

---

EE3206

Java Programming and  
Applications

Lecture 10

Event-Driven Programming

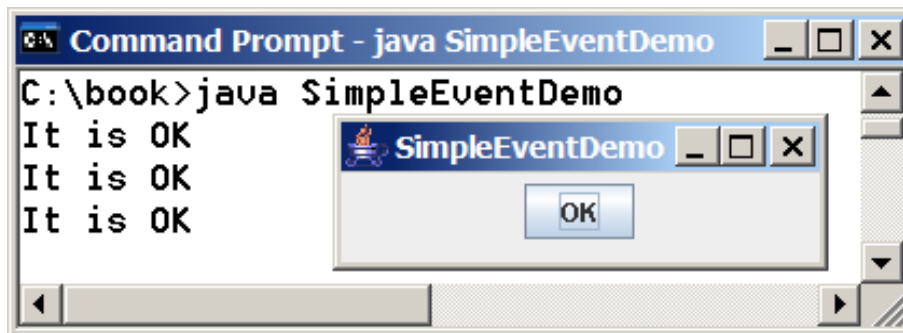
# Intended Learning Outcomes

---

- ▶ To explain the concept of event-driven programming.
- ▶ To understand events, event sources, and event classes.
- ▶ To declare listener classes and write the code to handle events.
- ▶ To write programs to deal with `ActionEvent`, `MouseEvent` and `KeyEvent`.
- ▶ To understand the use of Adpaters.
- ▶ To use the `Timer` class to control animations.
- ▶ To create GUI with various user-interface components.
- ▶ To create listeners for various types of events.

# Procedural vs. Event-Driven

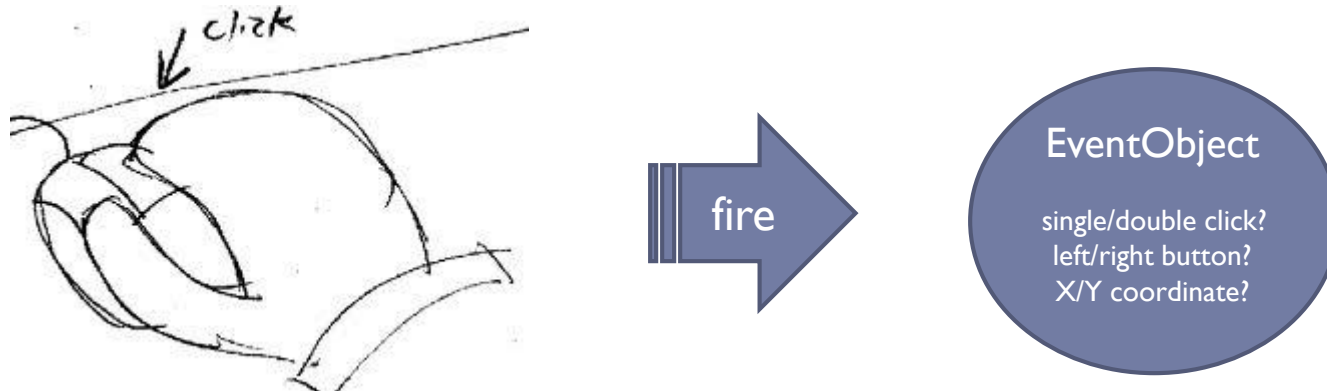
- ▶ Procedural programming – a list of procedures is executed in the order as they are listed.
- ▶ In event-driven programming, code is executed upon **activation of events**.
- ▶ The example displays a button in the frame. A message is displayed on the console when a button is clicked.



SimpleEventDemo

# What are Events?

- ▶ An event can be defined as a **type of signal** to the program that **something has happened**.
- ▶ The event is generated/fired by external user actions such as mouse movements, mouse clicks, and keystrokes, or by the operating system, such as a timer.

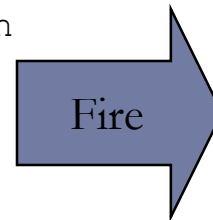
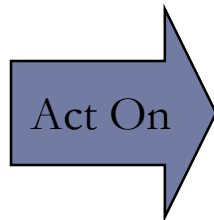


- ▶ An event object contains whatever properties are relevant to the event.
- ▶ You can identify the **firing source** object (e.g. a **JButton**) of the event using the **getSource()** instance method in the **EventObject** class. Its subclasses deal with special types of events, such as button actions, window events, component events, mouse movements, and keystrokes.

