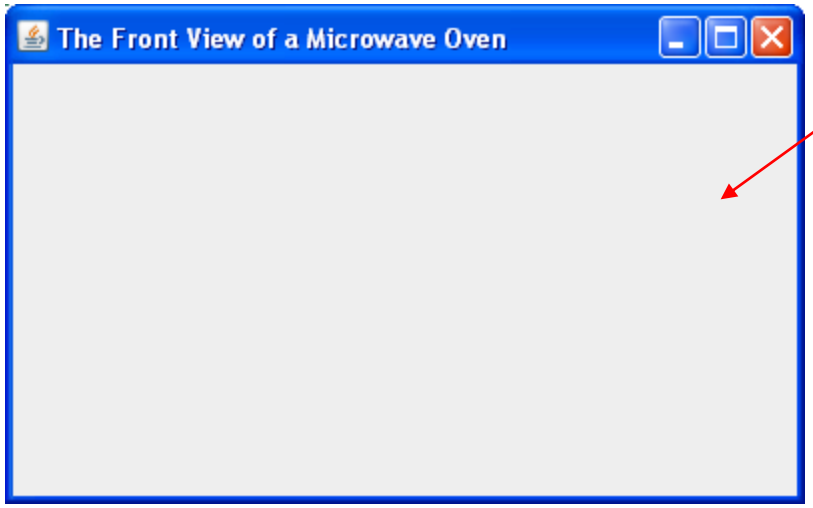
Example – Using Panels

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

public class TestPanels extends JFrame {
  public TestPanels() {
    // 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));
    }

    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"),
      BorderLayout.NORTH);
    p2.add(p1, BorderLayout.CENTER);
```

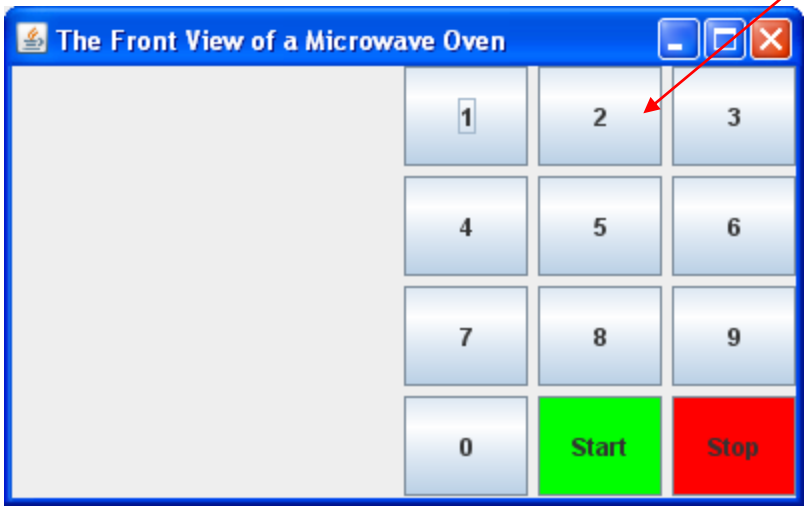

```
// add contents into the frame
    add(p2, BorderLayout.EAST);
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }

