Graphical User Interface (GUI)

- Object-oriented programming enables you to develop large-scale software and GUIs effectively.
- When you develop programs to create graphical user interfaces, you will use Java classes such as **JFrame**, **JButton**, **JRadioButton**, **JComboBox**, and **JList** to create frames, buttons, radio buttons, combo boxes, lists, and so on.

Swing vs. AWT

- 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)*.
- AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects.

Swing vs. AWT (Cont.)

- 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*.

Java FX

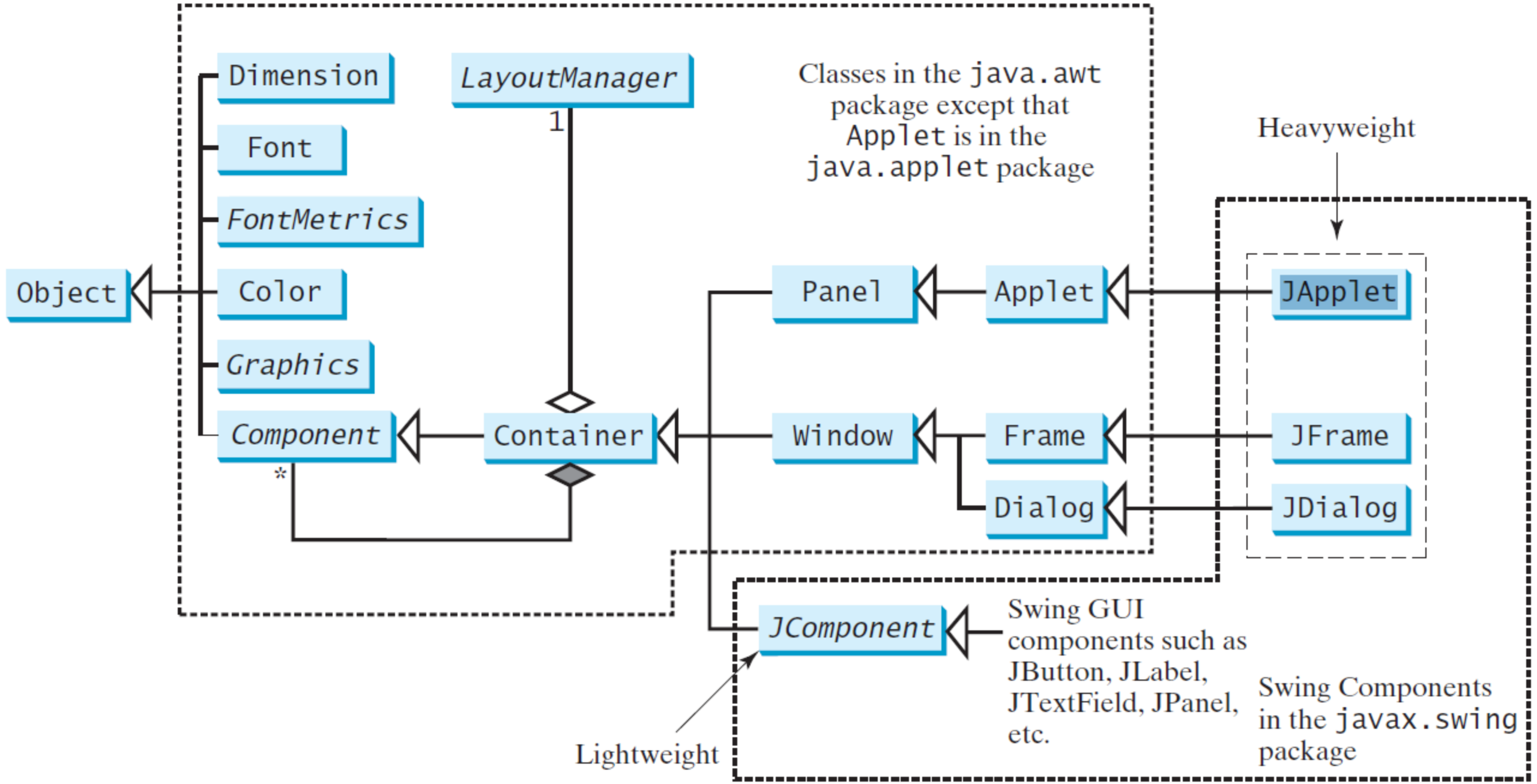
- Swing is designed for developing desktop GUI applications.
- It is now replaced by a completely new GUI platform known as *JavaFX*.
- JavaFX
  - ✓ incorporates modern GUI technologies to enable you to develop rich Internet applications
  - ✓ a JavaFX application can run on a desktop and from a Web browser.
  - ✓ provides a multi-touch support for touch enabled devices such as tablets and smart phones
- Supporting 2D, 3D, animation, video and audio playback

Java GUI API

- The GUI API contains classes that can be classified into three groups: **component classes**, **container classes**, and **helper classes**.
- The subclasses of **Component** are called *component classes* for creating the user interface.
- The classes, such as **JFrame**, **JPanel**, and **JApplet**, are called *container classes* used to contain other components.
- The classes, such as **Graphics**, **Color**, **Font**, **FontMetrics**, and **Dimension**, are called *helper classes* used to support GUI components.

Java GUI API (Cont.)

- The **JFrame**, **JApplet**, **JDialog**, and **JComponent** classes and their subclasses are grouped in the **javax.swing** package.
- **Applet** is in the **java.applet** class.
- Most of the other classes are grouped in the **java.awt** package.


Dimension
LayoutManager
1
Font
FontMetrics
Object
Color
Graphics
Component
Container
*
Classes in the java.awt package except that Applet is in the java.applet package
Heavyweight
Panel
Applet
JApplet
Window
Frame
JFrame
Dialog
JDialog
JComponent
Swing GUI components such as JButton, JLabel, JTextField, JPanel, etc.
Lightweight
Swing Components in the javax.swing package

Component Classes

- An instance of **Component** can be displayed on the screen
- **Component** is the root class of all the user-interface classes including container classes
- **JComponent** is the root class of all the lightweight Swing components

Container Classes

- An instance of **Container** can hold instances of **Component**.
- A container is called a *top-level container* if it can be displayed without being embedded in another container.
- **Window**, **Frame**, **Dialog**, **JFrame**, and **JDialog** are top-level containers.
- **Window**, **Panel**, **Applet**, **Frame**, and **Dialog** are the container classes for AWT components.
- To work with Swing components, use **Container**, **JFrame**, **JDialog**, **JApplet**, and **JPanel**

GUI Helper Classes

- The helper classes, such as **Graphics**, **Color**, **Font**, **FontMetrics**, **Dimension**, and **LayoutManager**, are not subclasses of **Component**.
- They are used to describe the properties of GUI components, such as graphics context, colors, fonts, and dimension
- The helper classes are in the **java.awt** package.
- The Swing components do not replace all the classes in the AWT, only the AWT GUI component classes (e.g., **Button**, **TextField**, **TextArea**).
- The AWT helper classes are still useful in GUI programming.

