Chapter 12 GUI Basics
Motivations

The design of the API for Java GUI programming is an excellent example of how the object-oriented principle is applied. In the chapters that follow, you will learn the framework of Java GUI API and use the GUI components 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.
GUI Class Hierarchy (Swing)

Classes in the java.awt package

- Dimension
- Font
- FontMetrics
- Graphics
- Component
- Container
- LayoutManager

Heavyweight
- Applet
- JApplet
- Frame
- JFrame
- Dialog
- JDialog

Lightweight
- Panel
- Window
- JComponent

Swing Components in the javax.swing package

Liang, Introduction to Java Programming, Ninth Edition, (c) 2013 Pearson Education, Inc. All rights reserved.
Container classes can contain other GUI components.
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.
Components Covered in the Brief Version

JComponent

AbstractButton

JMenuItem

JMenu

JMenu

JRadioButtonMenuItem

JRadioButton

JMenu

JCheckBoxMenuItem

JCheckBox

JRadioButton

JCheckBox

JToggleButton

JRadioButtonMenuItem

JMenu

JMenu

JInternalFrame

JLayeredPane

JList

JMenuBar

JOptionPane

JPopupMenu

JProgressBar

JFileChooser

JScrollBar

JScrollPane

JSeparator

JSplitPane

JSlider

JTabbedPane

JTable

JTableHeader

JInternalFrame

JProgressbar

JSpinner
Components Covered in the Comprehensive Version

```
JComponent
  ├── JMenuItem
  │    ├── JMenu
  │    │    ├── JCheckMenuItem
  │    │    │    └── JMenuItem
  │    │    └── JMenuItem
  │    └── JMenuItem
  ├── JCheckBoxMenuItem
  │    └── JMenuItem
  ├── JMenu
  │    └── JMenuItem
  ├── JMenu
  │    └── JMenuItem
  ├── JMenu
  │    └── JMenuItem
  ├── JMenuItem
  │    └── JMenuItem
  ├── JMenuItem
  │    └── JMenuItem
  └── JMenuItem

AbstractButton
  ├── JButton
  │    └── JMenuItem
  └── JMenuItem

JComponent
  └── JTextComponent
      ├── JTextField
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      │    └── JMenuItem
      └── JMenuItem

JTextComponent
  ├── JTextField
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  │    └── JMenuItem
  └── JMenuItem

JTextField
  └── JMenuItem

JTextArea
  └── JMenuItem
```

---
Liang, Introduction to Java Programming, Ninth Edition, (c) 2013 Pearson Education, Inc. All rights reserved.
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.
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);
    }
}

Creating Frames
// Add a button into the frame
frame.getContentPane().add(
    new JButton("OK"));
Content Pane Delegation in JDK 1.5

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFrame()</td>
<td>Creates a default frame with no title.</td>
</tr>
<tr>
<td>JFrame(title: String)</td>
<td>Creates a frame with the specified title.</td>
</tr>
<tr>
<td>setSize(width: int, height: int): void</td>
<td>Specifies the size of the frame.</td>
</tr>
<tr>
<td>setLocation(x: int, y: int): void</td>
<td>Specifies the upper-left corner location of the frame.</td>
</tr>
<tr>
<td>setVisible(visible: boolean): void</td>
<td>Sets true to display the frame.</td>
</tr>
<tr>
<td>setDefaultCloseOperation(mode: int): void</td>
<td>Specifies the operation when the frame is closed.</td>
</tr>
<tr>
<td>setLocationRelativeTo(c: Component): void</td>
<td>Sets the location of the frame relative to the specified component.</td>
</tr>
<tr>
<td>pack(): void</td>
<td>Automatically sets the frame size to hold the components in the frame.</td>
</tr>
</tbody>
</table>

- If the component is null, the frame is centered on the screen.
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.
The FlowLayout Class

java.awt.FlowLayout

- alignment: int
- hgap: int
- vgap: int

+FlowLayout()
+FlowLayout(alignment: int)
+FlowLayout(alignment: int, 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 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.
# The GridLayout Class

The `GridLayout` class is used to create a grid layout manager for placing components in a grid. It is part of the `java.awt` package.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GridLayout()</code></td>
<td>Creates a default <code>GridLayout</code> manager.</td>
</tr>
<tr>
<td><code>GridLayout(rows: int, columns: int)</code></td>
<td>Creates a <code>GridLayout</code> with a specified number of rows and columns.</td>
</tr>
<tr>
<td><code>GridLayout(rows: int, columns: int, hgap: int, vgap: int)</code></td>
<td>Creates a <code>GridLayout</code> manager with a specified number of rows and columns, horizontal gap, and vertical gap.</td>
</tr>
</tbody>
</table>

The `GridLayout` class has the following data fields:

- `rows`: int
- `columns`: int
- `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.

- `rows`: The number of rows in the grid (default: 1).
- `columns`: The number of columns in the grid (default: 1).
- `hgap`: The horizontal gap between the components (default: 0).
- `vgap`: The vertical gap between the components (default: 0).
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.

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

Show BorderLayout

Run
The **BorderLayout** Class

<table>
<thead>
<tr>
<th>java.awt.BorderLayout</th>
</tr>
</thead>
<tbody>
<tr>
<td>-hgap: int</td>
</tr>
<tr>
<td>-vgap: int</td>
</tr>
<tr>
<td>+BorderLayout()</td>
</tr>
<tr>
<td>+BorderLayout(hgap: int, vgap: int)</td>
</tr>
</tbody>
</table>

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)

```java
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,

```java
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.