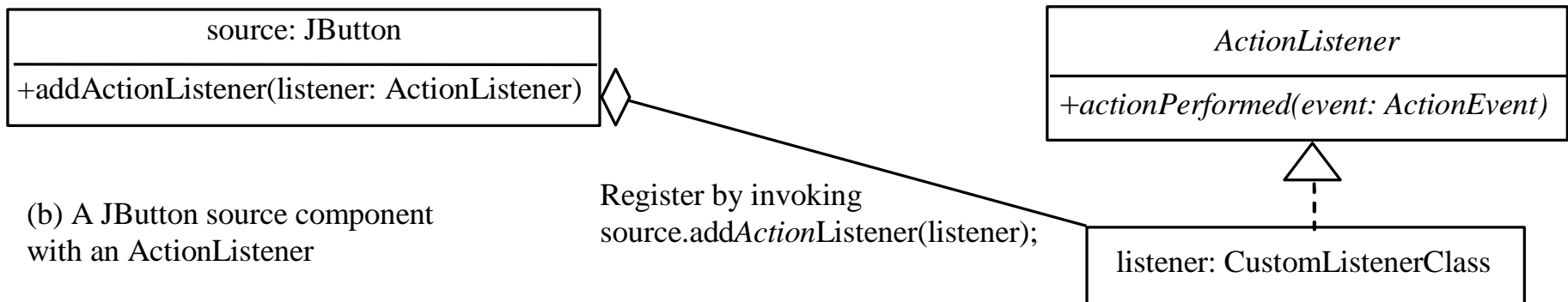
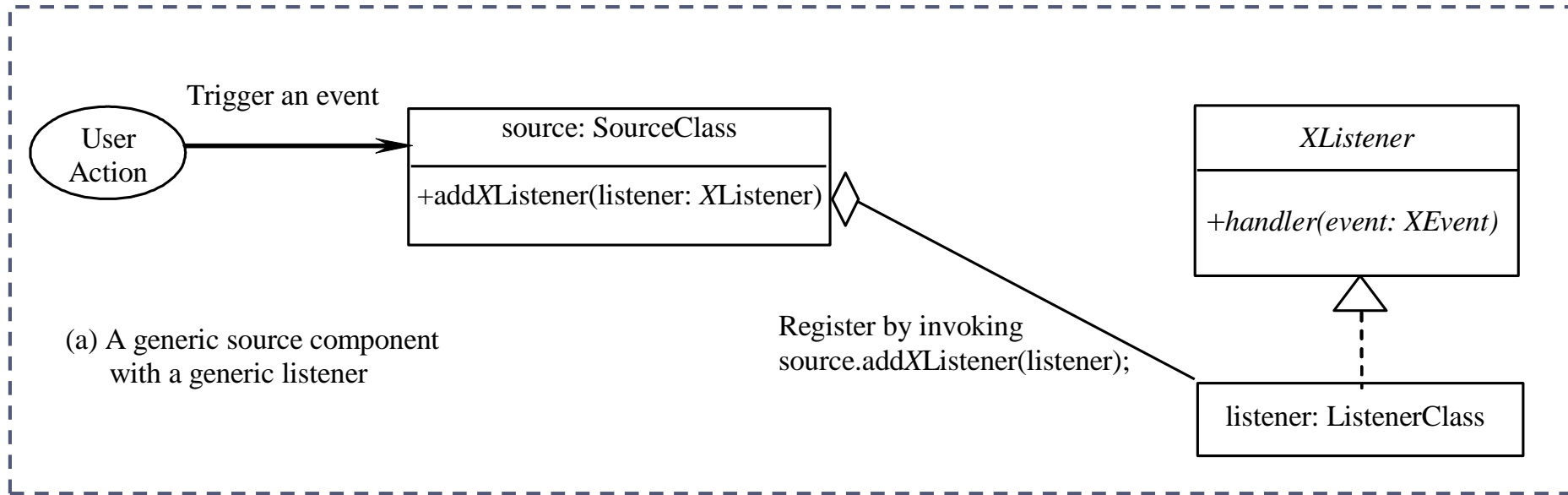
# Common User Actions

Table below lists external user actions, source objects, and event types generated.