Frames

- A **frame**, implemented as an instance of the **JFrame** class
- is a window that has decorations such as a border, a title, and supports button components that close or iconify the window. Applications with a GUI usually include at least one **frame**.
- The JFrame class can be used to create windows.
- For Swing GUI programs, use JFrame class to create widows.

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.

JFrame

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	Sets the size of the frame.
+setLocation(x: int, y: int): void	Sets 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.

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);
     }
}
```

Creating Frames (Cont.)

- The frame is not displayed until the **frame.setVisible(true)** method is invoked.
- **frame.setSize(400, 300)** specifies that the frame is **400** pixels wide and **300** pixels high.
- If the **setSize** method is not used, the frame will be sized to display just the title bar.
- Since the **setSize** and **setVisible** methods are both defined in the **Component** class, they are inherited by the **JFrame** class.

Creating Frames (Cont.)

- Invoking **setLocationRelativeTo(null)** centers the frame on the screen.
- Invoking **setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)** tells the program to terminate when the frame is closed.
- If this statement is not used, the program does not terminate when the frame is closed.
- You should invoke the **setSize(w, h)** method before invoking **setLocationRelativeTo(null)** to center the frame.

setDefaultCloseOperation()

- JFrame.EXIT_ON_CLOSE — Exit the application.
- JFrame.HIDE_ON_CLOSE — Hide the frame, but keep the application running.
- JFrame.DO_NOTHING_ON_CLOSE — Ignore the click.

Layout Managers

