Lecture 6: GUI Basics (Ch 12)

Adapted by Fangzhen Lin for COMP3021 from Y. Danial Liang's PowerPoints for Introduction to Java Programming, Comprehensive Version, 9/E, Pearson, 2013.

Motivations

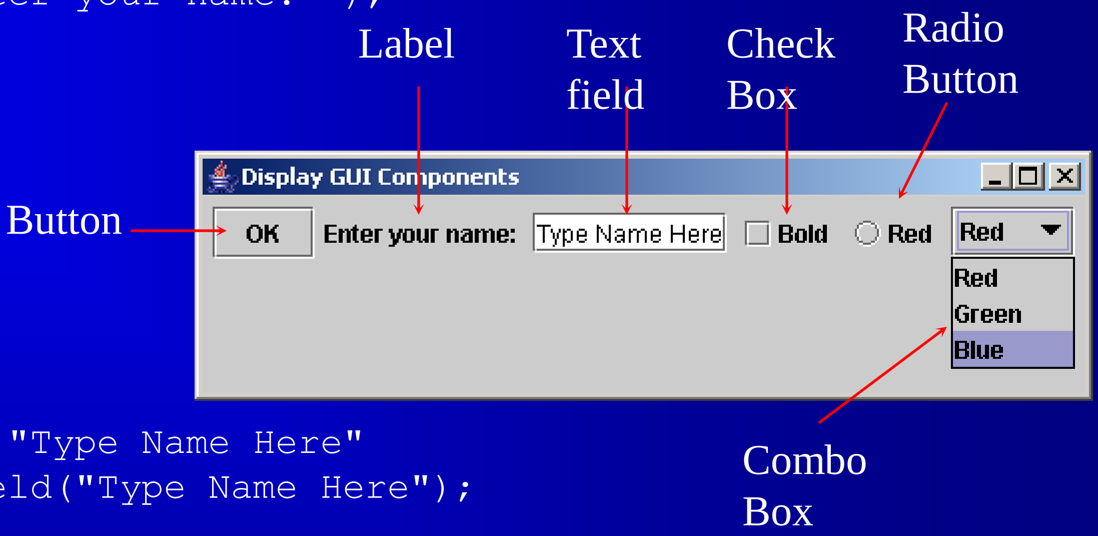
The design of the API for Java GUI programming is an excellent example of how the object-oriented principle is applied. Java GUI API and components are used to develop user-friendly interfaces for applications and applets.

Objectives

- To distinguish between Swing and AWT (§ 12.2).
- To describe the Java GUI API hierarchy (§ 12.3).
- To create user interfaces using frames, panels, and simple GUI components (§ 12.4).
- To understand the role of layout managers and use the **FlowLayout**, **GridLayout**, and **BorderLayout** managers to lay out components in a container (§ 12.5).
- To use **JPanel** to group components in a subcontainer (§ 12.6).
- To create objects for colors using the **Color** class (§ 12.7).
- To create objects for fonts using the **Font** class (§ 12.8).
- To apply common features such as borders, tool tips, fonts, and colors on Swing components (§ 12.9).
- To decorate the border of GUI components (§ 12.9).
- To create image icons using the **ImageIcon** class (§ 12.10).
- To create and use buttons using the **JButton** class (§ 12.11).
- To create and use check boxes using the **JCheckBox** class (§ 12.12).
- To create and use radio buttons using the **JRadioButton** class (§ 12.13).
- To create and use labels using the **JLabel** class (§ 12.14).
- To create and use text fields using the **JTextField** class (§ 12.15).

Creating GUI Objects

```
// Create a button with text OK
JButton jbtOK = new JButton("OK");

// Create a label with text "Enter your name: "
JLabel jlblName = new JLabel("Enter your name: ");
```



```
// Create a text field with text "Type Name Here"
JTextField jtfName = new JTextField("Type Name Here");

// Create a check box with text bold
JCheckBox jchkBold = new JCheckBox("Bold");

// Create a radio button with text red
JRadioButton jrbRed = new JRadioButton("Red");

// Create a combo box with choices red, green, and blue
JComboBox jcboColor = new JComboBox(new String[]{"Red",
  "Green", "Blue"});
```

Swing vs. AWT

So why do the GUI component classes have a prefix *J*? Instead of JButton, why not name it simply Button? In fact, there is a class already named Button in the java.awt package.

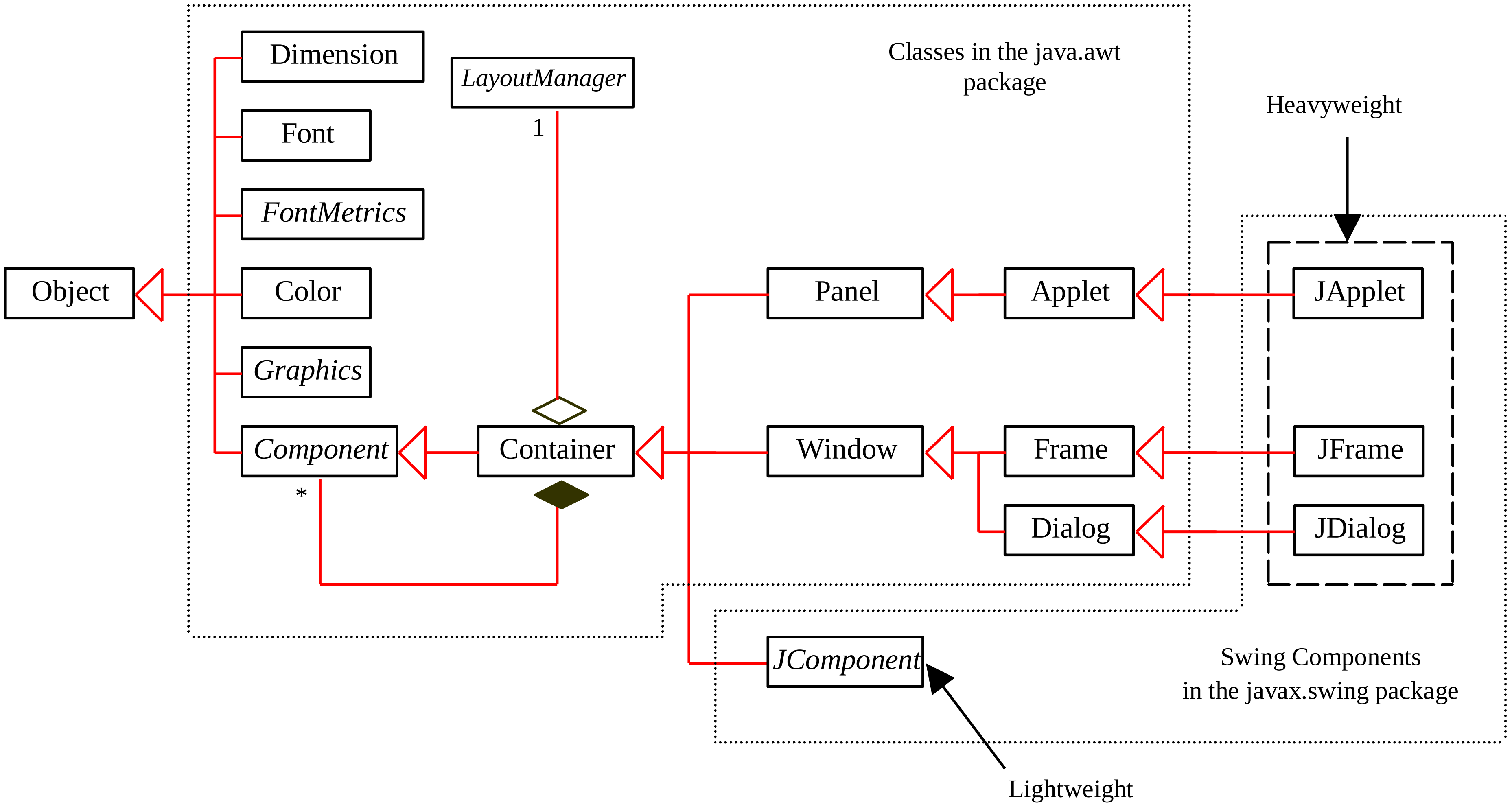
When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT). For every platform on which Java runs, the AWT components are automatically mapped to the platform-specific components through their respective agents, known as *peers*. AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects. Besides, AWT is prone to platform-specific bugs because its peer-based approach relies heavily on the underlying platform. With the release of Java 2, the AWT user-interface components were replaced by a more robust, versatile, and flexible library known as *Swing components*. Swing components are painted directly on canvases using Java code, except for components that are subclasses of java.awt.Window or java.awt.Panel, which must be drawn using native GUI on a specific platform. Swing components are less dependent on the target platform and use less of the native GUI resource. For this reason, Swing components that don’t rely on native GUI are referred to as *lightweight components*, and AWT components are referred to as *heavyweight components*.

The Java GUI API

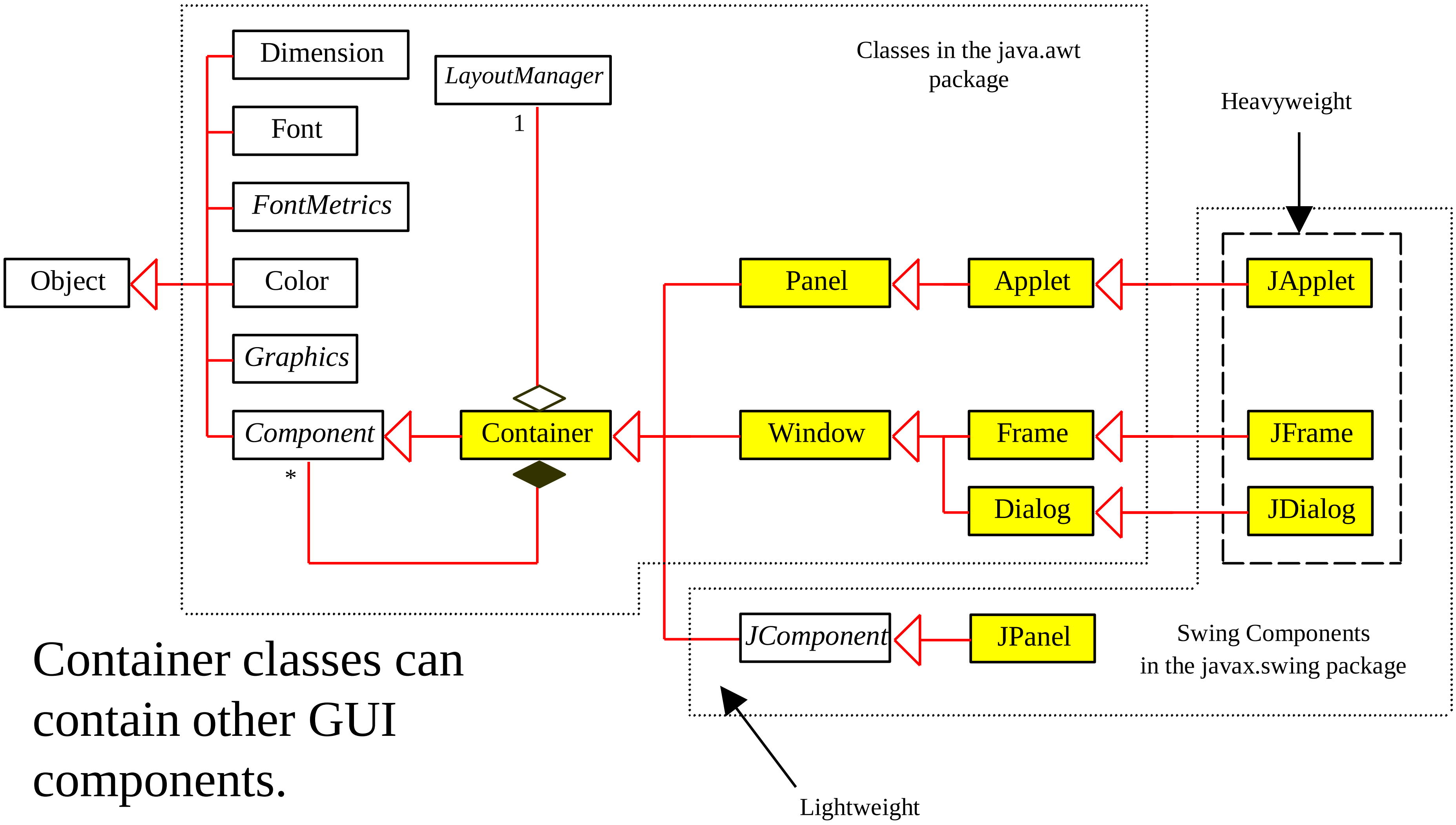
The GUI API contains classes that can be classified into three groups:

- Component classes for creating user interface.
- Container classes for grouping components.
- Helper classes for supporting component classes

GUI Class Hierarchy (Swing)

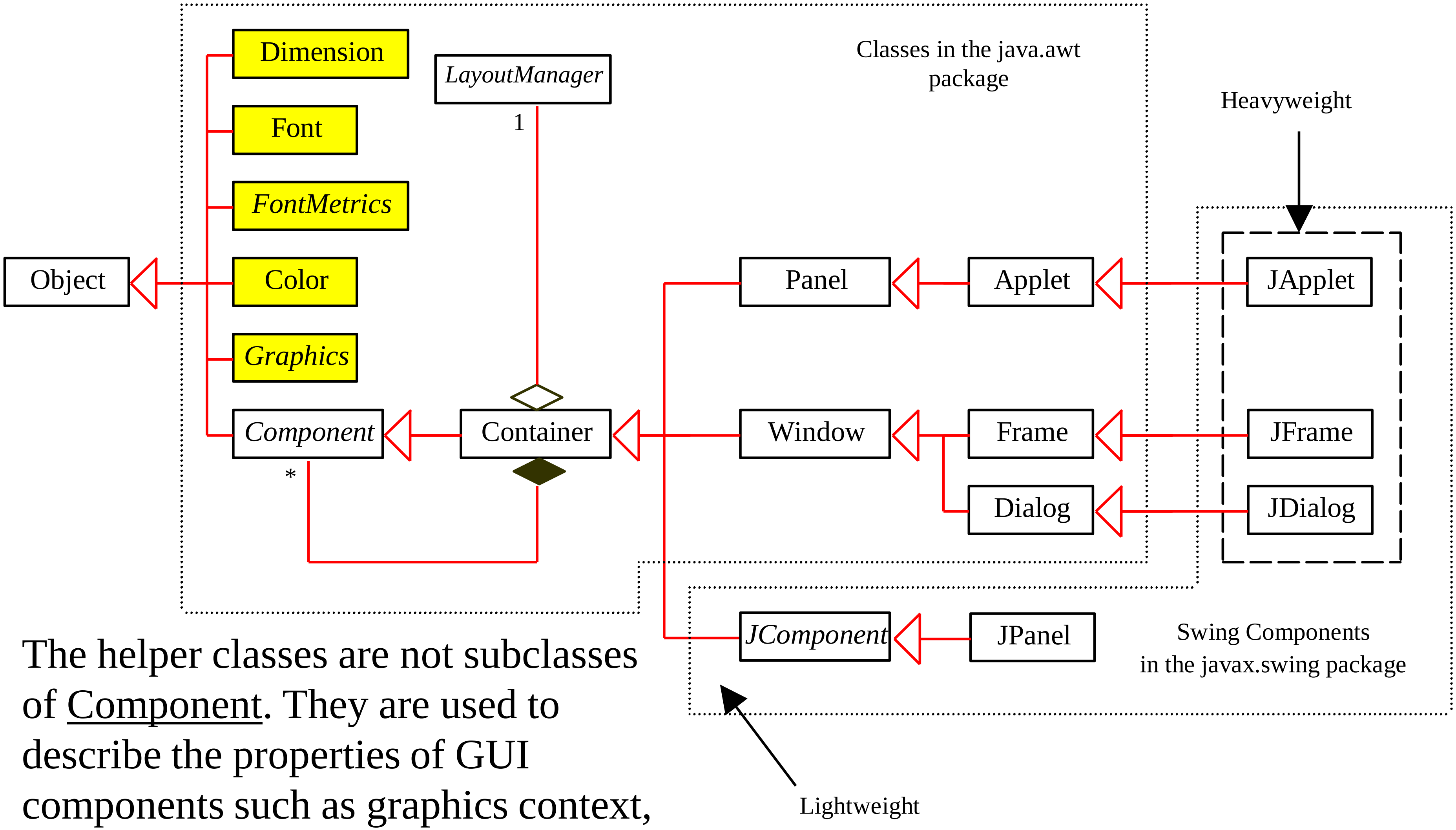


Container Classes



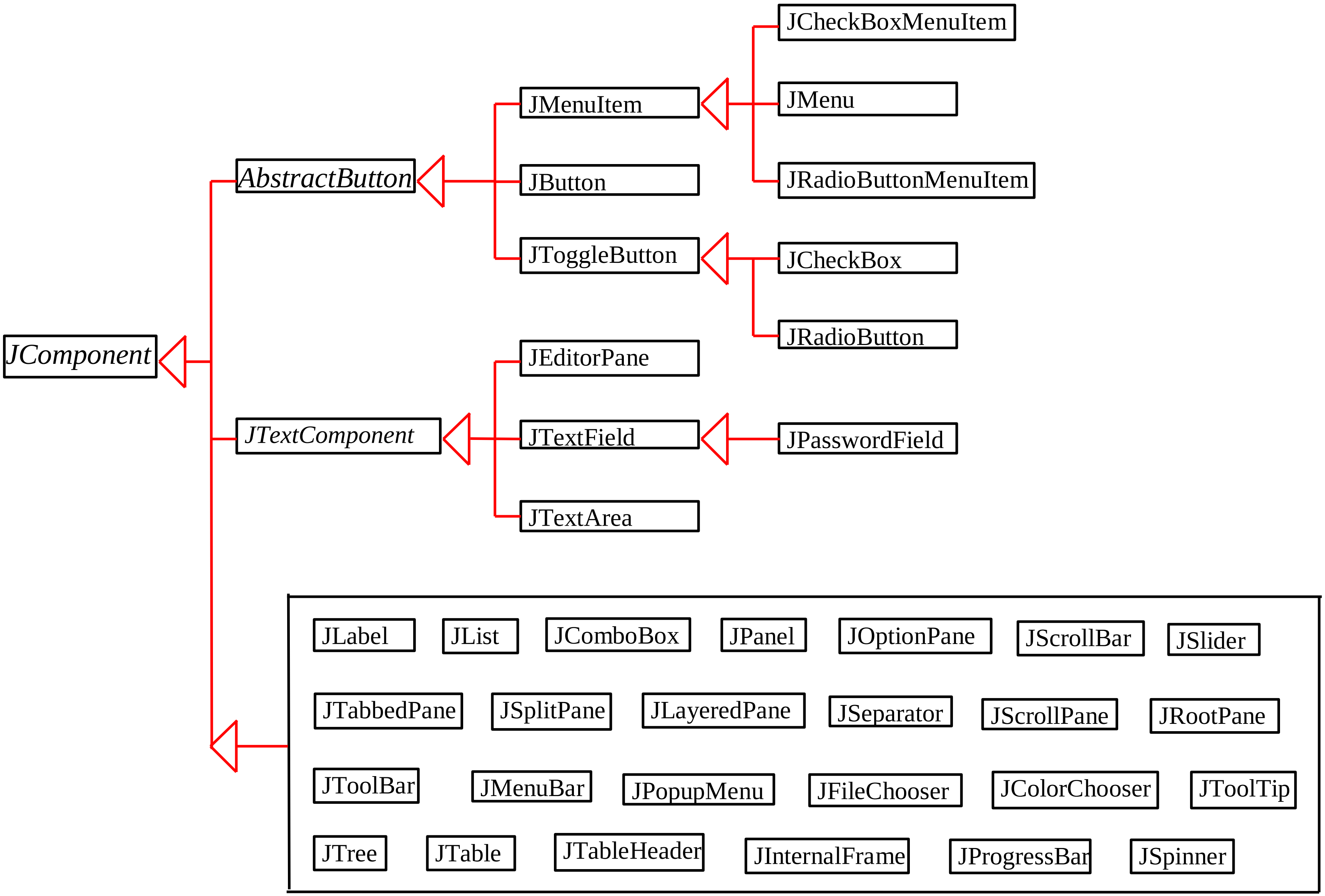
Container classes can contain other GUI components.