  /** Main method */
  public static void main(String[] args) {
    TestPanels frame = new TestPanels();
    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);
  }
}
```

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.

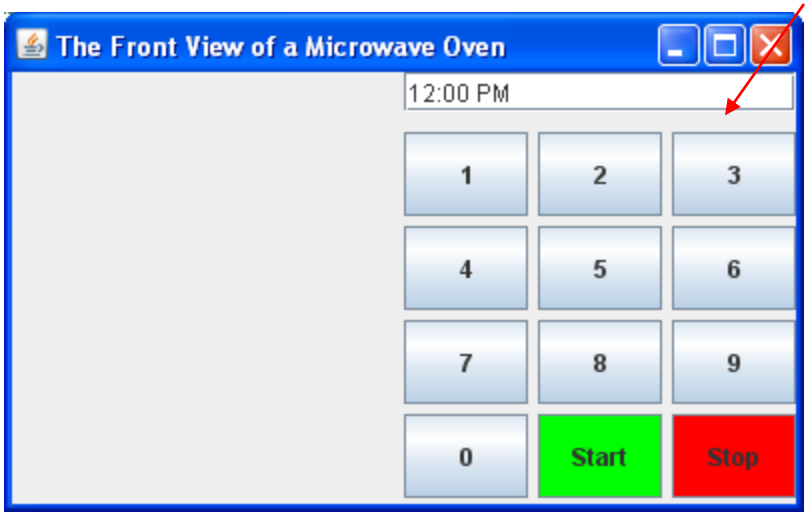

Content Panel
Frame
The Front View of a Microwave Oven
Panel p1
12:00 PM
Button
1
2
3
Frame
Food to be placed here
4
5
6
Panel p2
7
8
9
0
Start
Stop



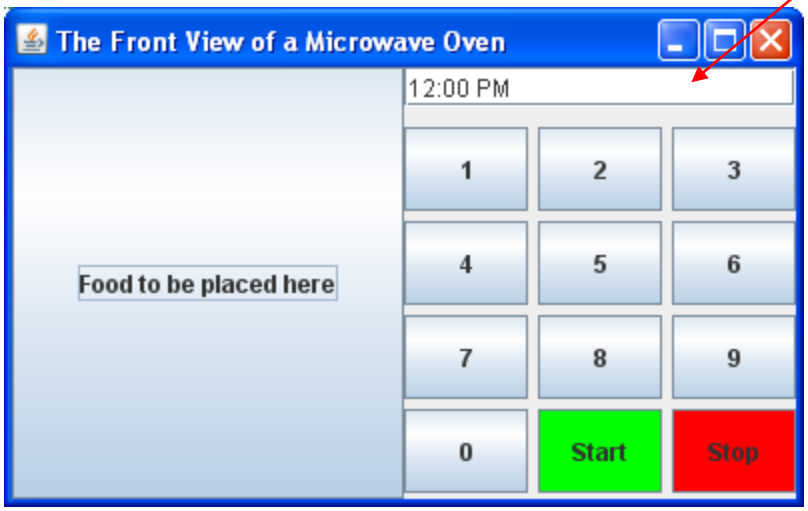
Initial frame – no components added yet.



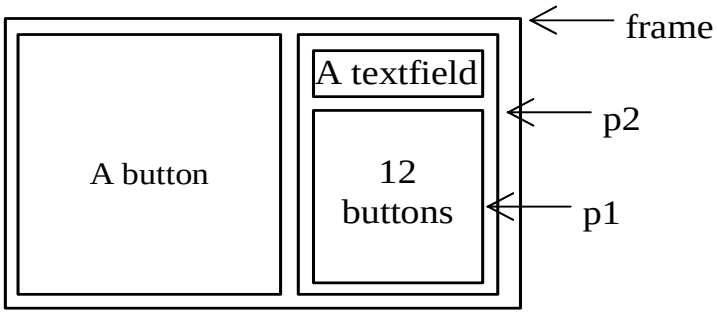
Showing just panel p1 added to the frame.



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.



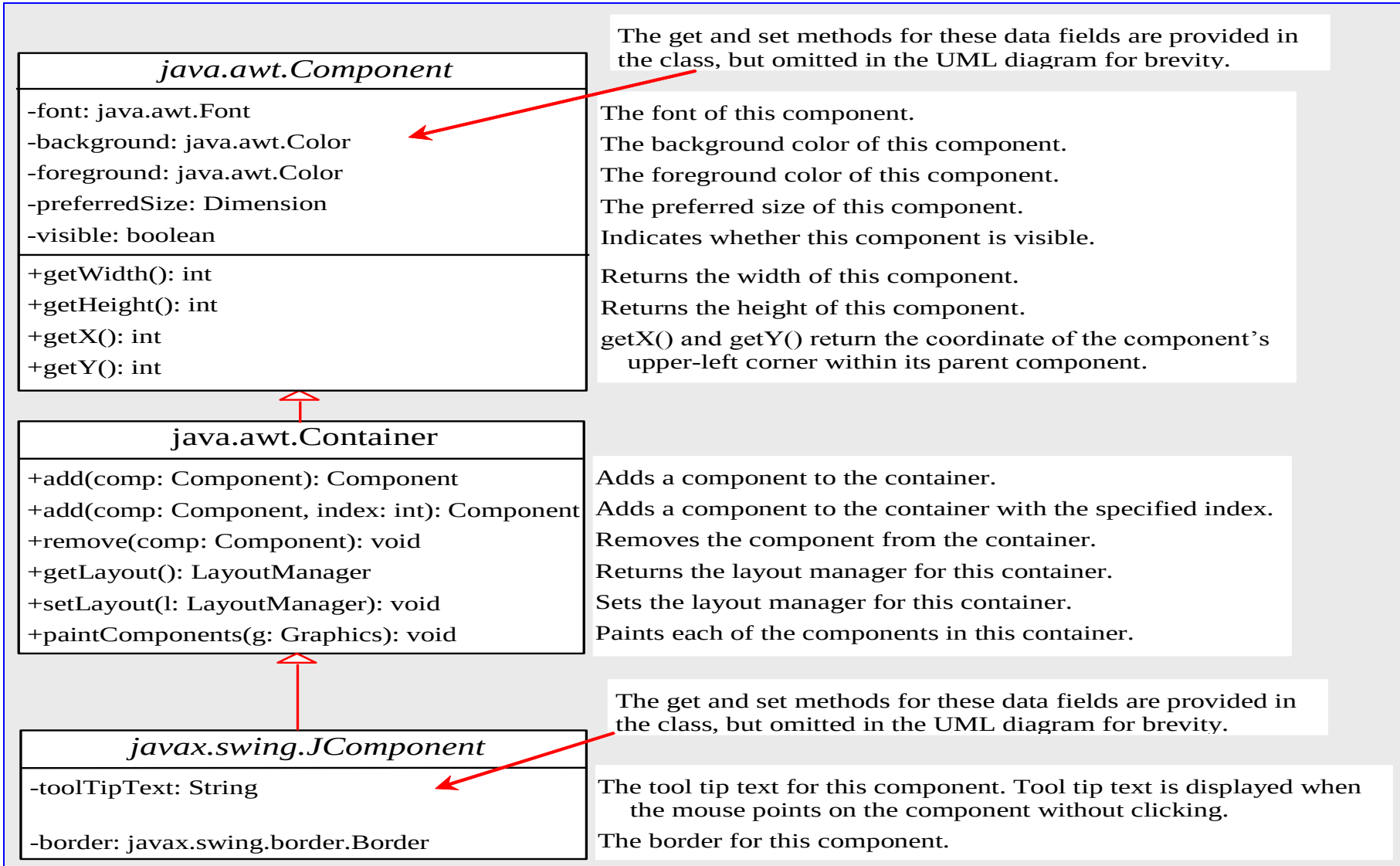
Showing final frame using `BorderLayout`. Added a `JButton` ("Food to be placed here") to the Center area. Added panel p2 to the East area.



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)
    ```

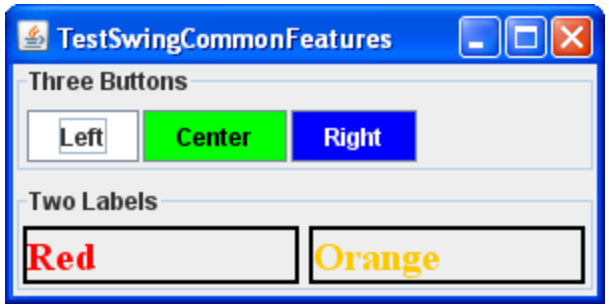
  - To create a line border use:

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

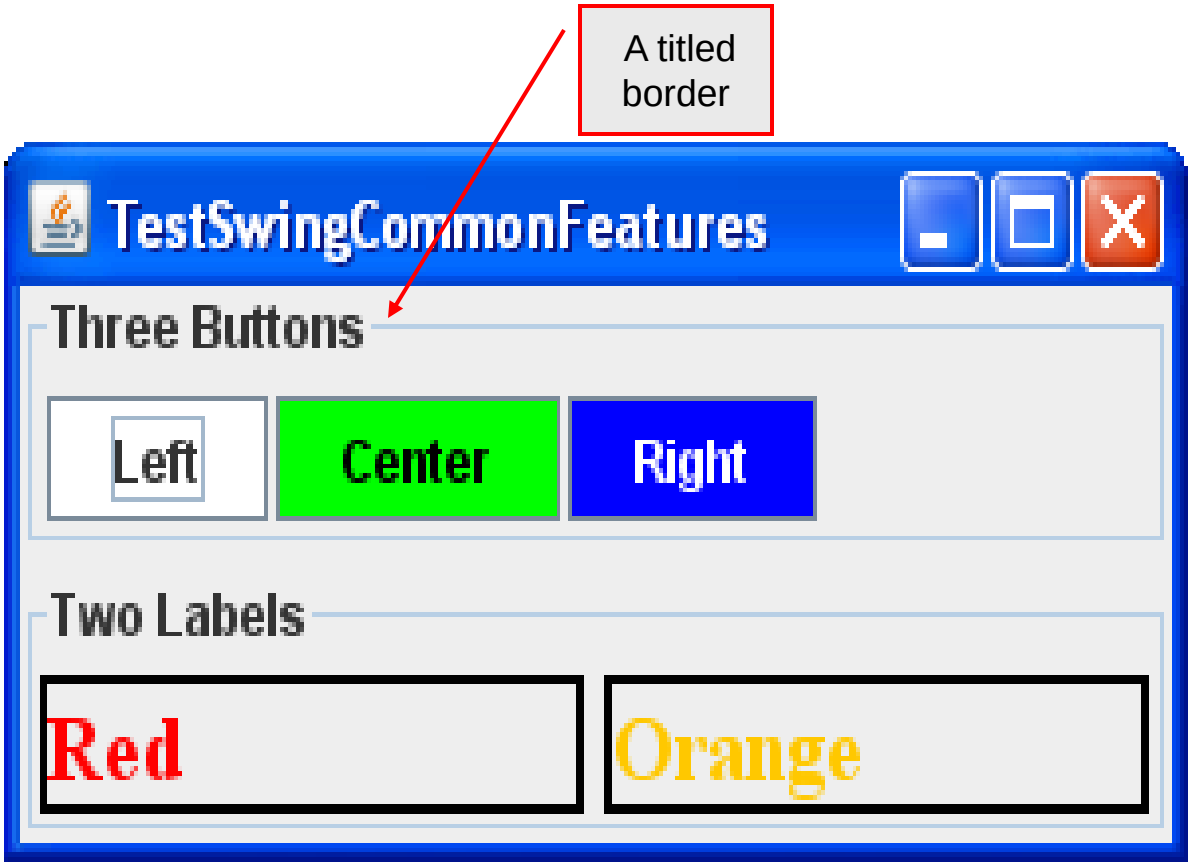
    where width specifies the thickness of the line in pixels.
- The following example illustrates some of the common Swing features.

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

public class TestSwingCommonFeatures extends JFrame {
  public TestSwingCommonFeatures() {
    // Create a panel to group three buttons
    JPanel p1 = new JPanel(new FlowLayout(FlowLayout.LEFT, 2, 2));
    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"));

    // Create a font and a line border
    Font largeFont = new Font("TimesRoman", Font.BOLD, 20);
    Border lineBorder = new LineBorder(Color.BLACK, 2);
```