- Each container contains a layout manager, which is an object responsible for laying out the GUI components in the container
- Java's layout managers provide a level of abstraction to automatically map your user interface on all window systems.
- The Java GUI components are placed in containers. Each container has a layout manager to arrange the GUI components within the container.
- Layout managers are set in containers using the **setLayout(aLayoutManager)** method in a container.

Kinds of Layout Managers

- There are three basic layout managers
  - `FlowLayout`
  - `GridLayout`
  - `BorderLayout`

FlowLayout

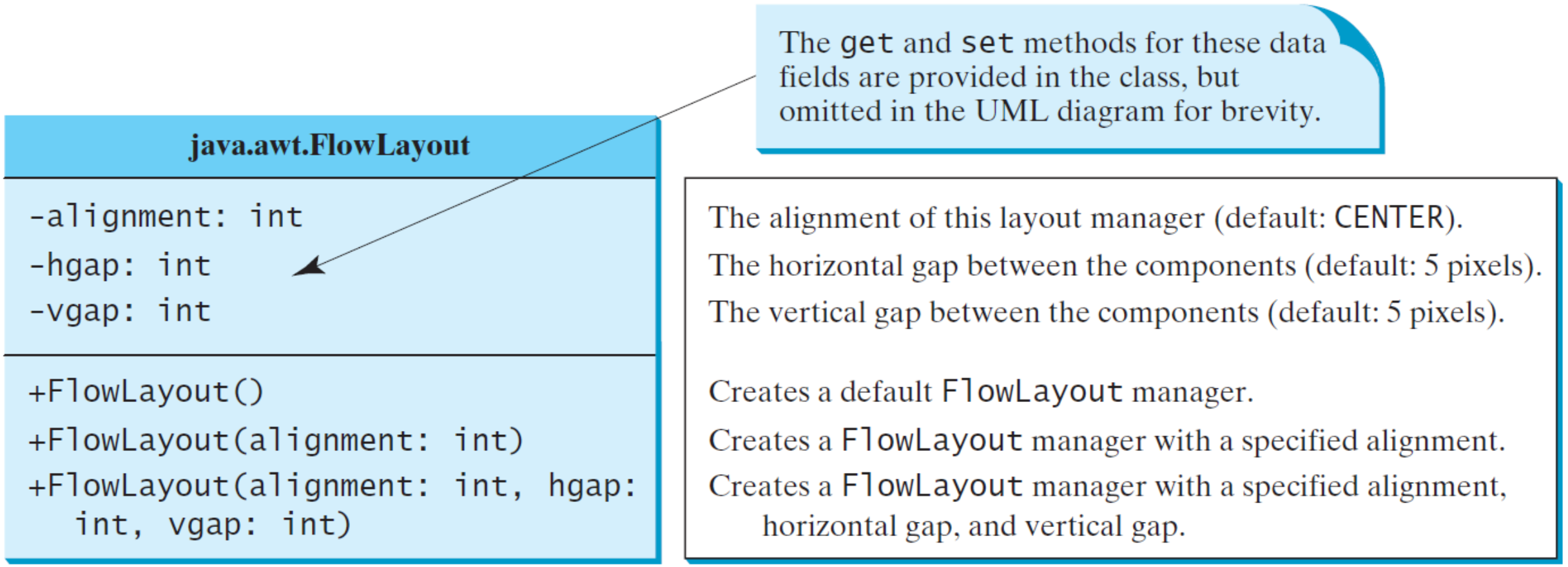
- **FlowLayout** is the simplest layout manager.
- The components are arranged in the container from left to right in the order in which they were added.
- When one row is filled, a new row is started.
- You can specify the way the components are aligned by using one of three constants:

  FlowLayout.RIGHT

  FlowLayout.CENTER

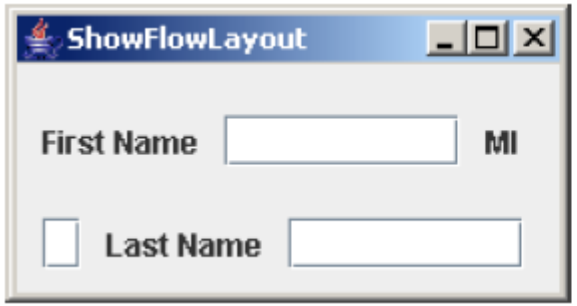
  FlowLayout.LEFT.
- You can also specify the gap between components in pixels.

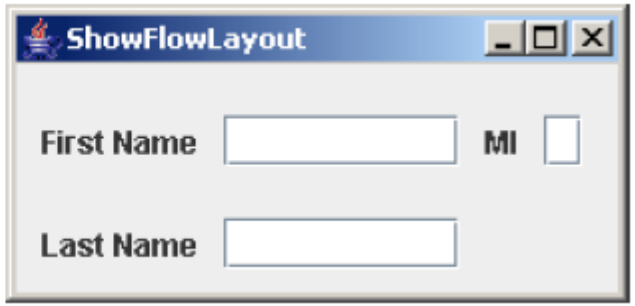
FlowLayout


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.

`FlowLayout` Example

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



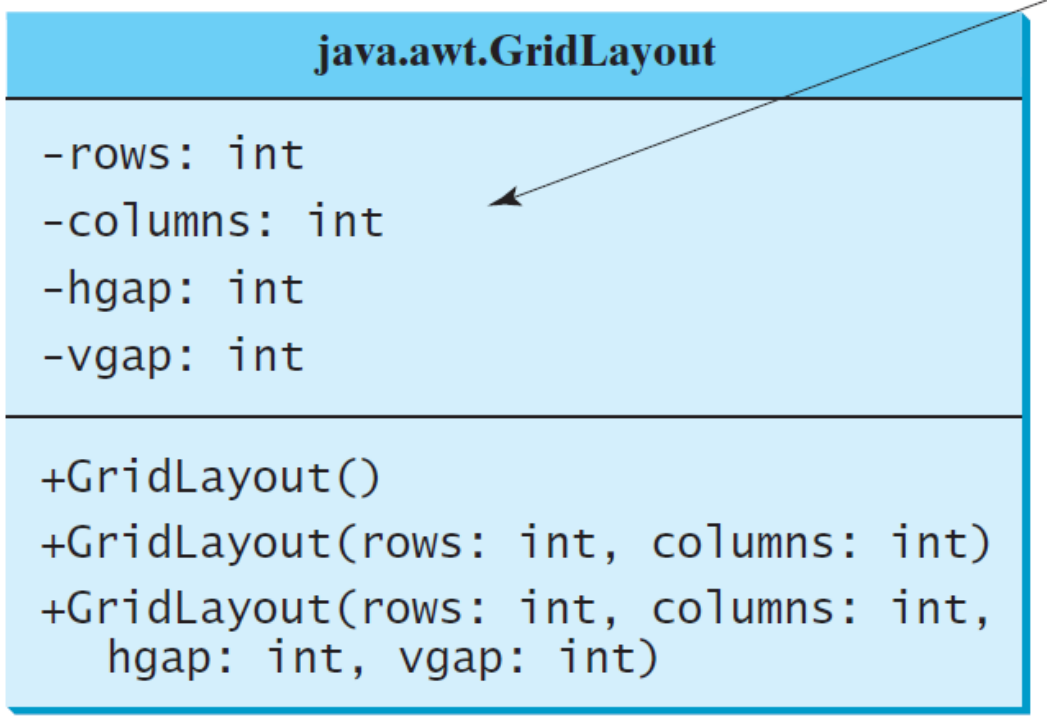


GridLayout

- The **GridLayout** manager arranges components in a grid (matrix) formation.
- The components are placed in the grid from left to right, starting with the first row, then the second, and so on, in the order in which they are added.
- The number of rows or the number of columns can be zero, but not for both
- If one is zero and the other is nonzero, the nonzero dimension is fixed, while the zero dimension is determined dynamically by the layout manager.