GUI Helper Classes

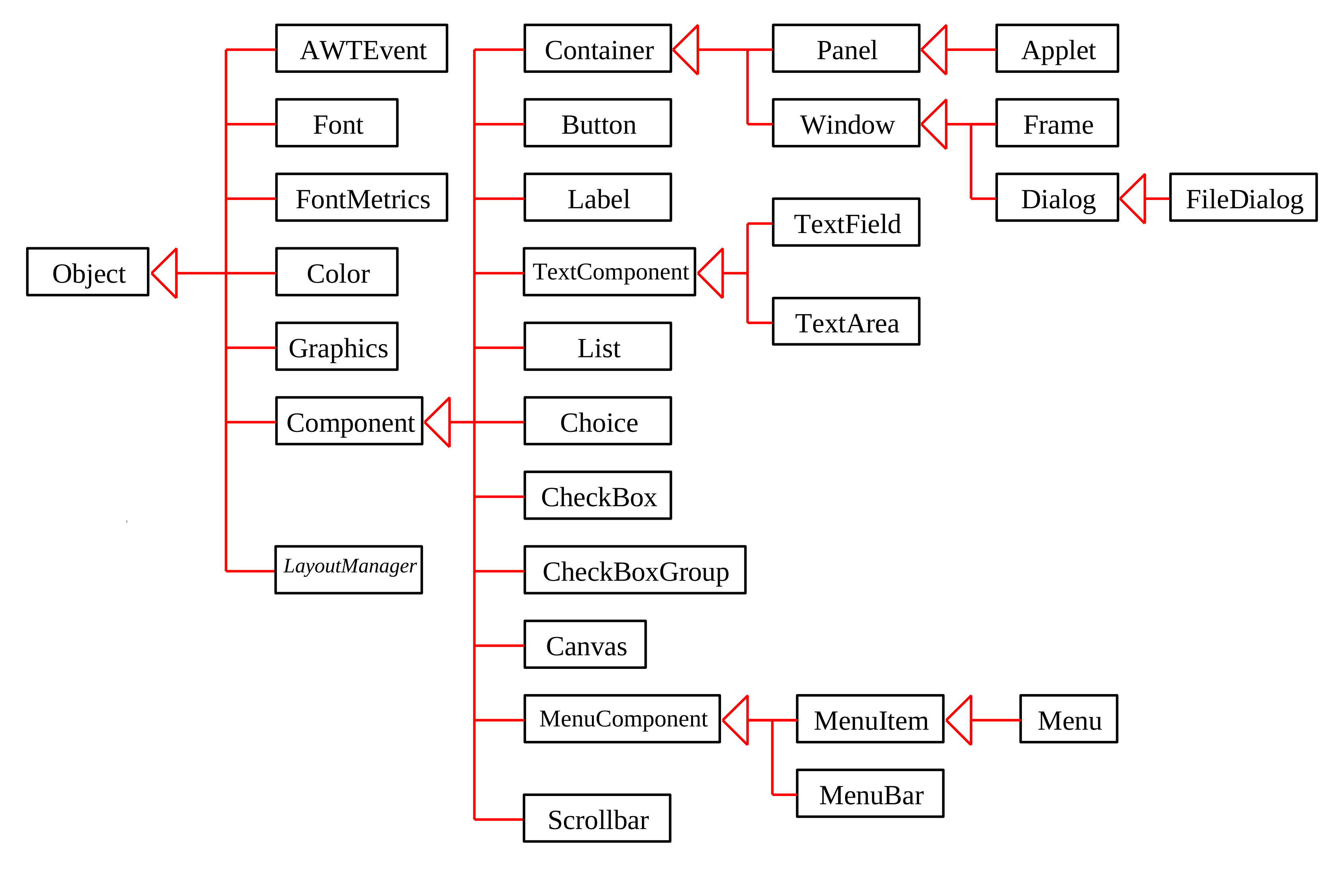


The helper classes are not subclasses of Component. They are used to describe the properties of GUI components such as graphics context, colors, fonts, and dimension.

Swing GUI Components



AWT (Optional)



Frames

- Frame is a window that is not contained inside another window. Frame is the basis to contain other user interface components in Java GUI applications.
- The JFrame class can be used to create windows.
- For Swing GUI programs, use JFrame class to create widows.

Creating Frames

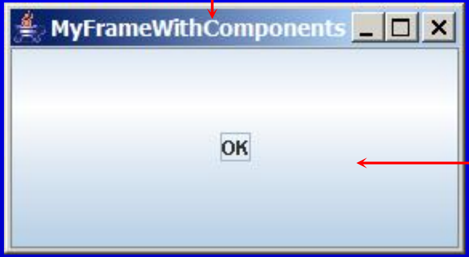
```
import javax.swing.*;
public class MyFrame {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Test Frame");
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
  }
 }
```

MyFrame

Adding Components into a Frame

```
// Add a button into the frame
frame.add(
  new JButton("OK"));
```

Title bar



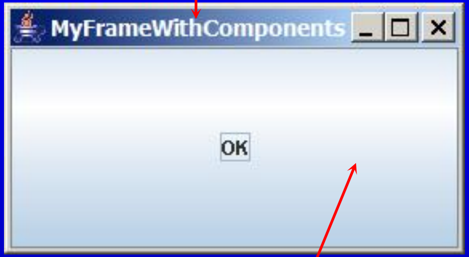
Content pane

MyFrameWithComponents

Content Pane Delegation in JDK 1.5

```
// Add a button into the frame
frame.getContentPane().add(
  new JButton("OK"));
```

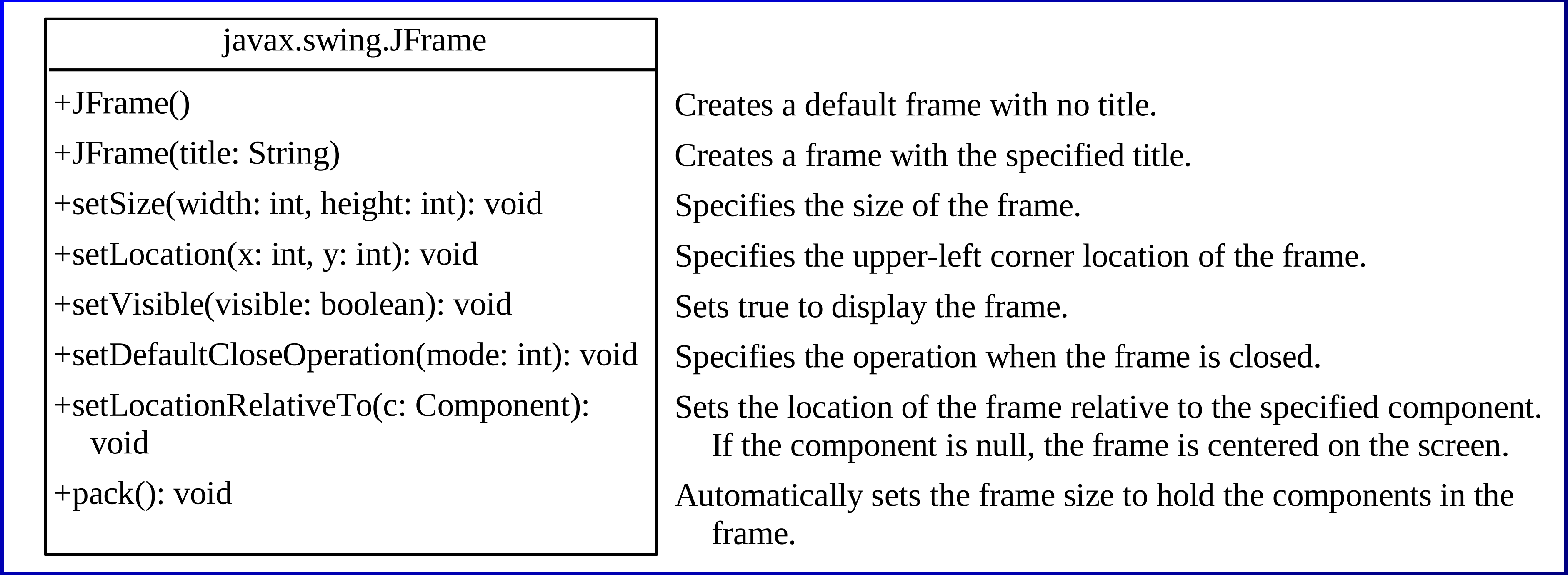
Title bar



Content pane

```
// Add a button into the frame
frame.add(
  new JButton("OK"));
```

JFrame Class



javax.swing.JFrame	
+JFrame()	Creates a default frame with no title.
+JFrame(title: String)	Creates a frame with the specified title.
+setSize(width: int, height: int): void	Specifies the size of the frame.
+setLocation(x: int, y: int): void	Specifies the upper-left corner location of the frame.
+setVisible(visible: boolean): void	Sets true to display the frame.
+setDefaultCloseOperation(mode: int): void	Specifies the operation when the frame is closed.
+setLocationRelativeTo(c: Component): void	Sets the location of the frame relative to the specified component. If the component is null, the frame is centered on the screen.
+pack(): void	Automatically sets the frame size to hold the components in the frame.

Layout Managers

- Java' s layout managers provide a level of abstraction to automatically map your user interface on all window systems.

- The UI components are placed in containers. Each container has a layout manager to arrange the UI components within the container.

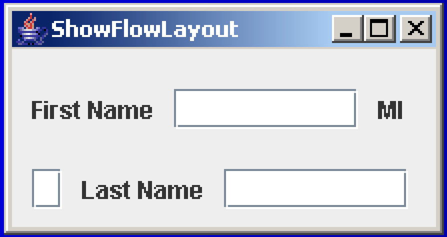
- Layout managers are set in containers using the setLayout(LayoutManager) method in a container.

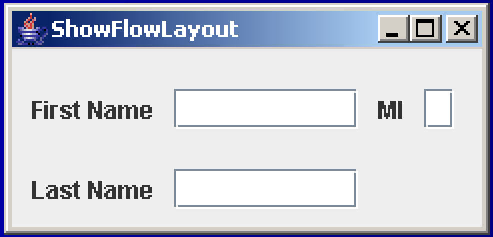
Kinds of Layout Managers

- FlowLayout (this chapter)
- GridLayout (this chapter)
- BorderLayout (this chapter)
- Several other layout managers will be introduced in bonus Chapter 37, “Containers, Layout Managers, and Borders”

FlowLayout Example

Write a program that adds three labels and text fields into the content pane of a frame with a FlowLayout manager.

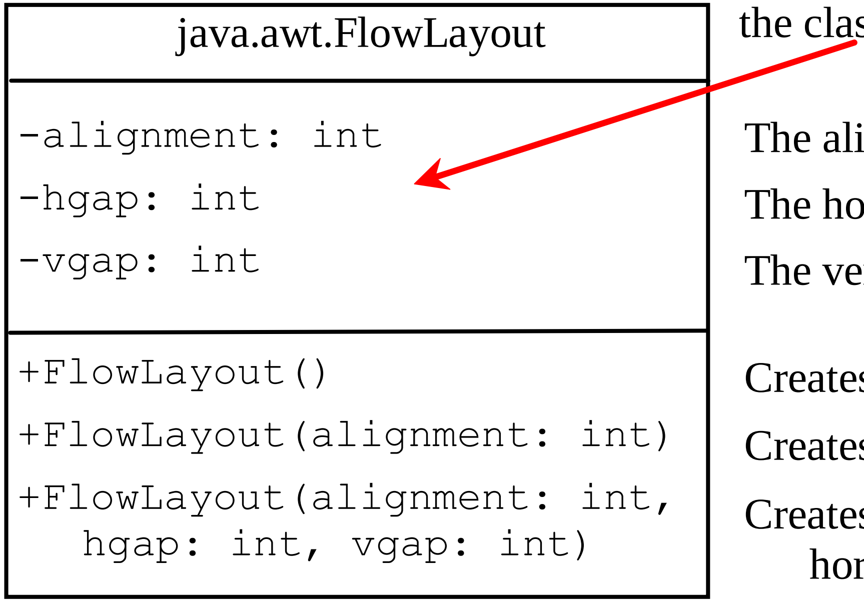




ShowFlowLayout

The FlowLayout Class

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



```
java.awt.FlowLayout
-alignment: int
-hgap: int
-vgap: int
+FlowLayout()
+FlowLayout(alignment: int)
+FlowLayout(alignment: int,
   hgap: int, vgap: int)
```

The alignment of this layout manager (default: `CENTER`).