User Action		Source Object		Event Type Generated
Click a button		JButton		ActionEvent
Click a check box		JCheckBox		ItemEvent, ActionEvent
Click a radio button		JRadioButton		ItemEvent, ActionEvent
Press return on a text field		JTextField		ActionEvent
Select a new item		JComboBox		ItemEvent, ActionEvent
Window opened, closed, etc.		Window		WindowEvent
Mouse pressed, released, etc.		Component		MouseEvent
Key released, pressed, etc.		Component		KeyEvent



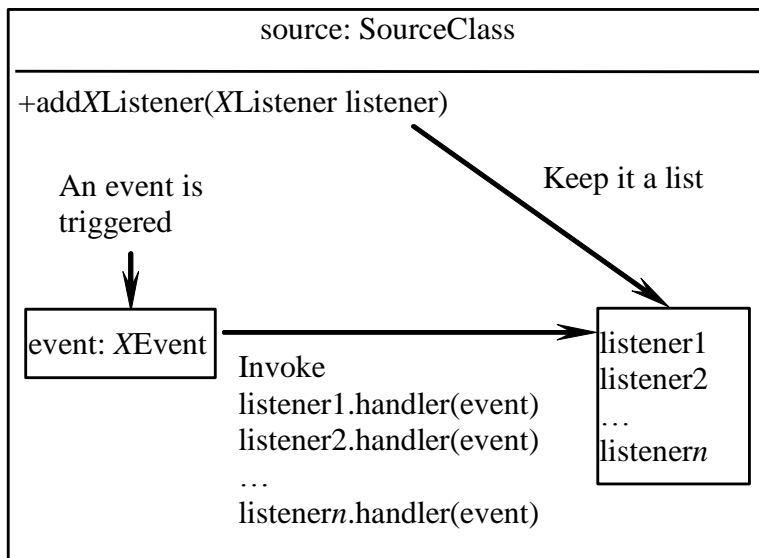
# The Delegation Model



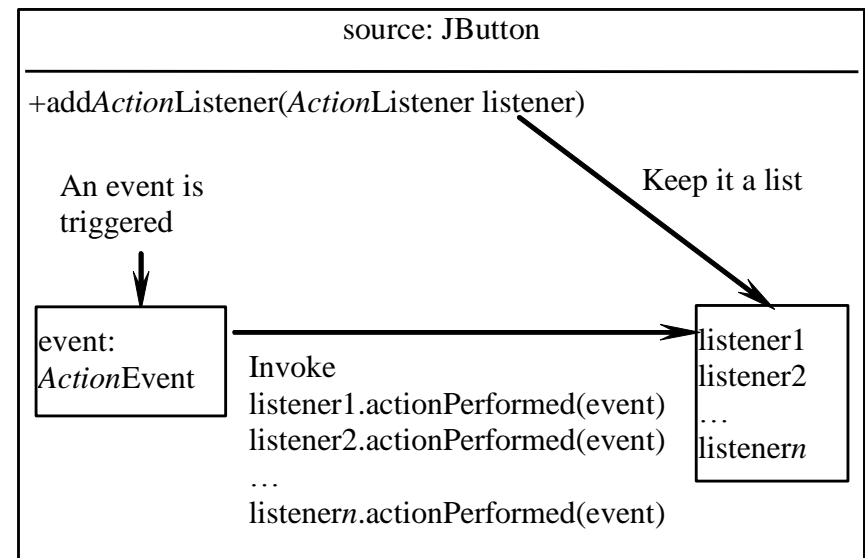
# The Delegation Model: Example

```
JButton jbt = new JButton("OK");  
ActionListener listener = new CustomListenerClass();  
jbt.addActionListener(listener);
```

## The internal function of a source component:



(a) Internal function of a generic source object



(b) Internal function of a JButton object

# Selected Event Handlers

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
MouseEvent	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseClicked(MouseEvent) mouseExited(MouseEvent) mouseEntered(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)

Handled by



# Inner Classes Listener

---

- ▶ A listener class is designed specifically to create a listener object for a GUI component (e.g., a button). It will not be shared by other applications. So, it is appropriate to define the listener class inside the frame class as an inner class.
  - ▶ The inner class is considered as a member of its outer class
  - ▶ The inner class *InnerClass* in *OuterClass* is compiled into *OuterClass\$InnerClass.class*
- ▶ Advantages:
  - ▶ In some applications, you can use an inner class to make programs more simple.
  - ▶ An inner class **can reference the data and methods defined in the outer class** in which it nests, so you do not need to pass the reference of the outer class to the constructor of the inner class (see next page).