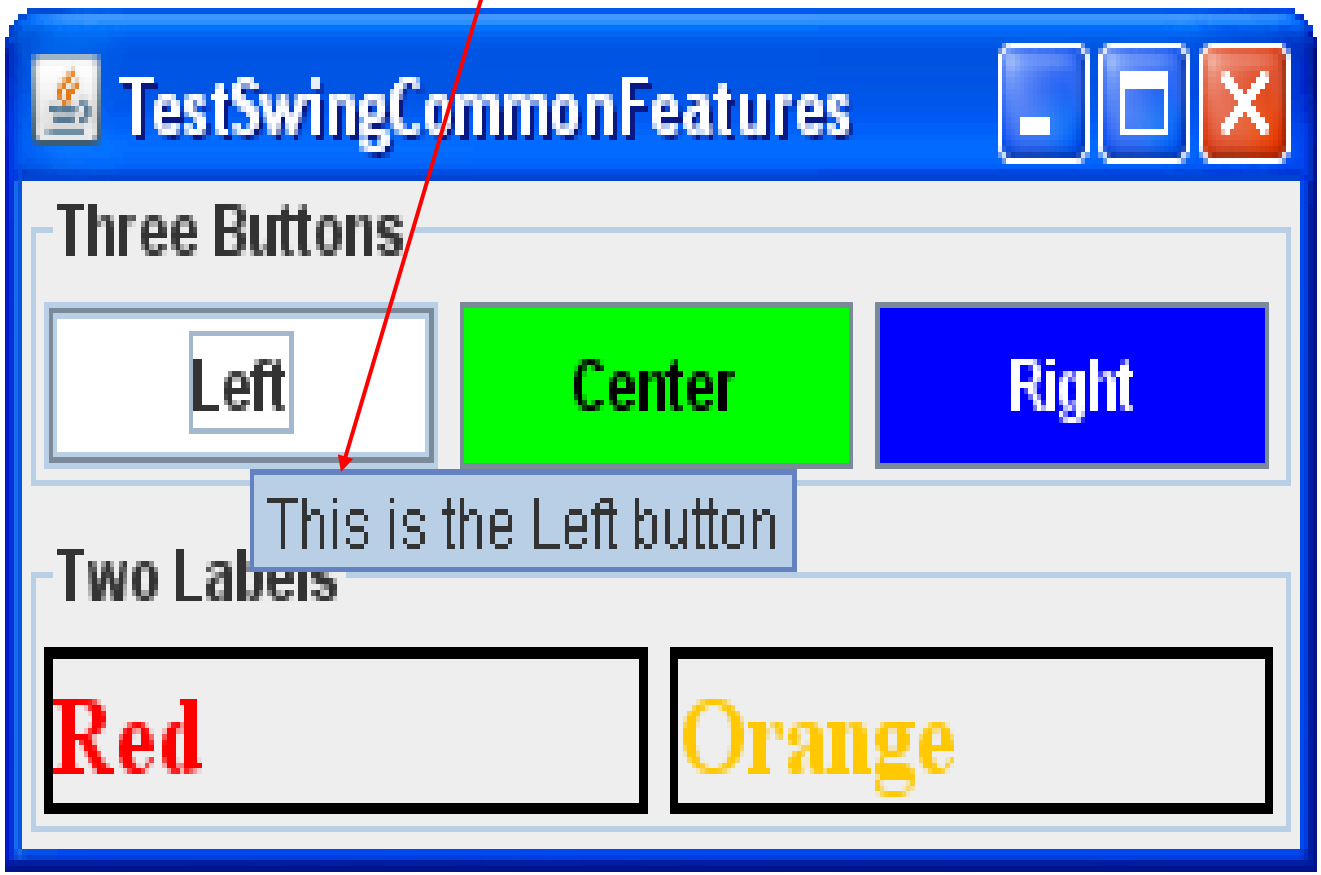

```
    // Create a panel to group two labels
    JPanel p2 = new JPanel(new GridLayout(1, 2, 5, 5));
    JLabel jlblRed = new JLabel("Red");
    JLabel jlblOrange = new JLabel("Orange");
    jlblRed.setForeground(Color.RED);
    jlblOrange.setForeground(Color.ORANGE);
    jlblRed.setFont(largeFont);
    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);
  }

  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);
  }
}
```