The horizontal gap between the components (default: 5 pixels).

The vertical gap between the components (default: 5 pixels).

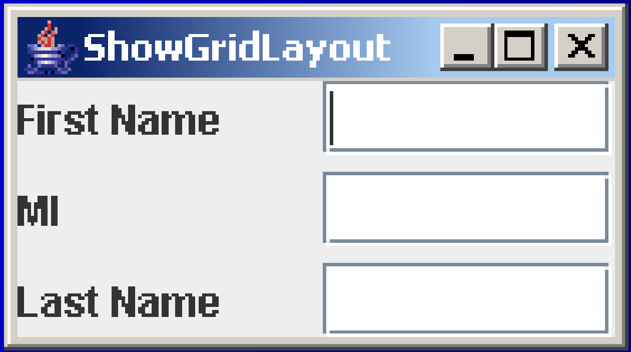
Creates a default `FlowLayout` manager.

Creates a `FlowLayout` manager with a specified alignment.

Creates a `FlowLayout` manager with a specified alignment, horizontal gap, and vertical gap.

GridLayout Example

Rewrite the program in the preceding example using a GridLayout manager instead of a FlowLayout manager to display the labels and text fields.



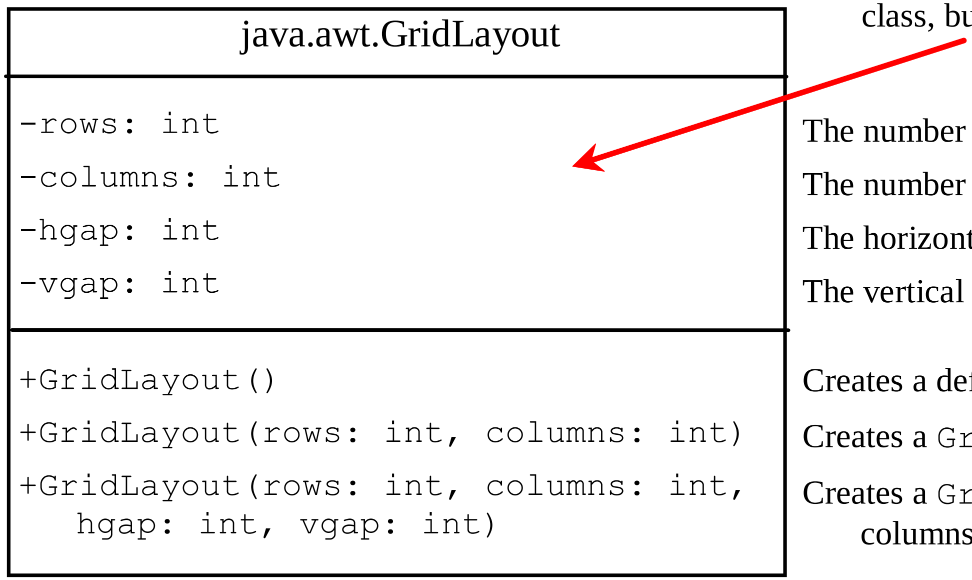
First Name

MI

Last Name

ShowGridLayout

The `GridLayout` Class



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

java.awt.GridLayout

```
-rows: int
-columns: int
-hgap: int
-vgap: int
```

```
+GridLayout()
+GridLayout(rows: int, columns: int)
+GridLayout(rows: int, columns: int,
   hgap: int, vgap: int)
```

The number of rows in the grid (default: 1).

The number of columns in the grid (default: 1).

The horizontal gap between the components (default: 0).

The vertical gap between the components (default: 0).

Creates a default `GridLayout` manager.

Creates a `GridLayout` with a specified number of rows and columns.

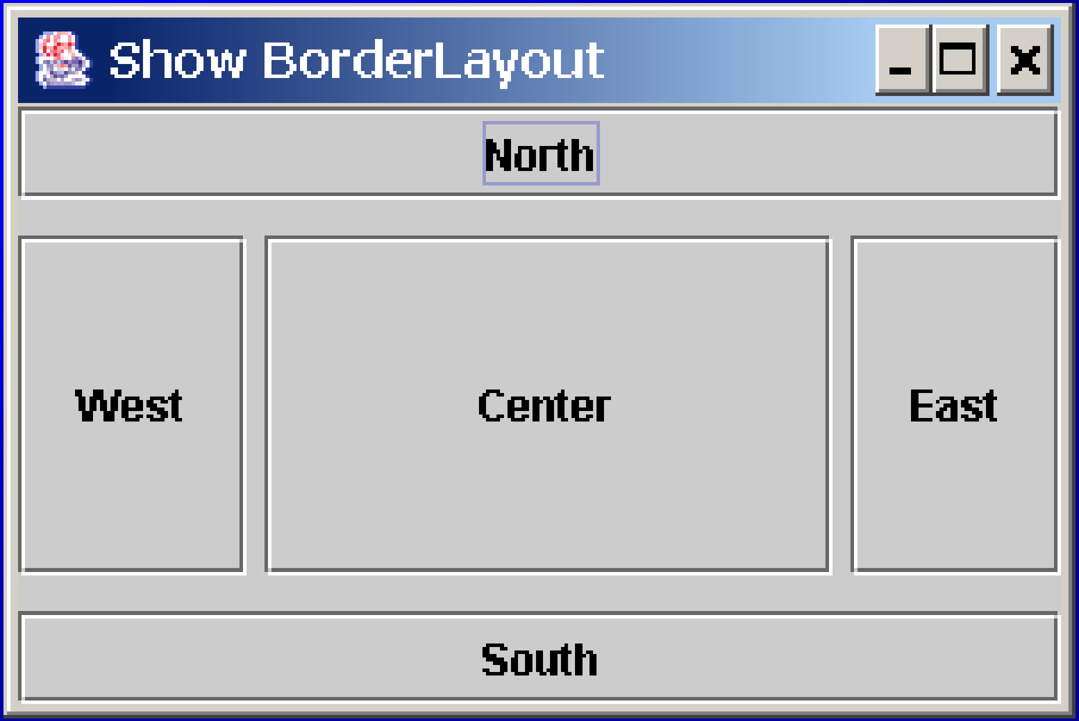
Creates a `GridLayout` manager with a specified number of rows and columns, horizontal gap, and vertical gap.

The `BorderLayout` Manager

The `BorderLayout` manager divides the container into five areas: East, South, West, North, and Center. Components are added to a `BorderLayout` by using the add method.

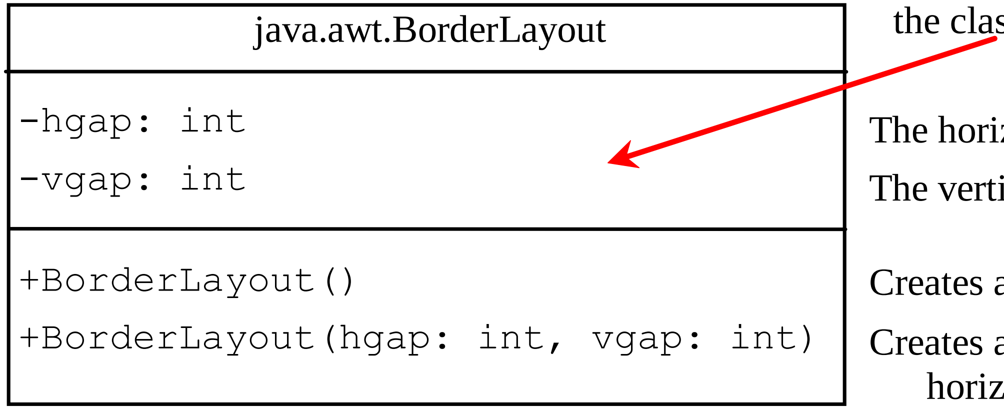
`add(Component, constraint)`, where `constraint` is `BorderLayout.EAST`, `BorderLayout.SOUTH`, `BorderLayout.WEST`, `BorderLayout.NORTH`, or `BorderLayout.CENTER`.

BorderLayout Example



ShowBorderLayout

The `BorderLayout` Class



java.awt.BorderLayout

```
-hgap: int
-vgap: int
```

```
+BorderLayout()
+BorderLayout(hgap: int, vgap: int)
```

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

The horizontal gap between the components (default: 0).

The vertical gap between the components (default: 0).

Creates a default `BorderLayout` manager.

Creates a `BorderLayout` manager with a specified number for horizontal gap and vertical gap.

The `Color` Class

You can set colors for GUI components by using the java.awt.Color class. Colors are made of red, green, and blue components, each of which is represented by a byte value that describes its intensity, ranging from 0 (darkest shade) to 255 (lightest shade). This is known as the *RGB model*.

```
   Color c = new Color(r, g, b);
```

`r`, `g`, and `b` specify a color by its red, green, and blue components.

Example:

```
   Color c = new Color(228, 100, 255);
```

Standard Colors

Thirteen standard colors (black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow) are defined as constants in java.awt.Color.

The standard color names are constants, but they are named as variables with lowercase for the first word and uppercase for the first letters of subsequent words. Thus the color names violate the Java naming convention. Since JDK 1.4, you can also use the new constants: BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, and YELLOW.

Setting Colors

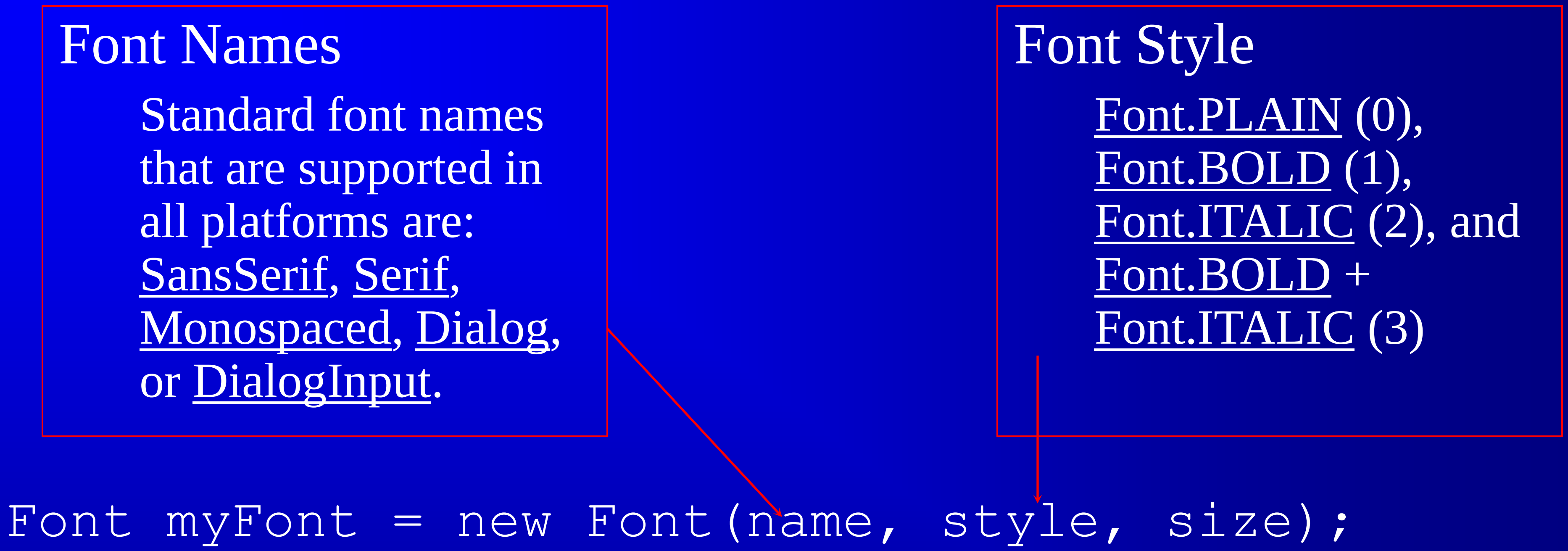
You can use the following methods to set the component' s background and foreground colors:

```
setBackground(Color c)

setForeground(Color c)
```

Example:

```
jbt.setBackground(Color.yellow);

jbt.setForeground(Color.red);
```

The `Font` Class



Font Names

Standard font names that are supported in all platforms are: SansSerif, Serif, Monospaced, Dialog, or DialogInput.