- If both the number of rows and the number of columns are nonzero, the number of rows is the dominating parameter; that is, the number of rows is fixed, and the layout manager dynamically calculates the number of columns.

GridLayout

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



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).

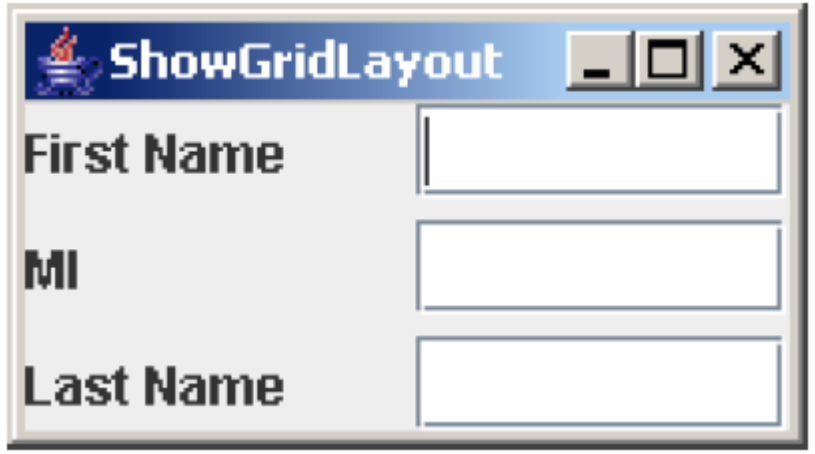
Creates a default `GridLayout` manager.

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.

`GridLayout` Example

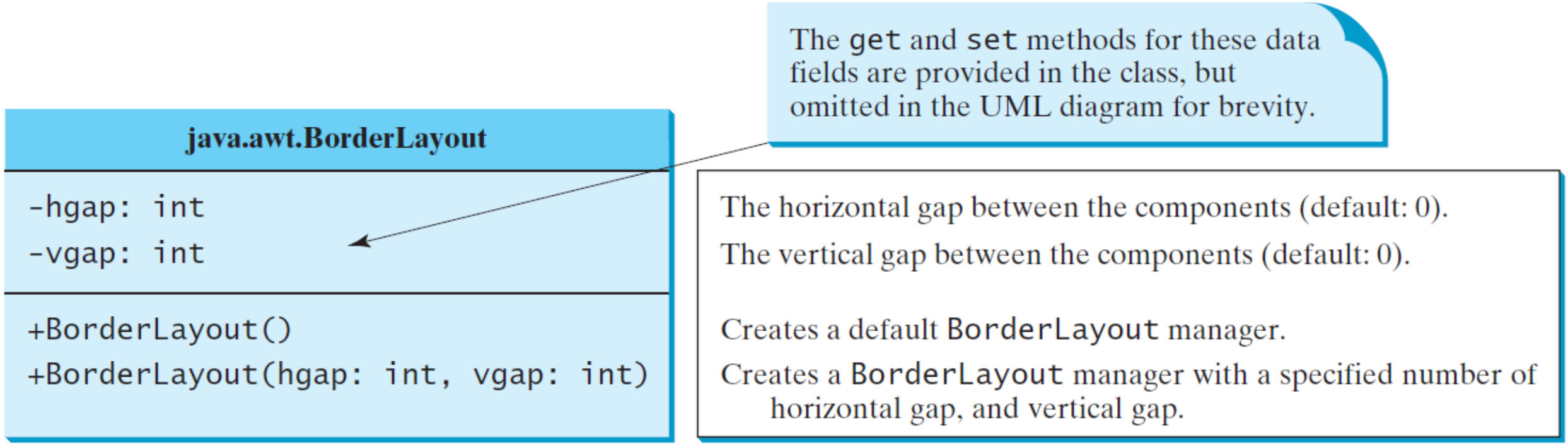
- Rewrite the program in the preceding example using a `GridLayout` manager instead of a `FlowLayout` manager to display the labels and text fields.



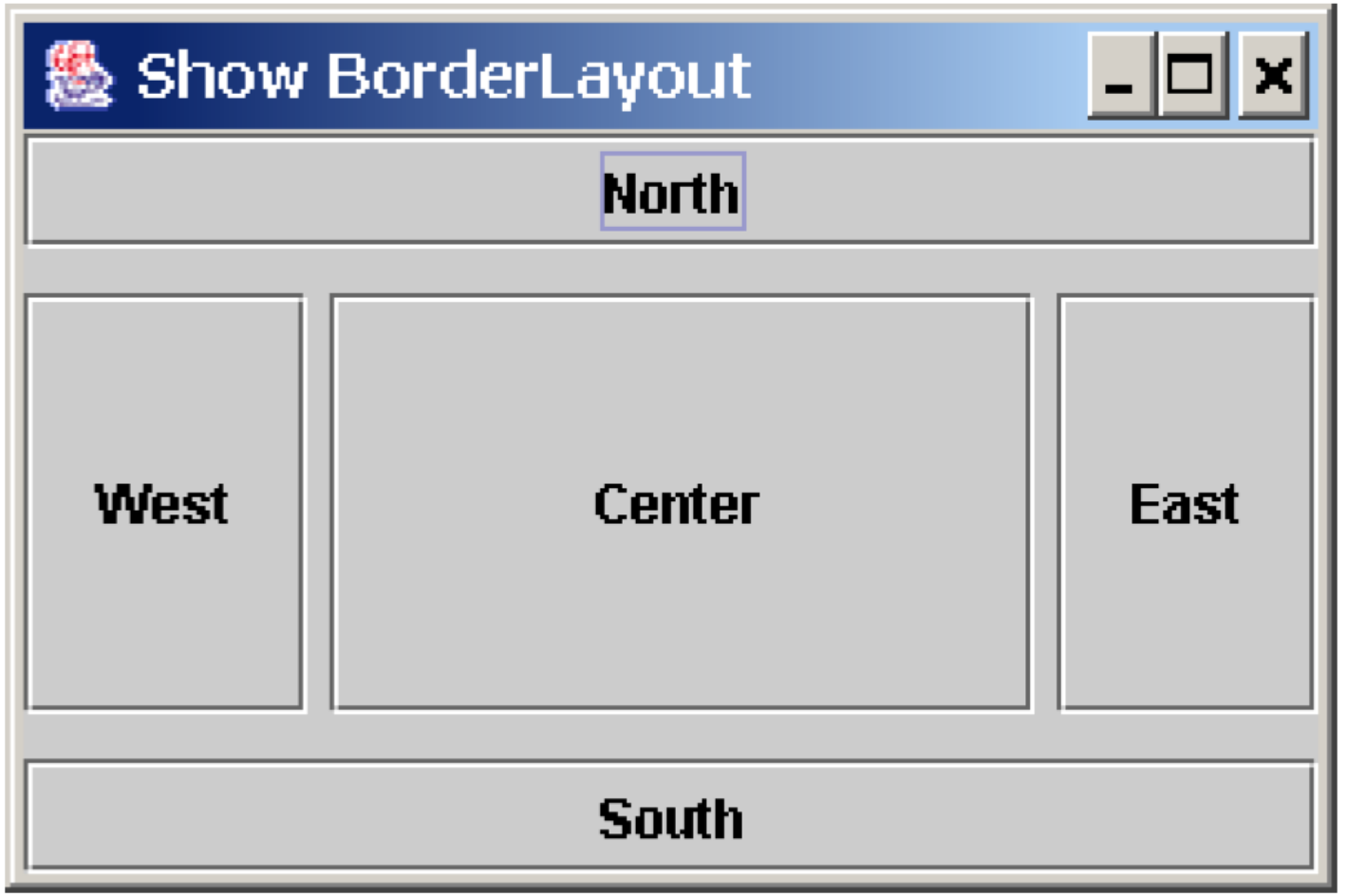
BorderLayout

- The **BorderLayout** manager divides a container into five areas: East, South, West, North, and Center.
- Components are added to a **BorderLayout** by using **add(Component, index)**, where **index** is a constant **BorderLayout.EAST**, **BorderLayout.SOUTH**, **BorderLayout.WEST**, **BorderLayout.NORTH**, or **BorderLayout.CENTER**.
- The North and South components can stretch horizontally;
- The East and West components can stretch vertically;
- The Center component can stretch both horizontally and vertically to fill any empty space.

BorderLayout (Cont.)



`BorderLayout` Example



Using Panels as Sub-Containers

- Suppose that you want to place 12 buttons and a text field in a frame.
- The buttons are placed in grid formation, but the text field is placed on a separate row.
- It is difficult to achieve the desired look by placing all the components in a single container.
- With Java GUI programming, you can divide a window into panels.
- Panels act as sub-containers to group user interface components.
- You add the buttons in one panel, then add the panel to the frame.

Using Panels as Sub-Containers (Cont.)

- 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"));
```