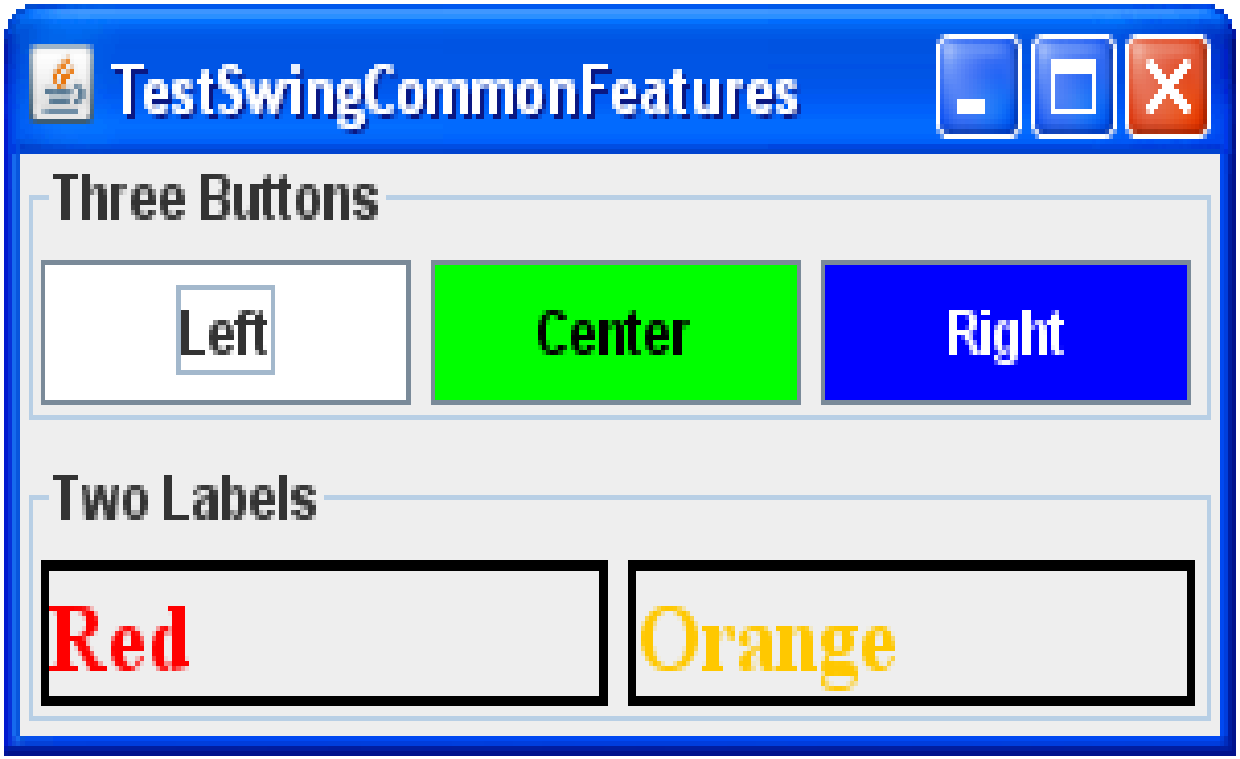
Insertion point for statements shown on page 16.