Font Style

Font.PLAIN (0), Font.BOLD (1), Font.ITALIC (2), and Font.BOLD + Font.ITALIC (3)

```
Font myFont = new Font(name, style, size);
```

Example:

```
    Font myFont = new Font("SansSerif ", Font.BOLD, 16);
    Font myFont = new Font("Serif", Font.BOLD+Font.ITALIC, 12);

    JButton jbtOK = new JButton("OK“);
    jbtOK.setFont(myFont);
```

Finding All Available Font Names

```
GraphicsEnvironment e =
  GraphicsEnvironment.getLocalGraphicsEnvironment();
String[] fontnames =
  e.getAvailableFontFamilyNames();
for (int i = 0; i < fontnames.length; i++)
  System.out.println(fontnames[i]);
```

Using Panels as Sub-Containers

- Panels act as sub-containers for grouping user interface components.
- It is recommended that you place the user interface components in panels and place the panels in a frame. You can also place panels in a panel.
- To add a component to JFrame, you actually add it to the content pane of JFrame. To add a component to a panel, you add it directly to the panel using the add method.

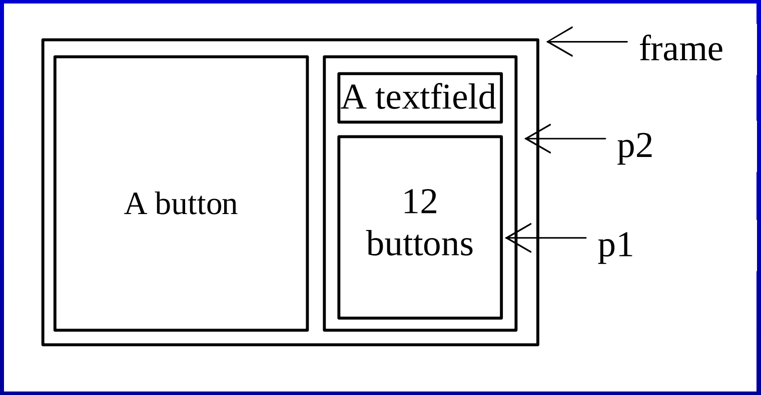
Creating a JPanel

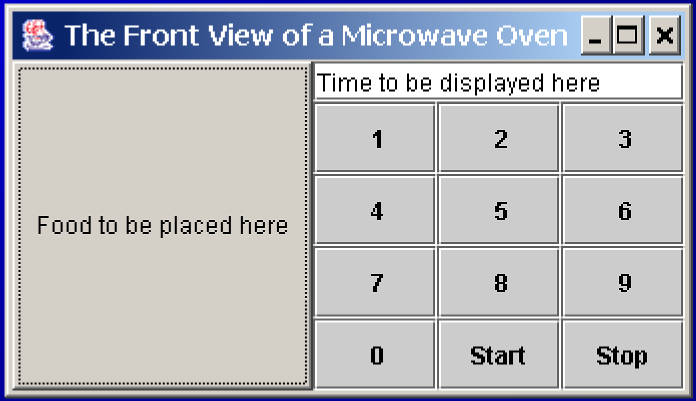
You can use new JPanel() to create a panel with a default FlowLayout manager or new JPanel(LayoutManager) to create a panel with the specified layout manager. Use the add(Component) method to add a component to the panel. For example,

```
JPanel p = new JPanel();
p.add(new JButton("OK"));
```

Testing Panels Example

This example uses panels to organize components. The program creates a user interface for a Microwave oven.

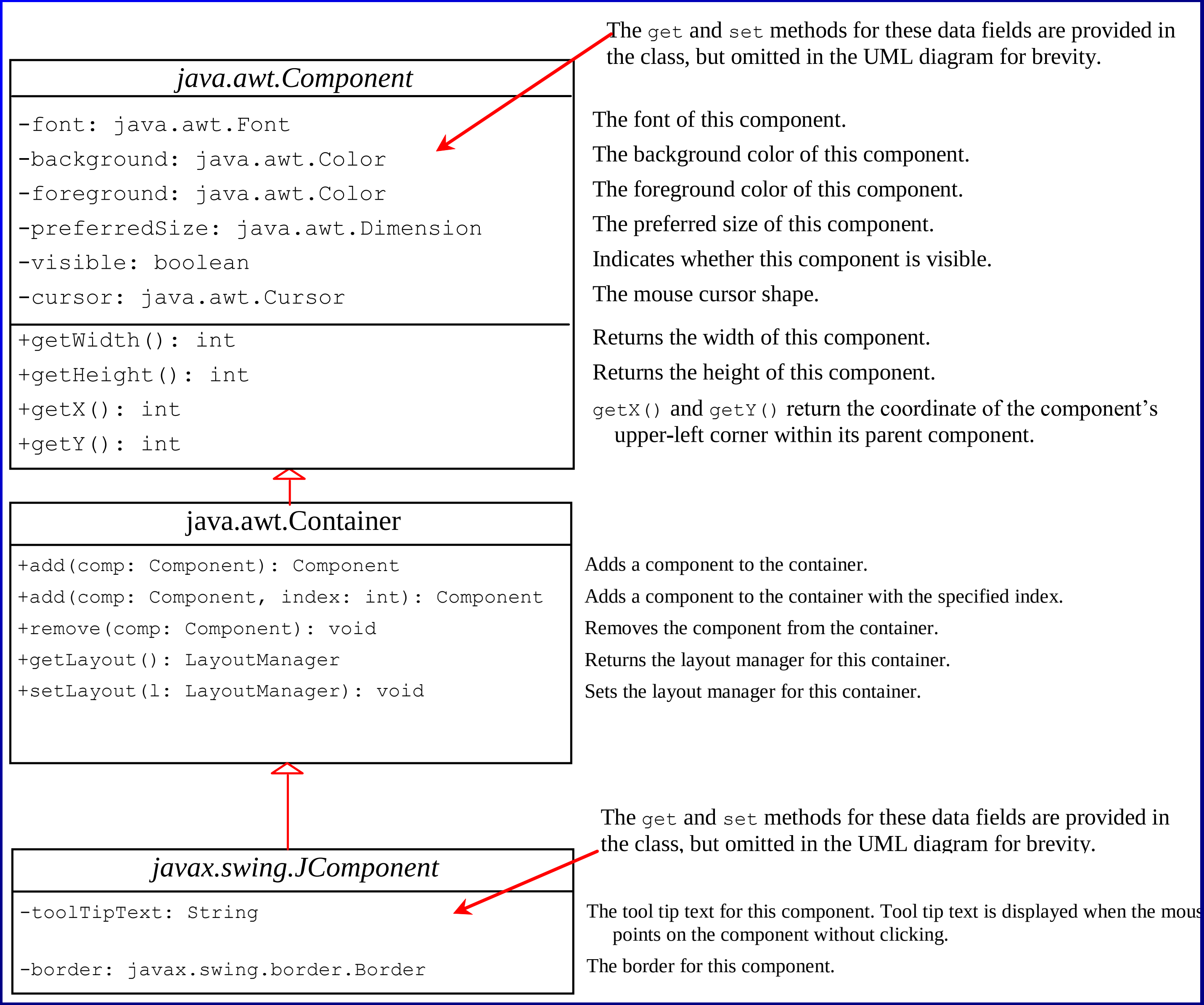




Time to be displayed here

TestPanels

Common Features of Swing Components



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

java.awt.Component

```
-font: java.awt.Font
-background: java.awt.Color
-foreground: java.awt.Color
-preferredSize: java.awt.Dimension
-visible: boolean
-cursor: java.awt.Cursor
```

```
+getWidth(): int
+getHeight(): int
+getX(): int
+getY(): int
```

The font of this component.

The background color of this component.

The foreground color of this component.

The preferred size of this component.

Indicates whether this component is visible.

The mouse cursor shape.

Returns the width of this component.

Returns the height of this component.

`getX()` and `getY()` return the coordinate of the component's upper-left corner within its parent component.

java.awt.Container

```
+add(comp: Component): Component
+add(comp: Component, index: int): Component
+remove(comp: Component): void
+getLayout(): LayoutManager
+setLayout(l: LayoutManager): void
```

Adds a component to the container.

Adds a component to the container with the specified index.

Removes the component from the container.

Returns the layout manager for this container.

Sets the layout manager for this container.

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

javax.swing.JComponent

```
-toolTipText: String

-border: javax.swing.border.Border
```

The tool tip text for this component. Tool tip text is displayed when the mous points on the component without clicking.

The border for this component.