# Inner Classes Pseudo Code

```
public class Test {  
    ...  
}  
  
public class A {  
    ...  
}
```

(a)

```
public class Test {  
    ...  
    // Inner class  
    public class A {  
        ...  
    }  
}
```

(b)

```
// OuterClass.java: inner class demo  
public class OuterClass {  
    private int data;  
  
    /** A method in the outer class */  
    public void m() {  
        // Do something  
    }  
  
    // An inner class  
    class InnerClass {  
        /** A method in the inner class */  
        public void mi() {  
            // Directly reference data and method  
            // defined in its outer class  
            data++;  
            m();  
        }  
    }  
}
```

(c)

SimpleEventDemoInnerClass

# Anonymous Class and Object

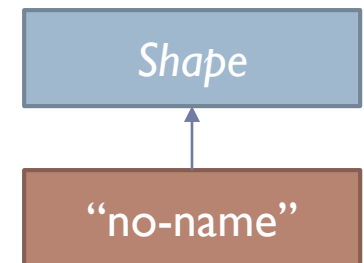
- ▶ Anonymous class/object is a mechanism for defining class/object **without explicitly stating the name** of the class/object. For example,

```
Circle c = new Circle();      // give an identifier c to the object explicitly
doSomething(c);
// create anonymous object and immediately pass to method
doSomething(new Circle());    // no name is given to the circle instance
```

- ▶ To create an anonymous class, you always **extend a superclass or implement an interface**. Any abstract methods must be implemented so as to instantiate the anonymous class.
- ▶ An anonymous class always uses the **no-arg constructor** from its superclass to create an instance. If an anonymous class implements an interface, the corresponding constructor is `Object()`.

```
Shape s = new Shape() {
    // Implement or override all methods
    // in superclass or interface
    // Other methods if necessary
};
```