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);
```

The `Color` Class (Cont.)

- You can use the **setBackground(Color c)** and **setForeground(Color c)** methods defined in the **java.awt.Component** class to set a component's background and foreground colors.
- Here is an example of setting the background and foreground of a button:

```
JButton jbtOK = new JButton("OK");
jbtOK.setBackground(color);
jbtOK.setForeground(new Color(100, 1, 1));
```

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)
Font.BOLD+Font.ITALIC(3)

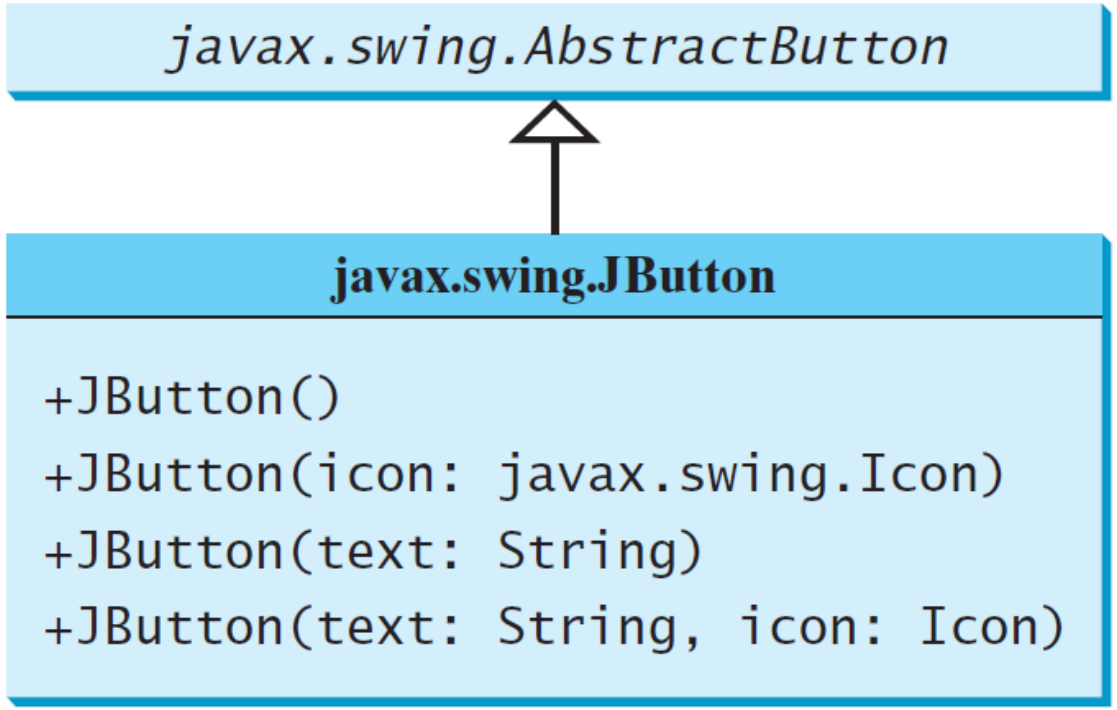
```
Font myFont = new Font(String name, int style, int size);
```

- Example:

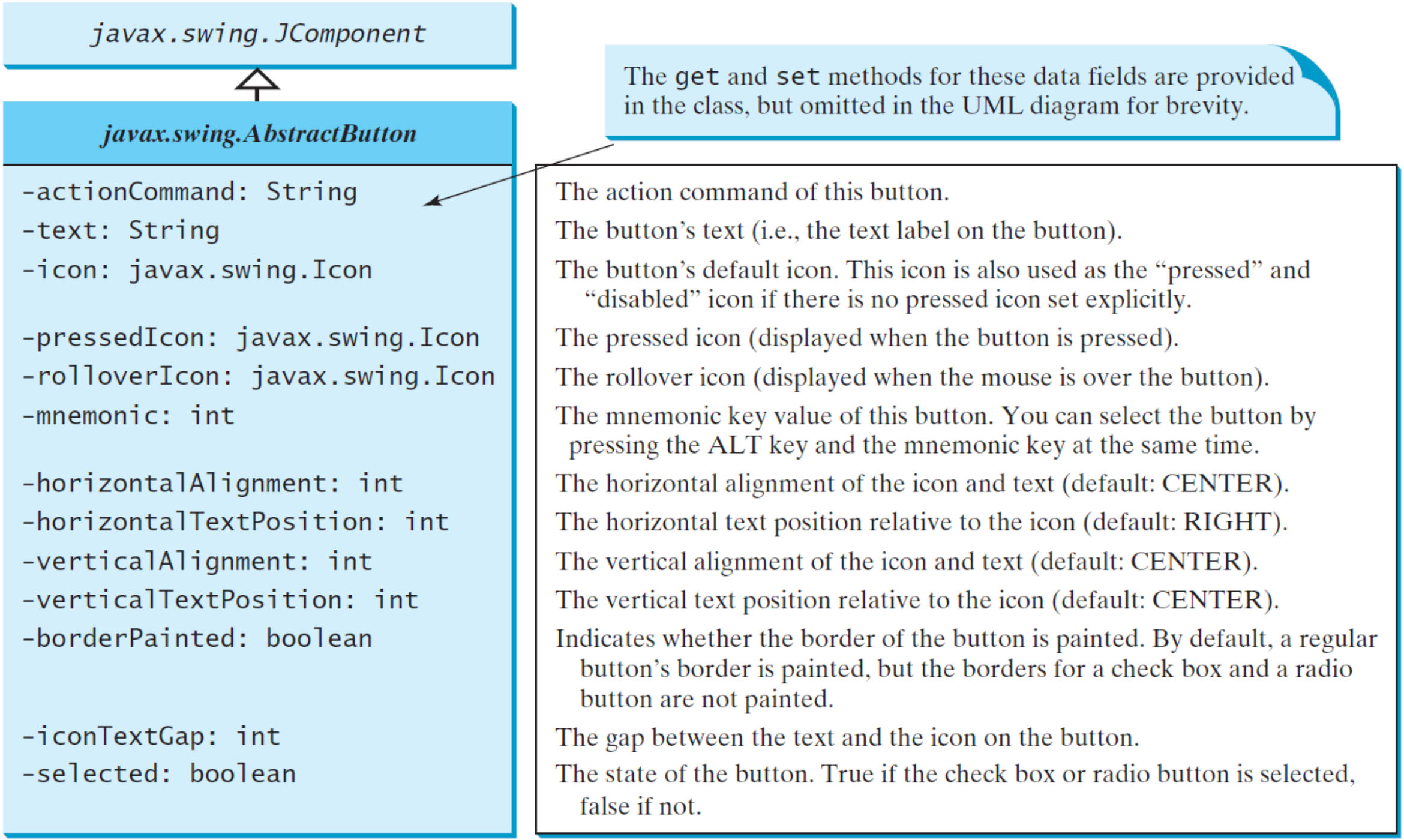
```
Font myFont = new Font("SansSerif ", Font.BOLD, 16);
Font myFont = new Font("Serif", Font.BOLD+Font.ITALIC, 12);
JButton jbtOK = new JButton("OK");
jbtOK.setFont(myFont);
```

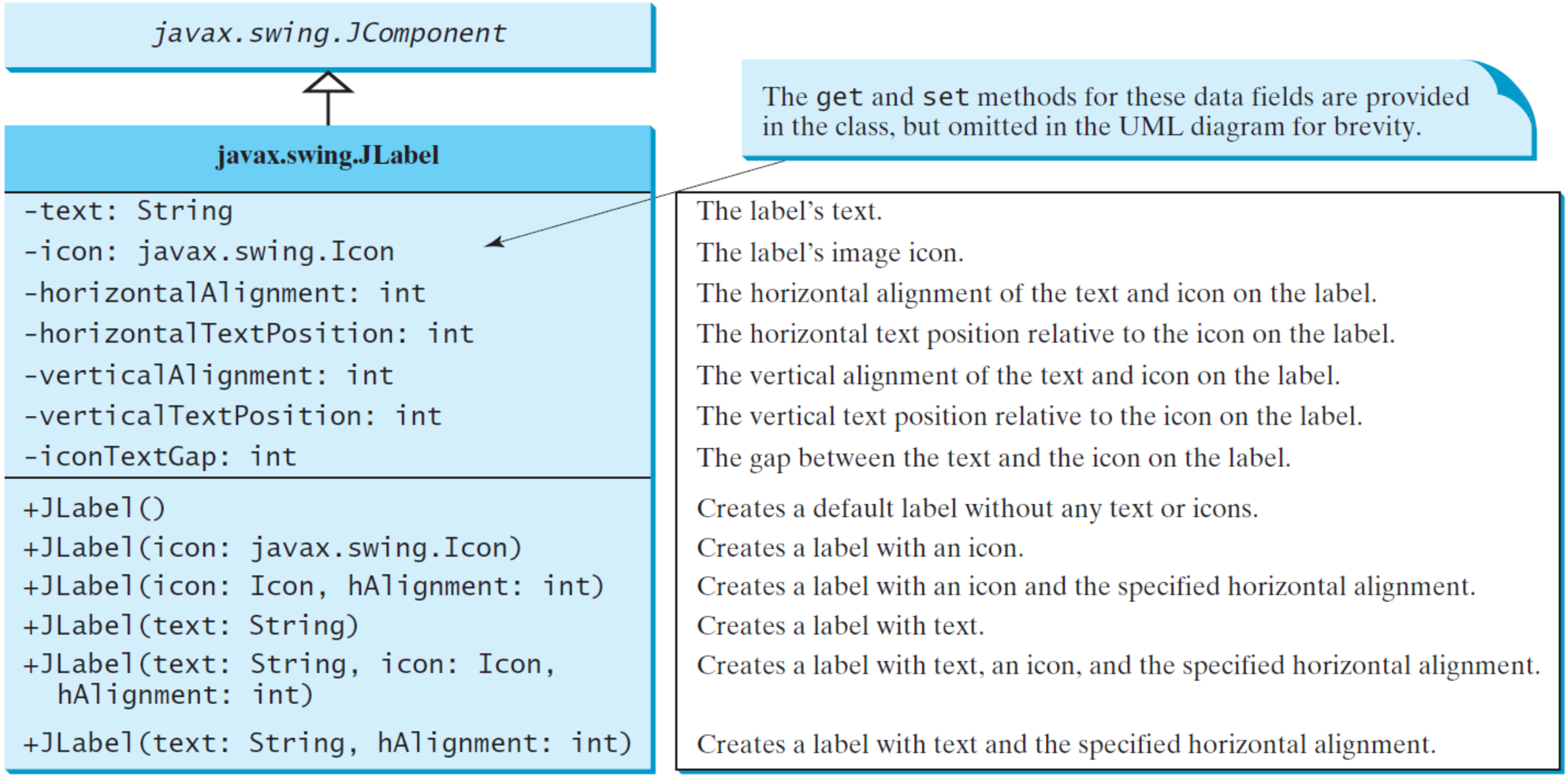
JButton Constructors

- To create a push button, use the **JButton** class.

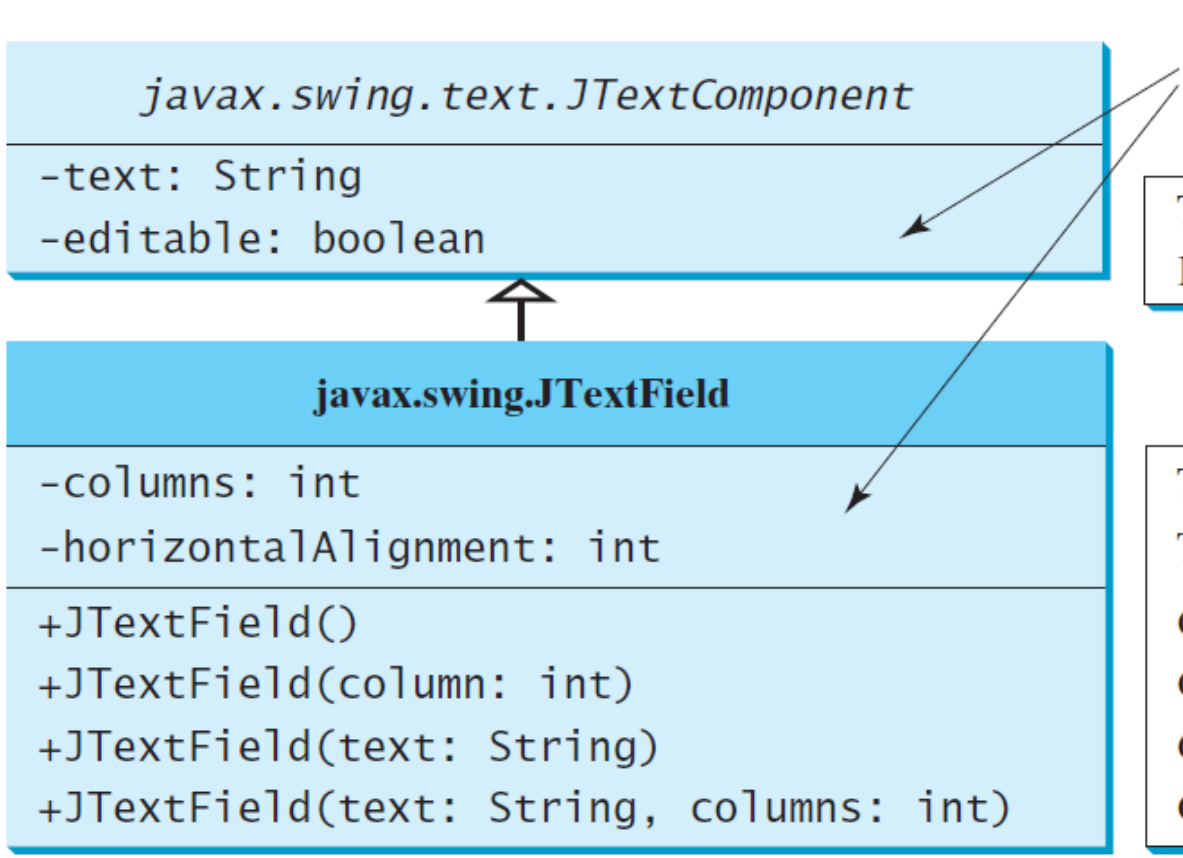


Creates a default button without any text or icons.	
Creates a button with an icon.	
Creates a button with text.	
Creates a button with text and an icon.	


javax.swing.JComponent
javax.swing.AbstractButton
-actionCommand: String
-text: String