Borders

You can set a border on any object of the JComponent class. Swing has several types of borders. 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. For example, the following code displays a titled border on a panel:

```
JPanel panel = new JPanel();
panel.setBorder(new TitleBorder(“My Panel”));
```

Test Swing Common Features

Component Properties

- `font`
- `background`
- `foreground`
- `preferredSize`
- `minimumSize`
- `maximumSize`

JComponent Properties

- `toolTipText`
- `border`

TestSwingCommonFeatures

Image Icons

Java uses the javax.swing.ImageIcon class to represent an icon. An icon is a fixed-size picture; typically it is small and used to decorate components. Images are normally stored in image files. You can use new ImageIcon(filename) to construct an image icon. 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");
```

TestImageIcon

Splash Screen

A *splash screen* is an image that is displayed while the application is starting up. If your program takes a long time to load, you may display a splash screen to alert the user. For example, the following command:

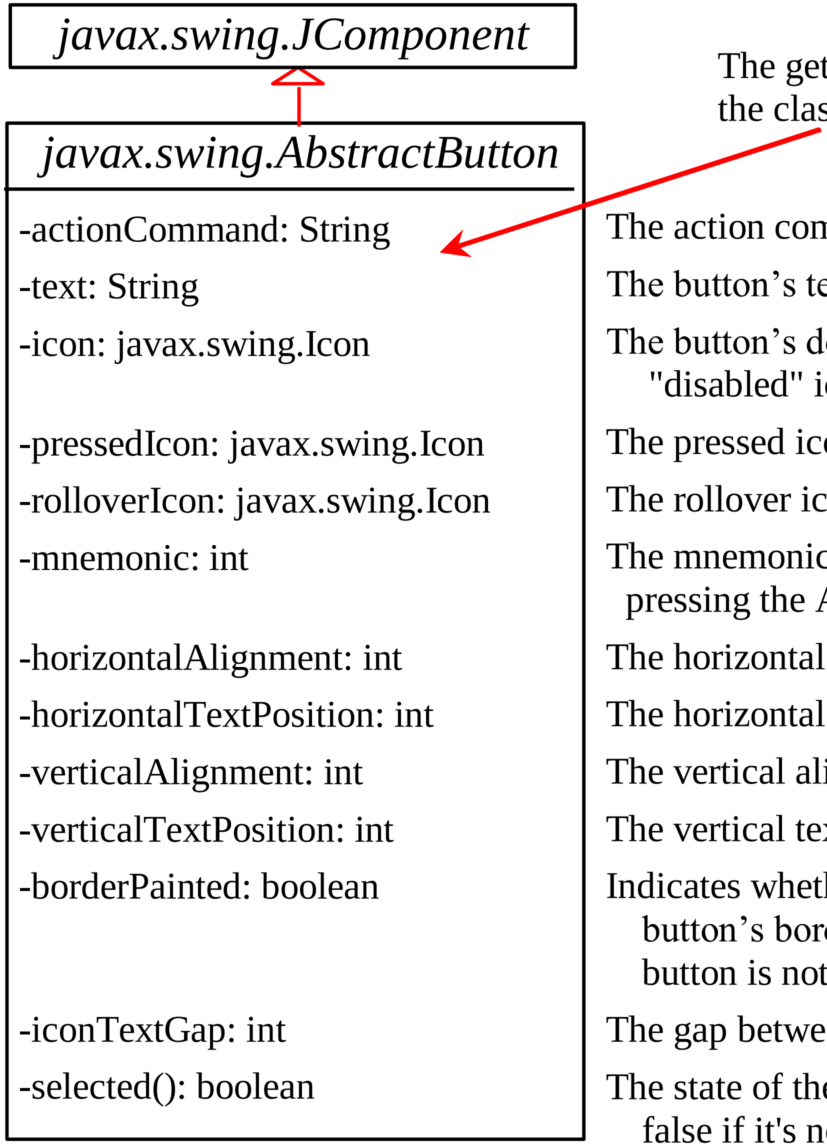
```
java –splash:image/us.gif TestImageIcon
```

displays an image while the program TestImageIcon is being loaded.

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.

AbstractButton



javax.swing.JComponent

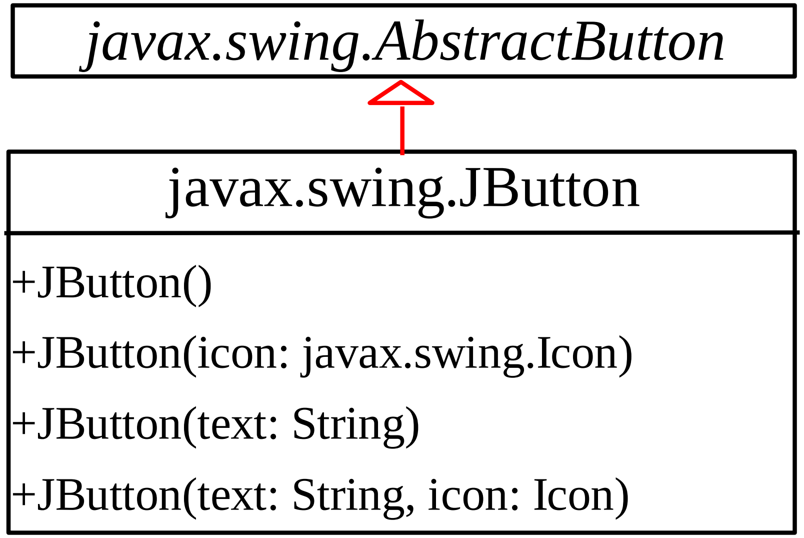
javax.swing.AbstractButton

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.

JButton

JButton inherits AbstractButton and provides several constructors to create buttons.



javax.swing.AbstractButton

javax.swing.JButton

+JButton()	Creates a default button with no text and icon.
+JButton(icon: javax.swing.Icon)	Creates a button with an icon.
+JButton(text: String)	Creates a button with text.
+JButton(text: String, icon: Icon)	Creates a button with text and an icon.

JButton Constructors

The following are `JButton` constructors:

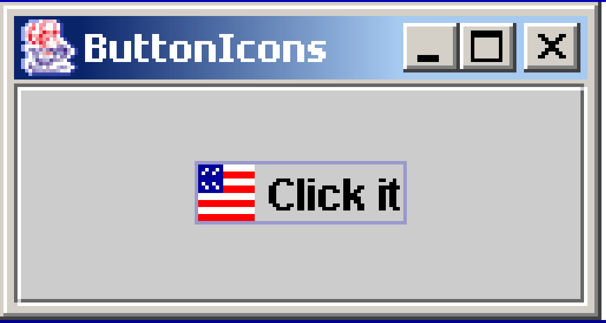
```
JButton()

JButton(String text)

JButton(String text, Icon icon)