TestPanels

Run
**Common Features of Swing Components**

### 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.

### 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.

### javax.swing.JComponent

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

The tooltip text for this component. Tool tip text is displayed when the mouse points on the component without clicking.
The border for this component.

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

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

```java
new TitledBorder(String title).
```

To create a line border, use

```java
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:

```java
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

Run
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:

```java
ImageIcon icon = new ImageIcon("image/us.gif");
```
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
java –splash:image/us.gif TestImageIcon
```

displays an image while the program `TestImageIcon` is being loaded.
Components Covered in the brief edition of the book

- JButton
- JToggleButton
- JCheckBox
- JRadioButton
- JComboBox
- JList
- JScrollBar
- JTextField
- JLabel
- JTextArea
- JPasswordField
- JScrollPane
- Component
- Container
- JComponent
- AbstractButton
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**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionCommand: String</td>
<td>The action command of this button.</td>
</tr>
<tr>
<td>text: String</td>
<td>The button’s text (i.e., the text label on the button).</td>
</tr>
<tr>
<td>icon: javax.swing.Icon</td>
<td>The button’s default icon. This icon is also used as the &quot;pressed&quot; and &quot;disabled&quot; icon if there is no explicitly set pressed icon.</td>
</tr>
<tr>
<td>pressedIcon: javax.swing.Icon</td>
<td>The pressed icon (displayed when the button is pressed).</td>
</tr>
<tr>
<td>rolloverIcon: javax.swing.Icon</td>
<td>The rollover icon (displayed when the mouse is over the button).</td>
</tr>
<tr>
<td>mnemonic: int</td>
<td>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.</td>
</tr>
<tr>
<td>horizontalAlignment: int</td>
<td>The horizontal alignment of the icon and text (default: CENTER).</td>
</tr>
<tr>
<td>horizontalTextPosition: int</td>
<td>The horizontal text position relative to the icon (default: RIGHT).</td>
</tr>
<tr>
<td>verticalAlignment: int</td>
<td>The vertical alignment of the icon and text (default: CENTER).</td>
</tr>
<tr>
<td>verticalTextPosition: int</td>
<td>The vertical text position relative to the icon (default: CENTER).</td>
</tr>
<tr>
<td>borderPainted: boolean</td>
<td>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.</td>
</tr>
<tr>
<td>iconTextGap: int</td>
<td>The gap between the text and the icon on the button (JDK 1.4).</td>
</tr>
<tr>
<td>selected(): boolean</td>
<td>The state of the button. True if the check box or radio button is selected, false if it's not.</td>
</tr>
</tbody>
</table>
JButton

JButton inherits AbstractButton and provides several constructors to create buttons.

<table>
<thead>
<tr>
<th>javax.swing.AbstractButton</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.swing.JButton</td>
</tr>
<tr>
<td>JButton()</td>
</tr>
<tr>
<td>JButton(icon: javax.swing.Icon)</td>
</tr>
<tr>
<td>JButton(text: String)</td>
</tr>
<tr>
<td>JButton(text: String, icon: Icon)</td>
</tr>
</tbody>
</table>

Creates a default button with no text and icon.

Creates a button with an icon.

Creates a button with text.

Creates a button with text and an icon.
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
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.

![Examples of horizontal alignments](image-url)
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: \texttt{TOP}, \texttt{CENTER}. The default vertical text position is \texttt{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()
+JCheckBox(text: String)
+JCheckBox(text: String, selected: boolean)
+JCheckBox(icon: Icon)
+JCheckBox(text: String, icon: Icon)
+JCheckBox(text: String, icon: Icon, selected: boolean)
```

- Creates a default check box button with no text and icon.
- Creates a check box with text.
- Creates a check box with text and specifies whether the check box is initially selected.
- Creates a checkbox with an icon.
- Creates a checkbox with text and an icon.
- Creates a check box with text and an icon, and specifies whether the check box is initially selected.
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);
A label is a display area for a short text, an image, or both.

```
javax.swing.JLabel
- text: String
- icon: javax.swing.Icon
- horizontalAlignment: int
- horizontalTextPosition: int
- verticalAlignment: int
- verticalTextPosition: int
- iconTextGap: int

+ JLabel()
+ JLabel(icon: javax.swing.Icon)
+ JLabel(icon: Icon, hAlignment: int)
+ JLabel(text: String)
+ JLabel(text: String, icon: Icon, hAlignment: int)
+ JLabel(text: String, hAlignment: int)
```

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

- The label’s text.
- The label’s image icon.
- The horizontal alignment of the text and icon on the label.
- The horizontal text position relative to the icon on the label.
- The vertical alignment of the text and icon on the label.
- The vertical text position relative to the icon on the label.
- The gap between the text and the icon on the label (JDK 1.4).

- Creates a default label with no text and icon.
- Creates a label with an icon.
- Creates a label with an icon and the specified horizontal alignment.
- Creates a label with text.
- Creates a label with text, an icon, and the specified horizontal alignment.
- Creates a label with text and the specified horizontal alignment.
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);
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.JTextField**

- `columns`: int
- `horizontalAlignment`: int

- **create()**
  - Creates a default empty text field with number of columns set to 0.
- **create(column: int)**
  - Creates an empty text field with specified number of columns.
- **create(text: String)**
  - Creates a text field initialized with the specified text.
- **create(text: String, columns: int)**
  - Creates a text field initialized with the specified text and columns.

**javax.swing.text.JTextComponent**

- `text`: String
- `editable`: boolean

- **get**
  - The text contained in this text component.
- **set**
  - Indicates whether this text component is editable (default: true).

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