-icon: javax.swing.Icon
-pressedIcon: javax.swing.Icon
-rolloverIcon: javax.swing.Icon
-mnemonic: int
-horizontalAlignment: int
-horizontalTextPosition: int
-verticalAlignment: int
-verticalTextPosition: int
-borderPainted: boolean
-iconTextGap: int
-selected: boolean
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
The action command of this button.
The button's text (i.e., the text label on the button).
The button's default icon. This icon is also used as the "pressed" and "disabled" icon if there is no pressed icon set explicitly.
The pressed icon (displayed when the button is pressed).
The rollover icon (displayed when the mouse is over the button).
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.
The horizontal alignment of the icon and text (default: CENTER).
The horizontal text position relative to the icon (default: RIGHT).
The vertical alignment of the icon and text (default: CENTER).
The vertical text position relative to the icon (default: CENTER).
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 are not painted.
The gap between the text and the icon on the button.
The state of the button. True if the check box or radio button is selected, false if not.


javax.swing.JComponent
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.
Creates a default label without any text or icons.
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.

JTextField Constructors



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 a 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` Methods

- Returns the string from the text field.

  `getText()`

- Puts the given string in the text field.

  `setText(String text)`

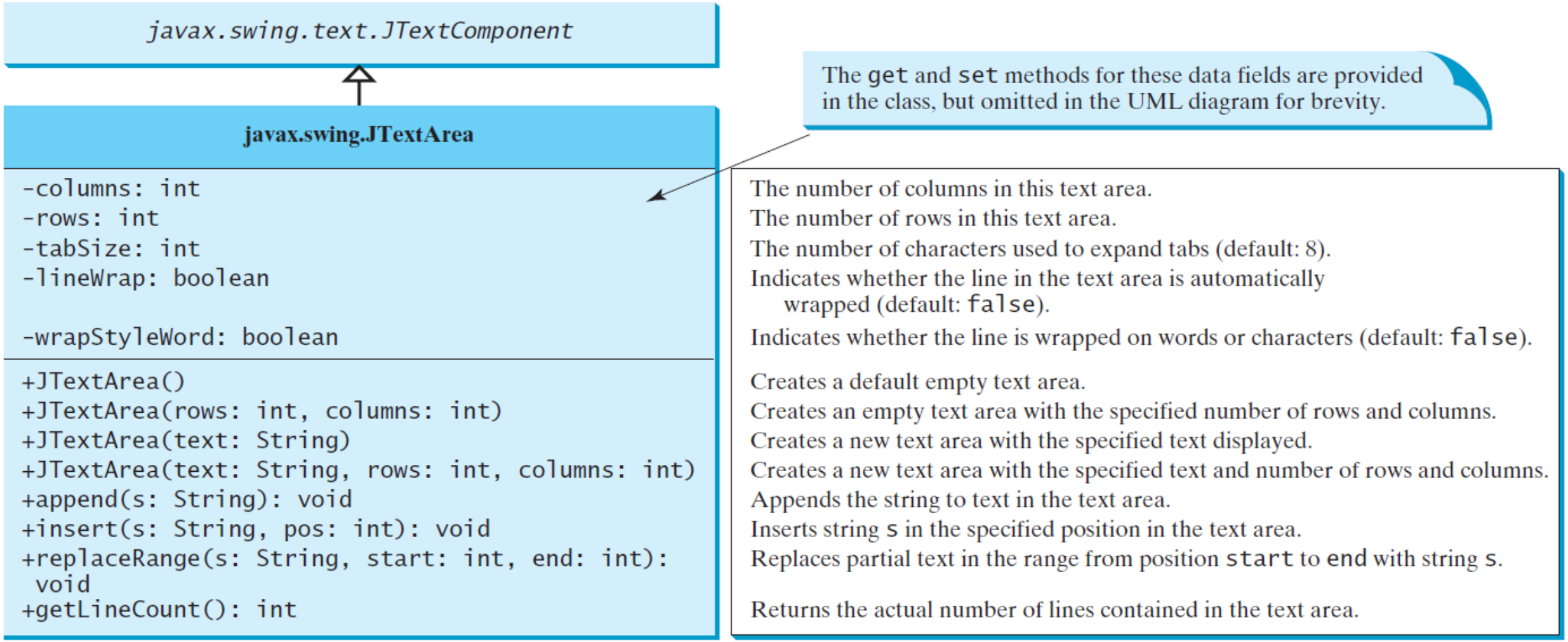
- Enables or disables the text field to be edited. By default, `editable` is `true`.

  `setEditable(boolean editable)`

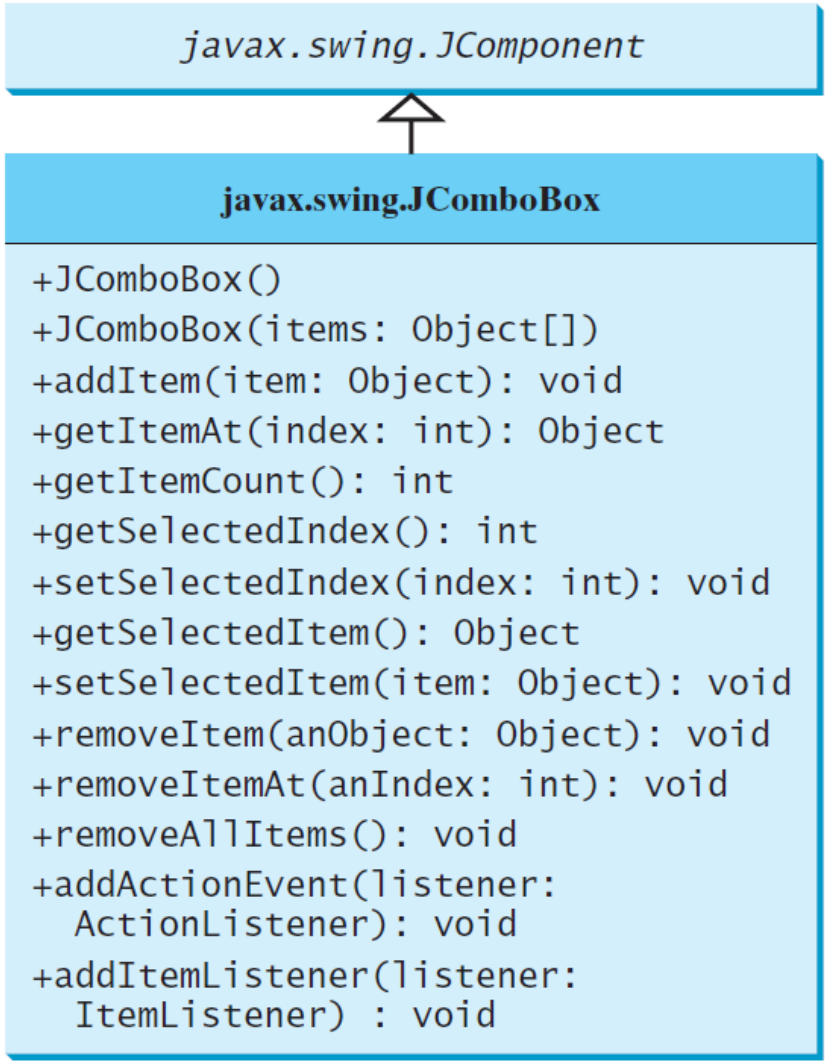
- Sets the number of columns in this text field. The length of the text field is changeable.

  `setColumns(int)`

JTextArea

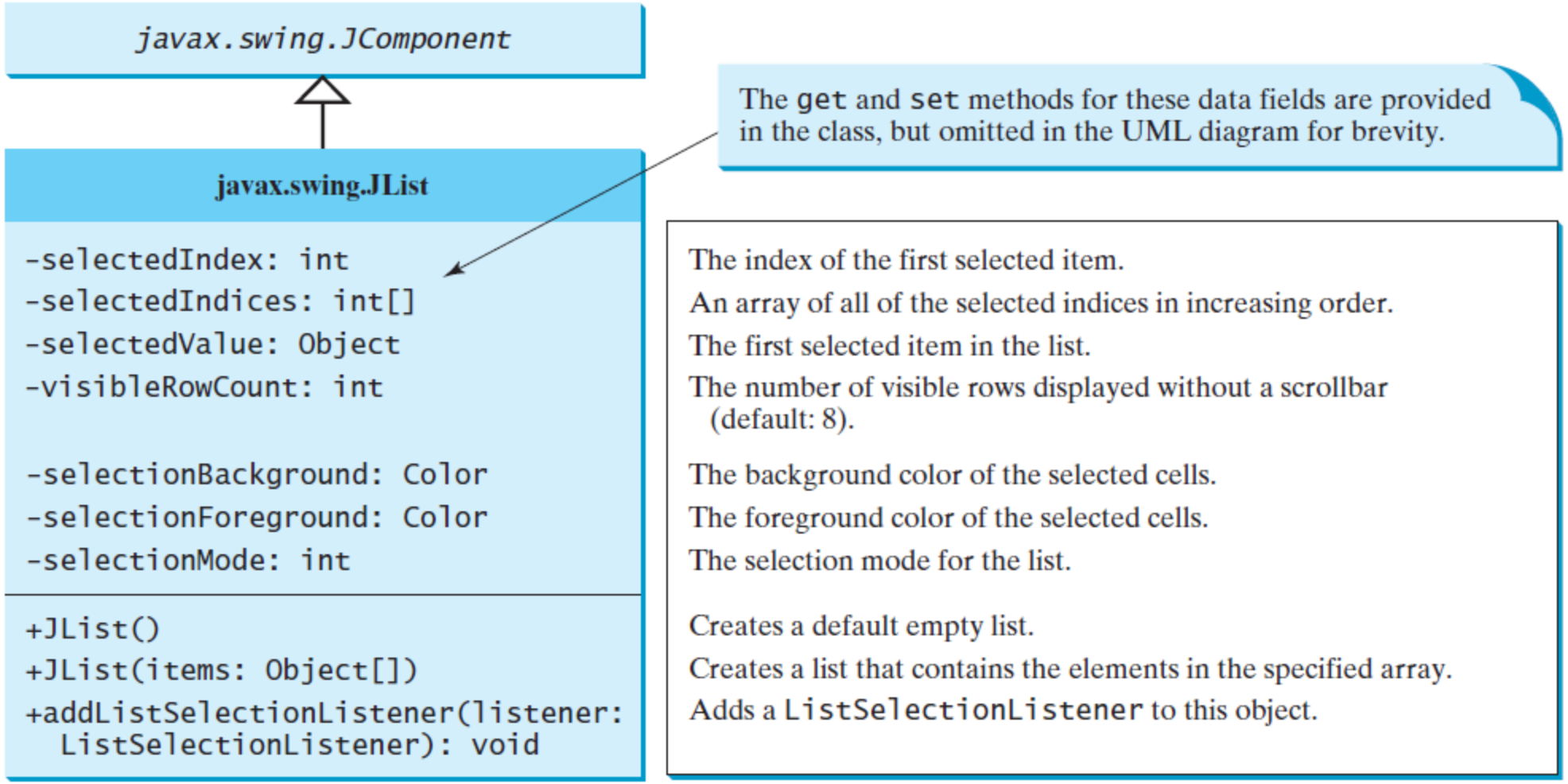

javax.swing.text.JTextComponent
javax.swing.JTextArea
-columns: int
-rows: int
-tabSize: int
-lineWrap: boolean
-wrapStyleWord: boolean
+JTextArea()
+JTextArea(rows: int, columns: int)
+JTextArea(text: String)
+JTextArea(text: String, rows: int, columns: int)
+append(s: String): void
+insert(s: String,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

JComboBox Methods



Creates a default empty combo box.	
Creates a combo box that contains the elements in the specified array.	
Adds an item to the combo box.	
Returns the item at the specified index.	
Returns the number of items in the combo box.	
Returns the index of the selected item.	
Sets the selected index in the combo box.	
Returns the selected item.	
Sets the selected item in the combo box.	
Removes an item from the item list.	
Removes the item at the specified index in the combo box.	
Removes all the items in the combo box.	
Adds an `ActionListener` for this object.	
Adds an `ItemListener` for this object.	

JList


javax.swing.JComponent
javax.swing.JList
-selectedIndex: int
-selectedIndices: int[]
-selectedValue: Object
-visibleRowCount: int
-selectionBackground: Color
-selectionForeground: Color
-selectionMode: int
+JList()
+JList(items: Object[])
+addListSelectionListener(listener: ListSelectionListener): void
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
The index of the first selected item.
An array of all of the selected indices in increasing order.
The first selected item in the list.
The number of visible rows displayed without a scrollbar (default: 8).
The background color of the selected cells.
The foreground color of the selected cells.
The selection mode for the list.
Creates a default empty list.
Creates a list that contains the elements in the specified array.
Adds a ListSelectionListener to this object.