JButton(Icon icon)
```

JButton Properties

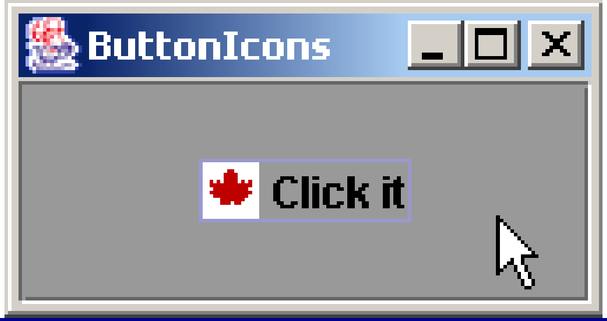
- text
- icon
- mnemonic
- horizontalAlignment
- verticalAlignment
- horizontalTextPosition
- verticalTextPosition
- iconTextGap

Default Icons, Pressed Icon, and Rollover Icon

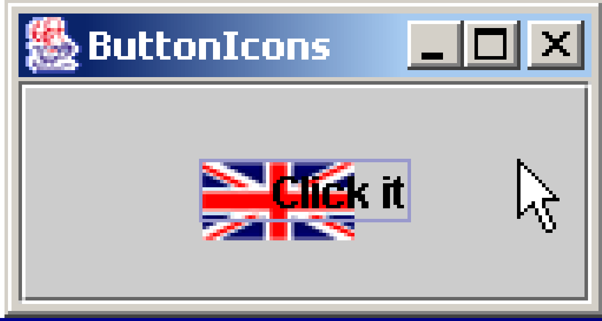
A regular button has a default icon, pressed icon, and rollover icon. Normally, you use the default icon. All other icons are for special effects. A pressed icon is displayed when a button is pressed and a rollover icon is displayed when the mouse is over the button but not pressed.



(A) Default icon



(B) Pressed icon



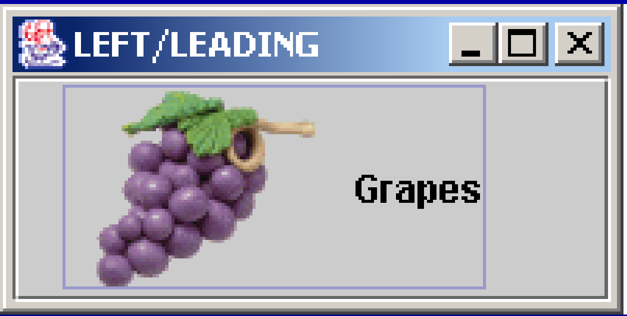
(C) Rollover icon

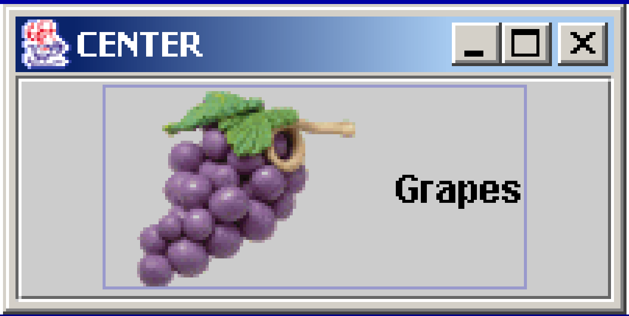
Demo

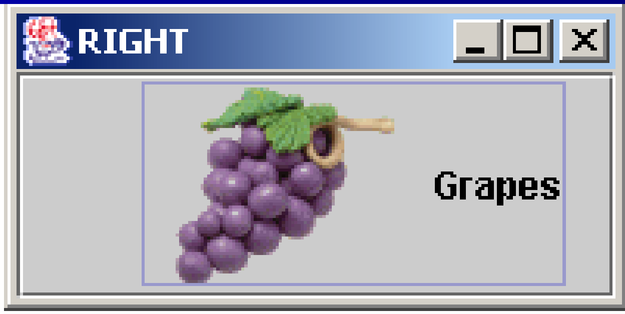
TestButtonIcons

Horizontal 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.

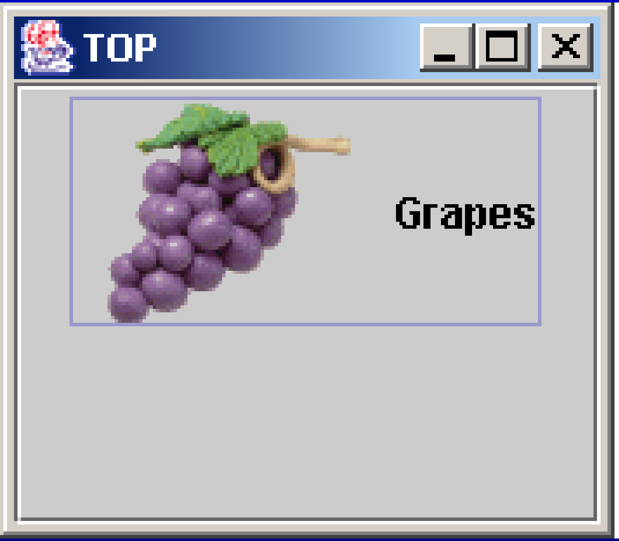


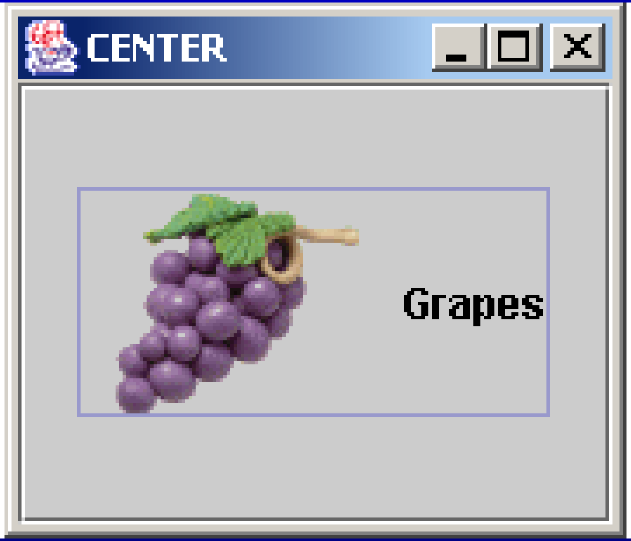


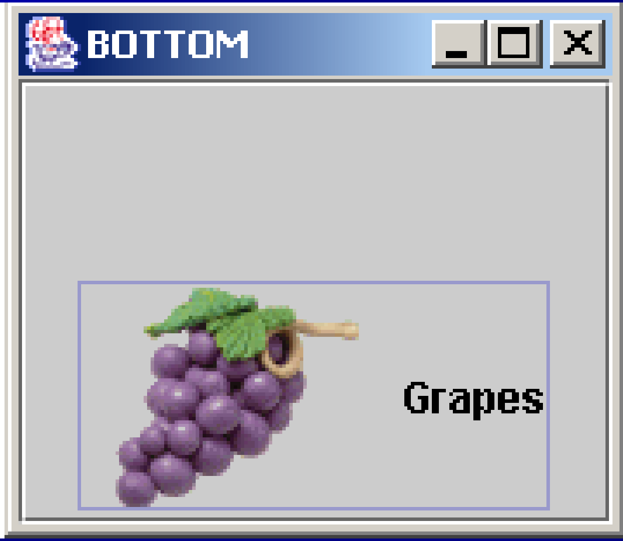


Vertical Alignments

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.

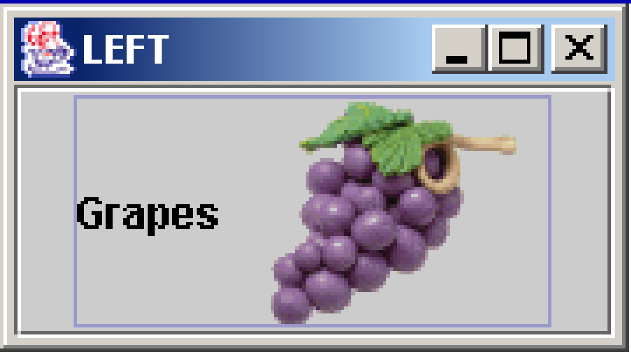


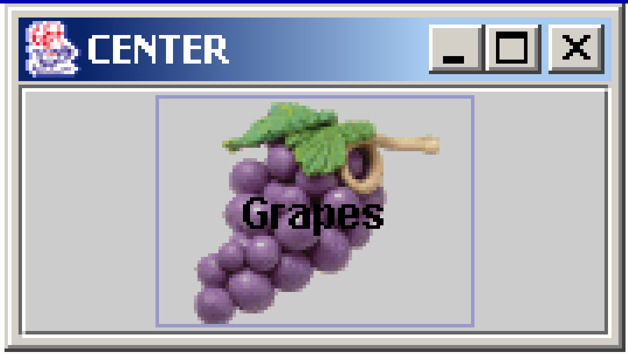


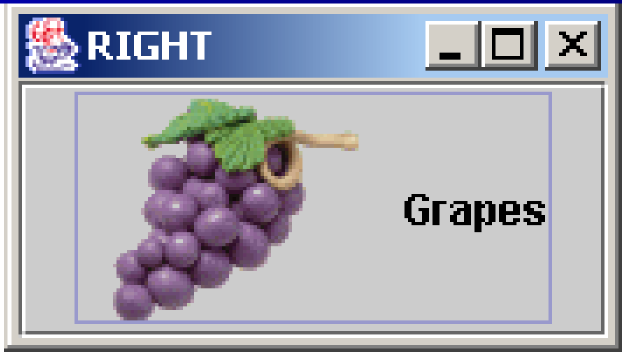


Horizontal Text Positions

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.

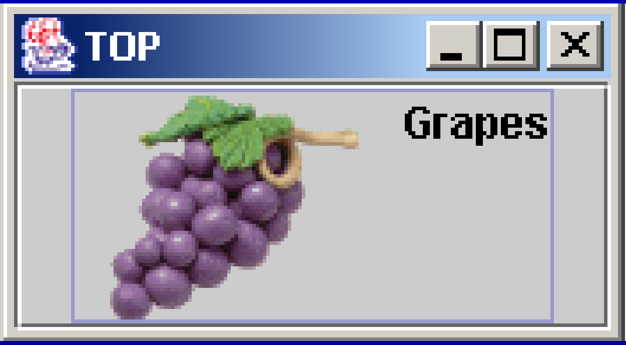


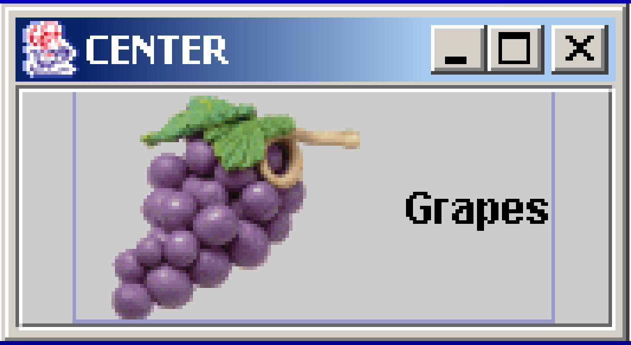


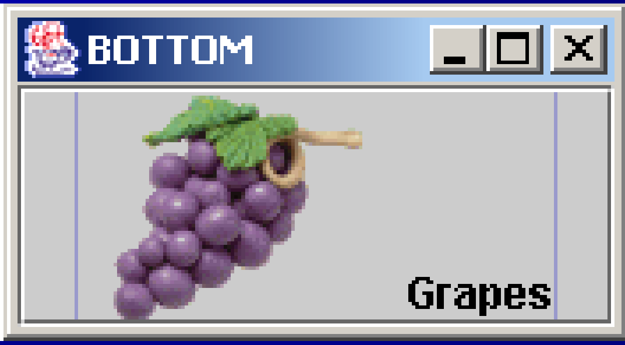


Vertical Text Positions

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.

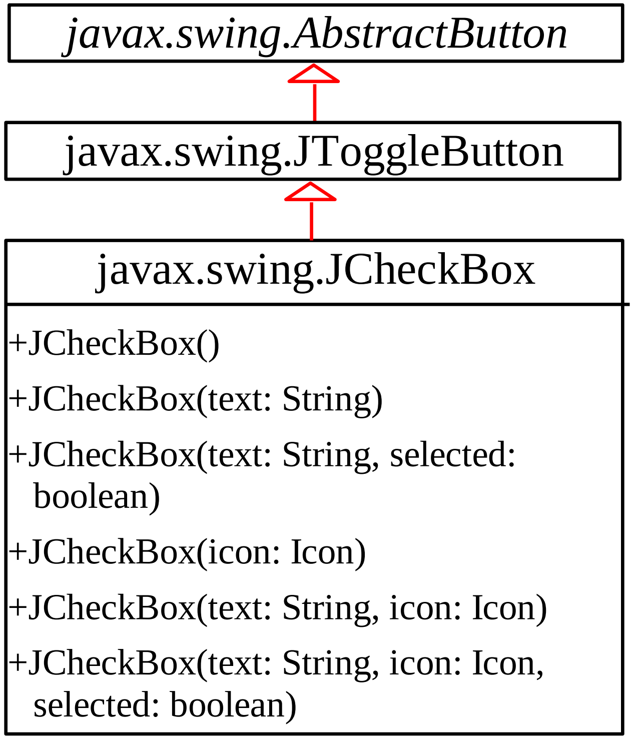






JCheckBox

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



javax.swing.AbstractButton

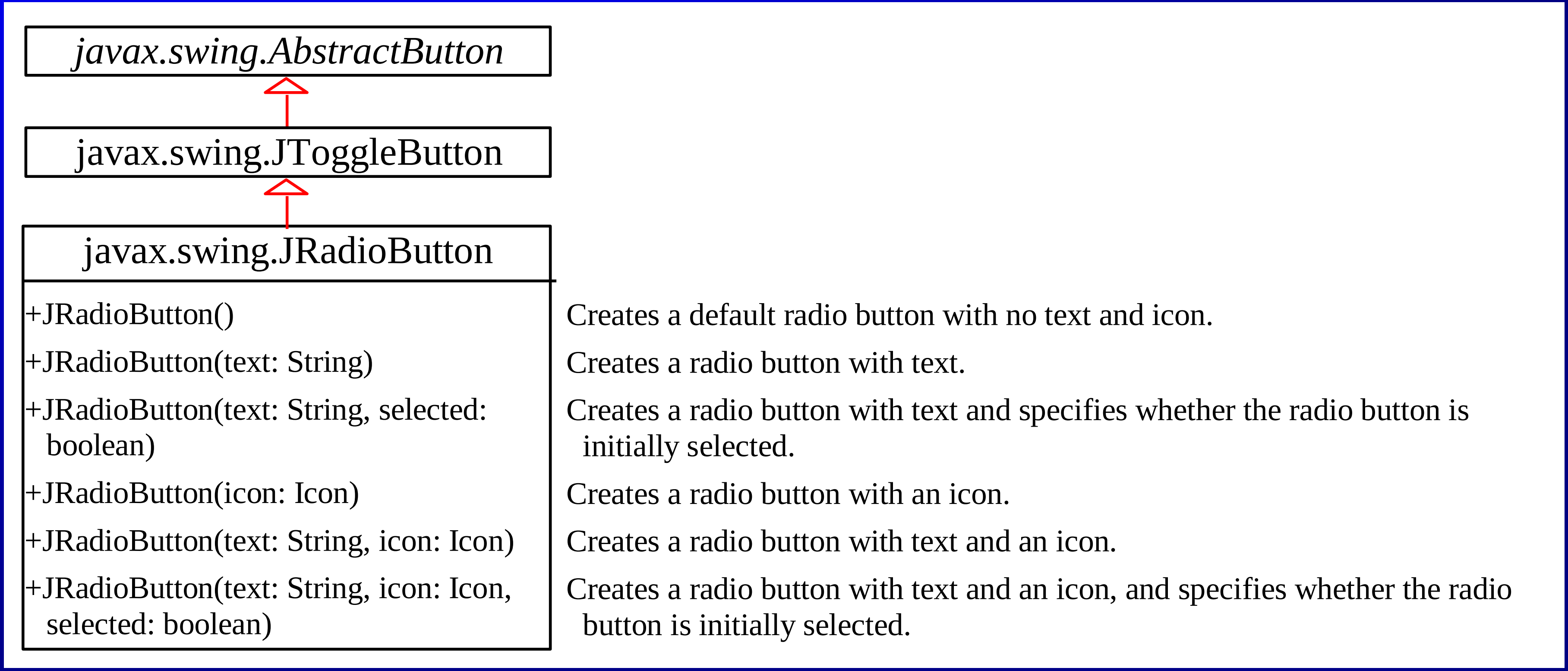
javax.swing.JToggleButton

javax.swing.JCheckBox

+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.

JRadioButton

Radio buttons are variations of check boxes. They are often used in the group, where only one button is checked at a time.



javax.swing.AbstractButton

javax.swing.JToggleButton

javax.swing.JRadioButton

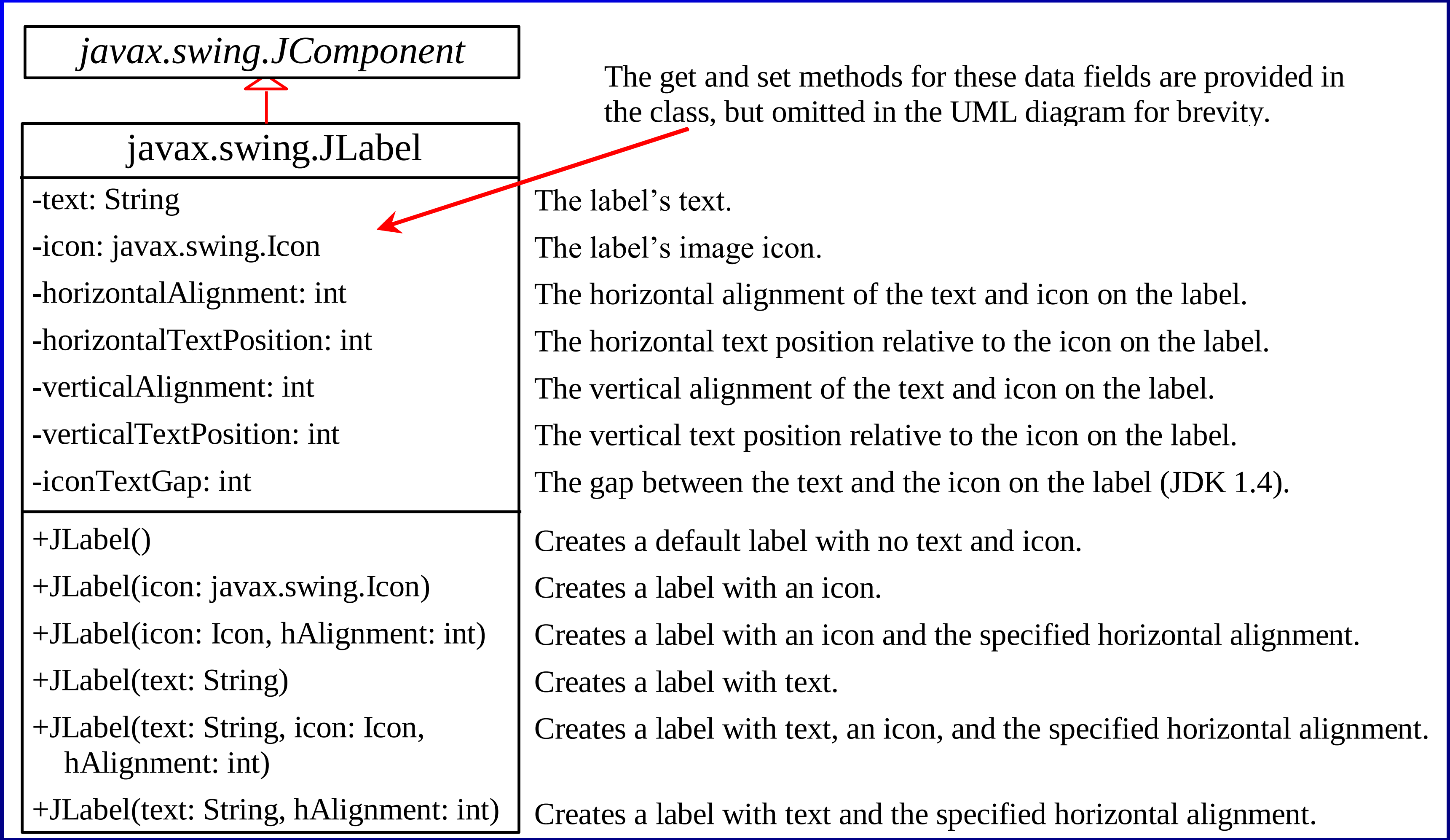
+JRadioButton()	Creates a default radio button with no text and icon.
+JRadioButton(text: String)	Creates a radio button with text.
+JRadioButton(text: String, selected: boolean)	Creates a radio button with text and specifies whether the radio button is initially selected.
+JRadioButton(icon: Icon)	Creates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

Grouping Radio Buttons