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?



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

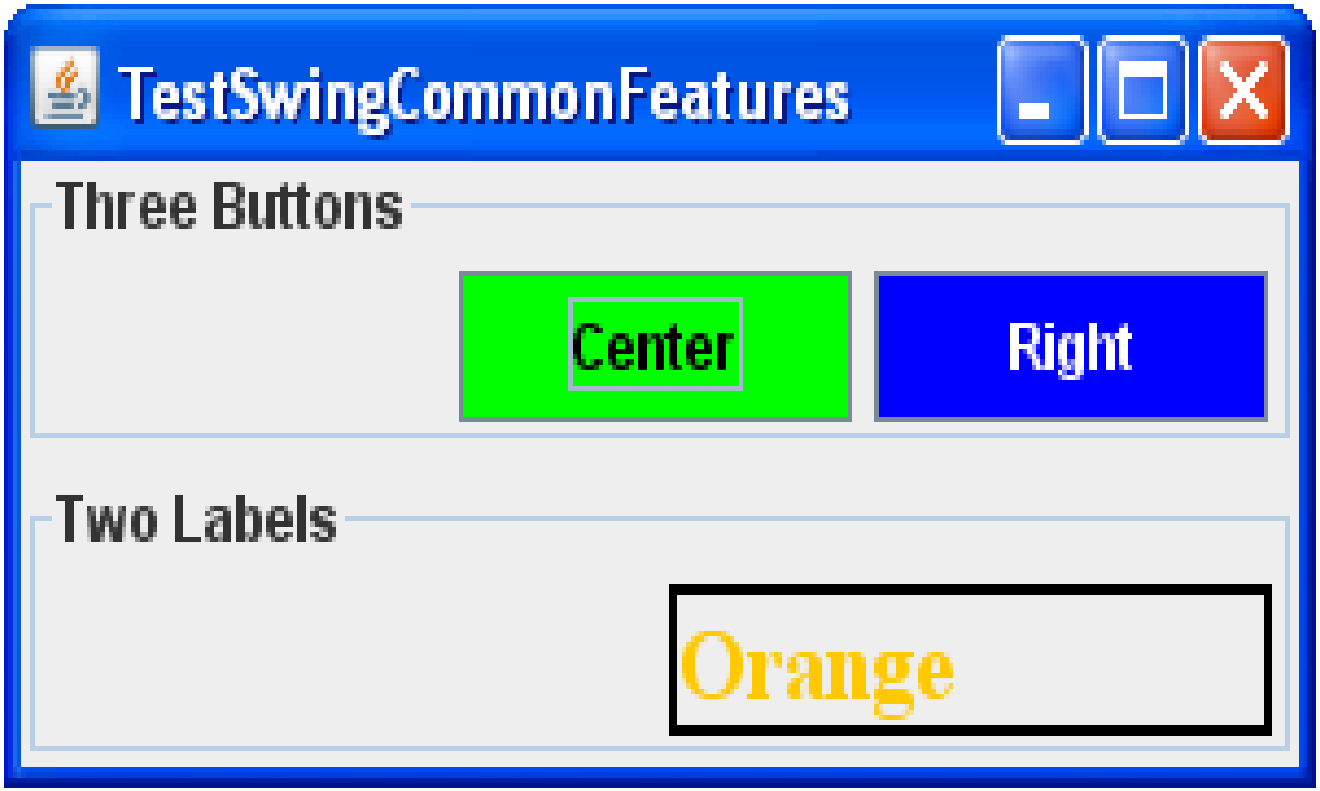


Image Icons

- 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 an absolute path name to the image files.

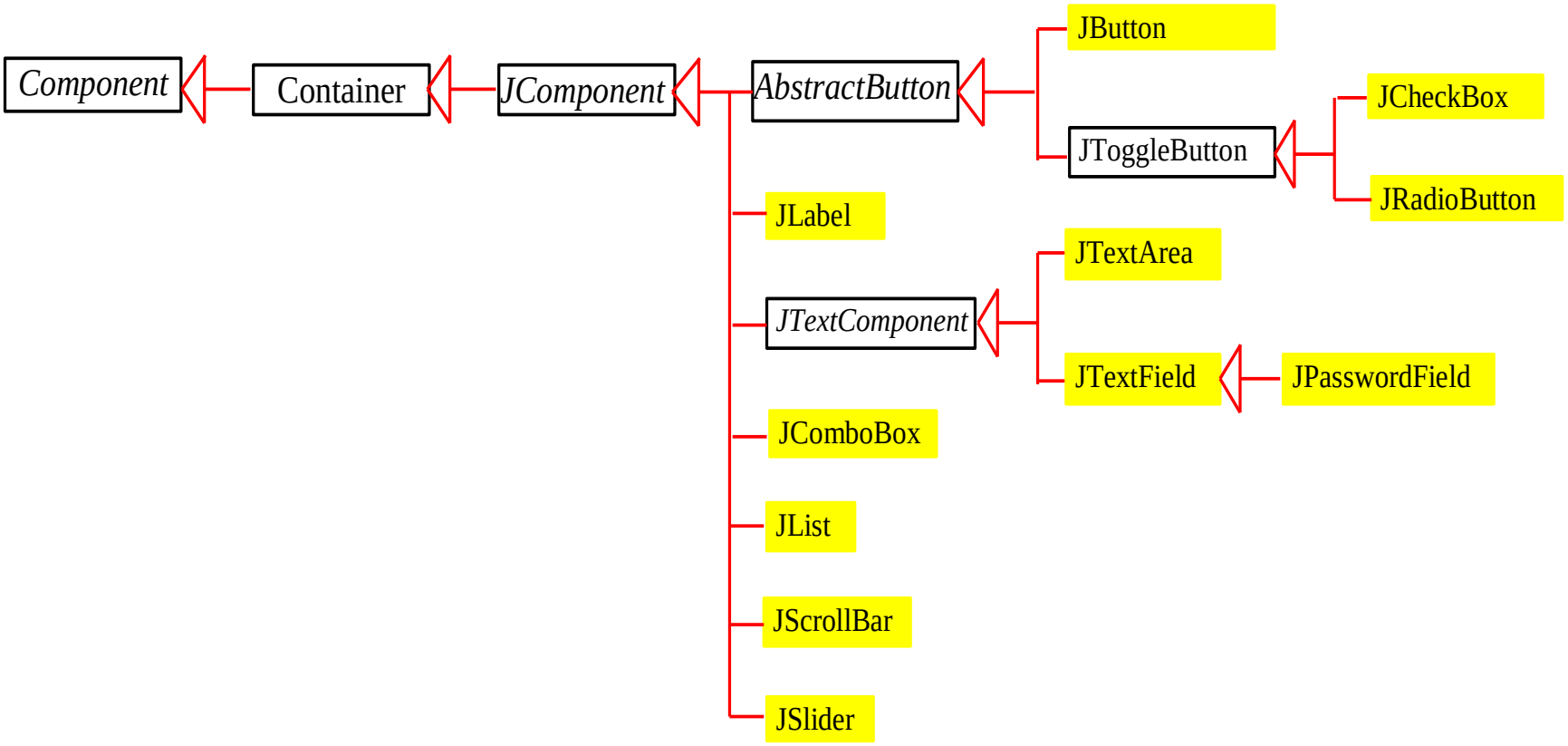
Example – Using Image Icons

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

public class TestImageIcon extends JFrame {
  private ImageIcon usIcon = new ImageIcon("E:/image/us.gif");
  private ImageIcon myIcon = new ImageIcon("E:/image/sw-t.jpg");
  private ImageIcon frIcon = new ImageIcon("E:/image/fr.gif");
  private ImageIcon ukIcon = new ImageIcon("E:/image/uk.gif");

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


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

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.