```
ButtonGroup btg = new ButtonGroup();
btg.add(jrb1);
btg.add(jrb2);
```

JLabel

A *label* is a display area for a short text, an image, or both.



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 Constructors

The constructors for labels are as follows:

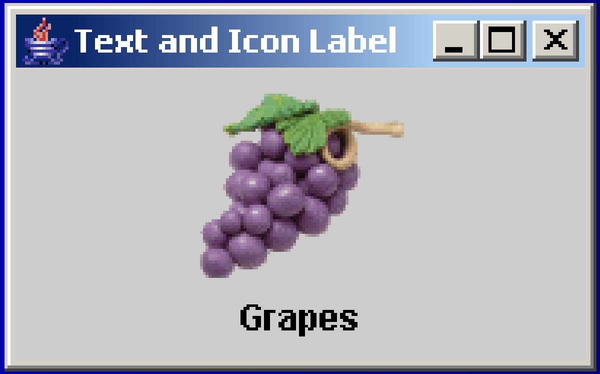
```
JLabel()

JLabel(String text, int horizontalAlignment)

JLabel(String text)

JLabel(Icon icon)

JLabel(Icon icon, int horizontalAlignment)

JLabel(String text, Icon icon, int
horizontalAlignment)
```

JLabel Properties

JLabel inherits all the properties from JComponent and has many properties similar to the ones in JButton, such as text, icon, horizontalAlignment, verticalAlignment, horizontalTextPosition, verticalTextPosition, and iconTextGap.

Using Labels

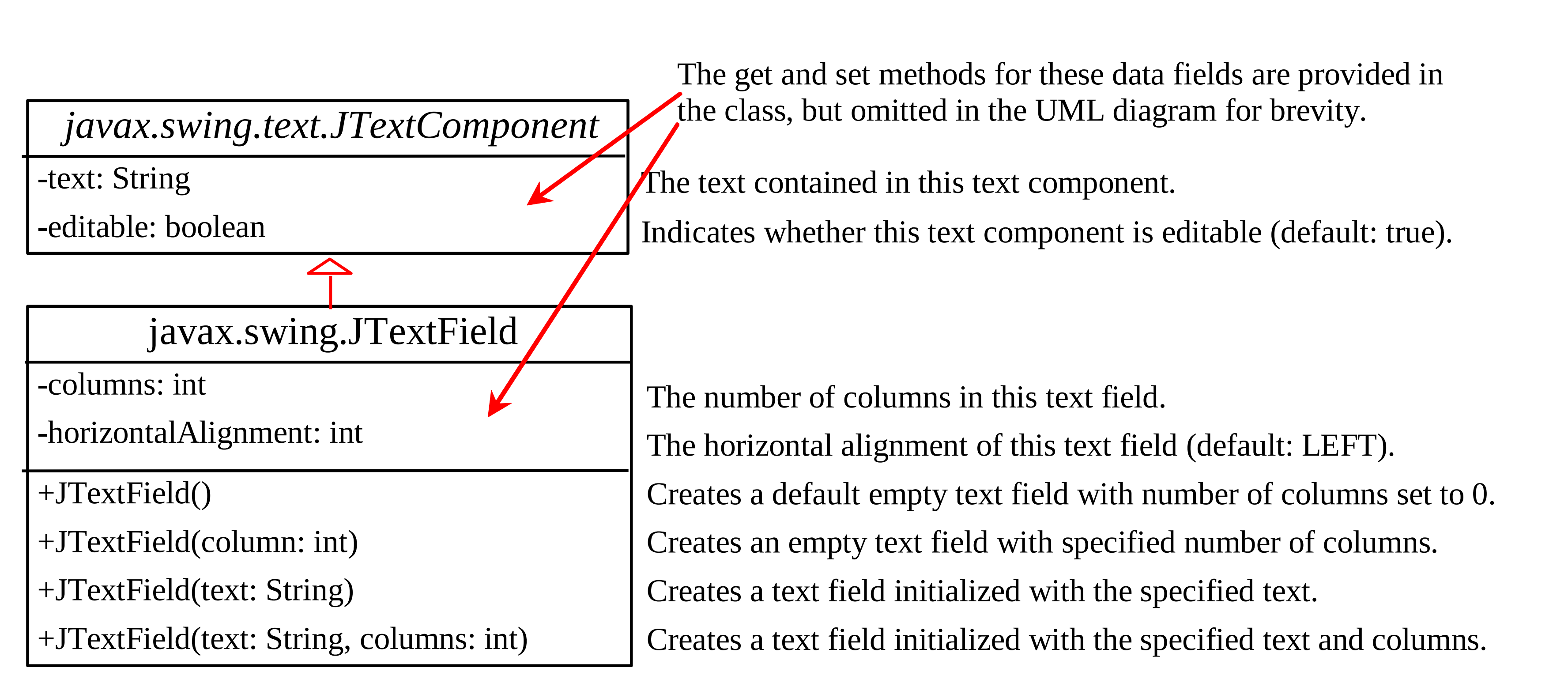
```
// Create an image icon from image file
ImageIcon icon = new ImageIcon("image/grapes.gif");

// 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);
```

JTextField

A *text field* is an input area where the user can type in characters. Text fields are useful in that they enable the user to enter in variable data (such as a name or a description).



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.

`JTextField` Constructors

- `JTextField(int columns)`

  Creates an empty text field with the specified number of columns.

- `JTextField(String text)`

  Creates a text field initialized with the specified text.

- `JTextField(String text, int columns)`

  Creates a text field initialized with the specified text and the column size.

JTextField Properties

- text
- horizontalAlignment
- editable
- columns

`JTextField` Methods

- `getText()`

  Returns the string from the text field.

- `setText(String text)`

  Puts the given string in the text field.

- `setEditable(boolean editable)`

  Enables or disables the text field to be edited. By default, `editable` is `true`.

- `setColumns(int)`

  Sets the number of columns in this text field.
  The length of the text field is changeable.