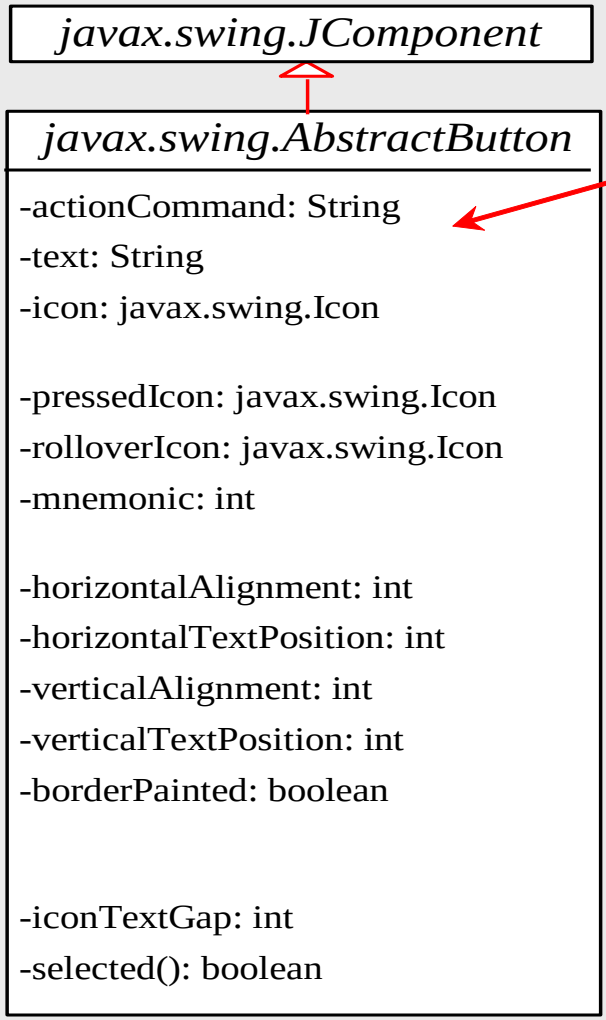
Commonly Used GUI Components



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.

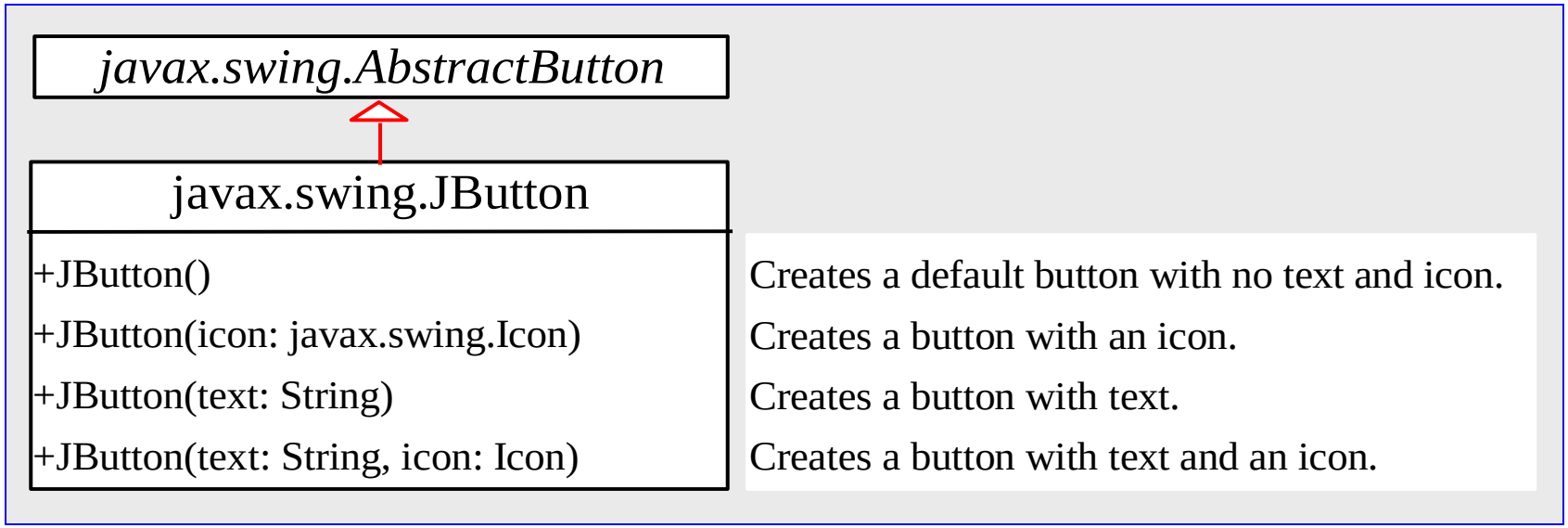
javax.swing.AbstractButton



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

```
import javax.swing.*;
public class TestButtonIcons extends JFrame  {
  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);
  }

  public TestButtonIcons() {
    ImageIcon usIcon = new ImageIcon("E:/image/usIcon.gif");
    ImageIcon caIcon = new ImageIcon("E:/image/caIcon.gif");
    ImageIcon ukIcon = new ImageIcon("E:/image/ukIcon.gif");

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

    add(jbt);
  }
}
```

default icon

rollover icon

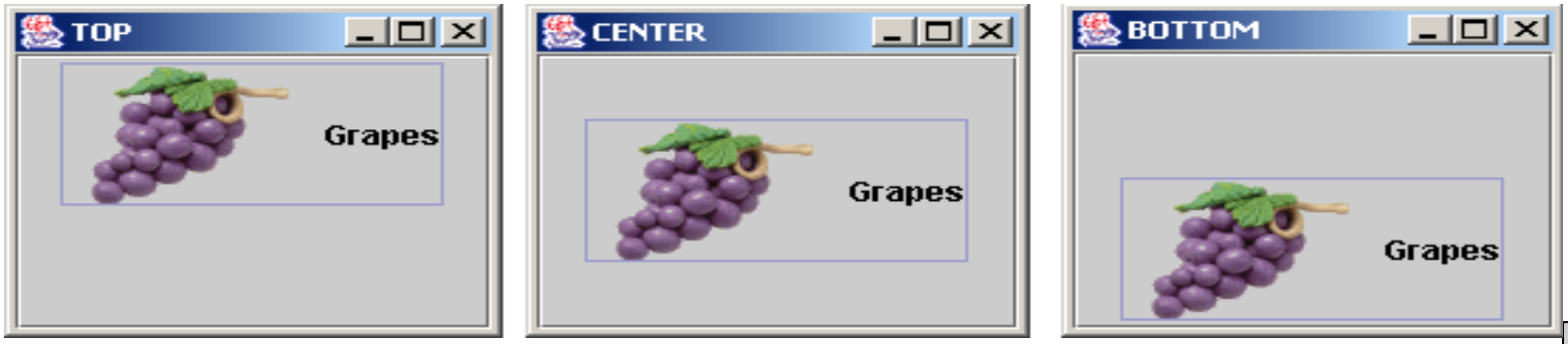
pressed icon

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`.

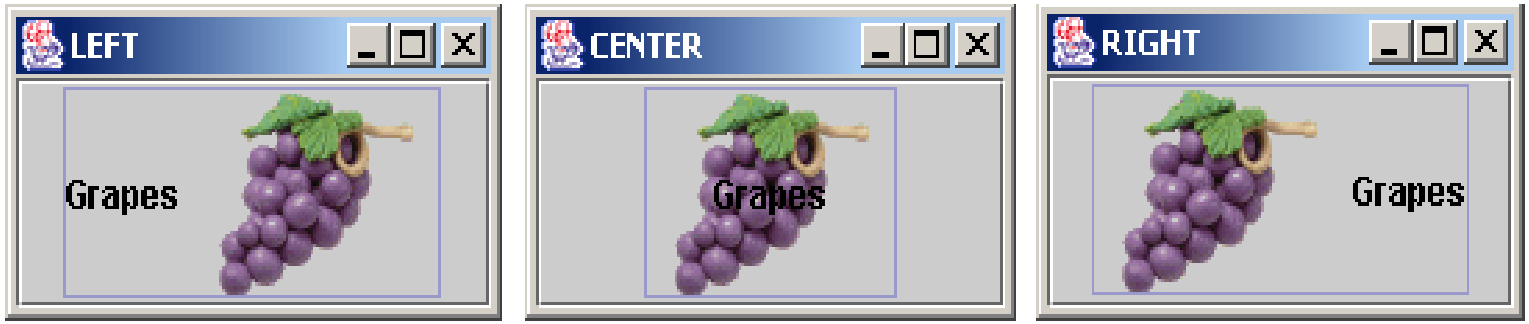
Alignments

- 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`.



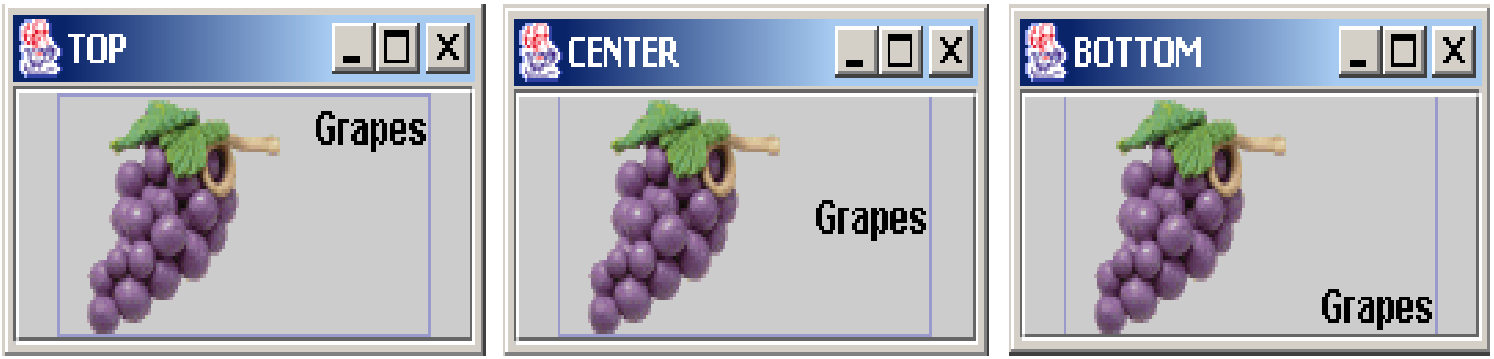
Text Positions

- 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`.

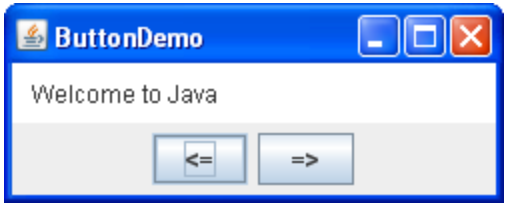
Using Buttons

- 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



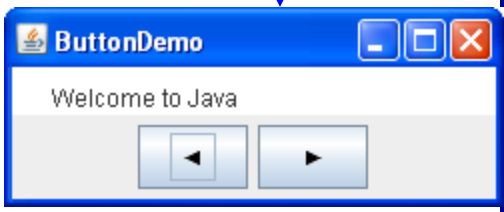
Frame after user has clicked the left button a few times

Frame after user has clicked the right button a few times

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

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);
  }
```


```
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);

    // 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.



```
// 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()`