\* s is an instance  
of “no-name” which  
extends Shape



# Anonymous Inner Classes

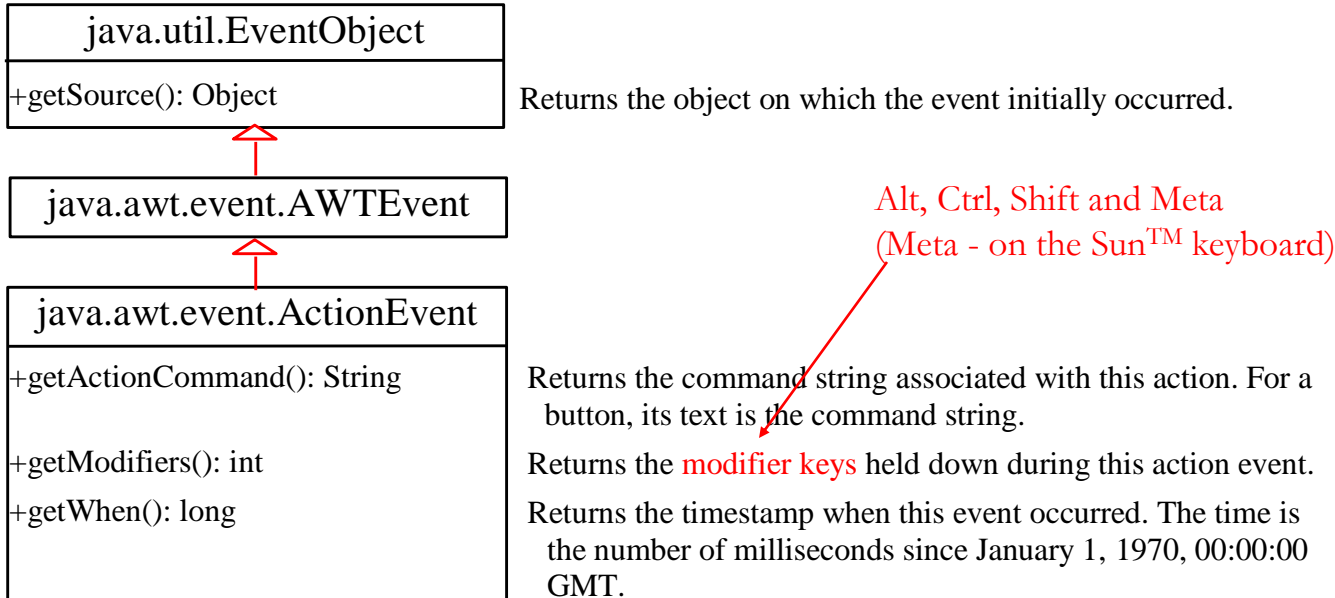
---

- ▶ Because without a name to reference, anonymous class must be **declared and instantiated at the same time**.
- ▶ Anonymous inner class is a special form of inner class that **does not have a name**.
- ▶ An anonymous inner class is compiled into a class named `OuterClassName$n.class`. For example, if the outer class `Test` has two anonymous inner classes, these two classes are compiled into `Test$1.class` and `Test$2.class`.
- ▶ Anonymous class is a convenient way for creating **one-time class** (e.g. listener class) without polluting the class namespace.



SimpleEventDemoAnonymousInnerClass

# Handling(ActionEvent)



- ▶ **Example 1:** Display two buttons OK and Cancel in the window. A message is displayed on the console to indicate which button is clicked, when a button is clicked. **TestActionEvent**
- ▶ **Example 2:** This example modifies TestActionEvent to add an additional listener for the action events on the buttons. When a button is clicked, both listeners respond to the action event. **TestMultipleListener**

# Handling Window Events

---

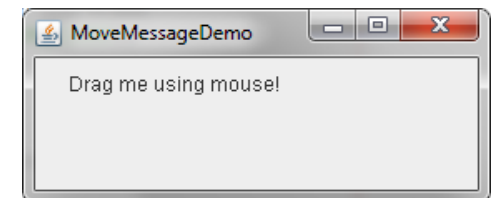
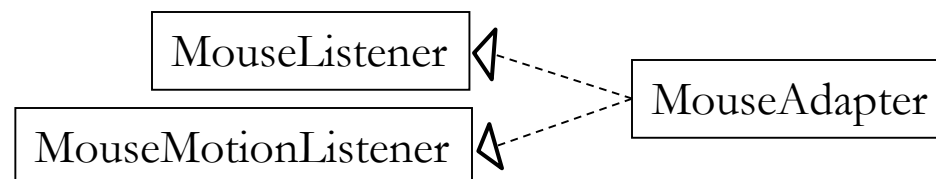
- ▶ Objective: Demonstrate handling the window events. Any subclass of the Window class can generate the following window events:
  - ▶ window opened, closing, closed, activated, deactivated, iconified, and deiconified.
- ▶ This program creates a frame, listens to the window events, and displays a message to indicate the occurring event.



TestWindowEvent

# Handling Mouse Events

- ▶ Java provides two listener interfaces to handle mouse events.
  - ▶ **MouseListener** - for actions such as when the mouse is pressed, released, entered, exited, or clicked
  - ▶ **MouseMotionListener** – for actions such as dragging or moving the mouse
- ▶ Because there are many abstract methods in these two interfaces, it is inconvenient to implement all handlers every time even you do not use all of them.
- ▶ Java provides a class **MouseAdapter** which receives both mouse events and mouse motion events. **The methods in this class are empty**; this class is provided as a convenience for easily creating listeners by extending this class and overriding only the methods of interest.



MoveMessageDemo

# Mouse Listeners

---

## *java.awt.event.MouseListener*

+*mousePressed(e: MouseEvent): void*

Invoked when the mouse button has been pressed on the source component.

+*mouseReleased(e: MouseEvent): void*

Invoked when the mouse button has been released on the source component.

+*mouseClicked(e: MouseEvent): void*

Invoked when the mouse button has been clicked (pressed and released) on the source component.

+*mouseEntered(e: MouseEvent): void*

Invoked when the mouse enters the source component.

+*mouseExited(e: MouseEvent): void*

Invoked when the mouse exits the source component.

## *java.awt.event.MouseMotionListener*

+*mouseDragged(e: MouseEvent): void*

Invoked when a mouse button is moved with a button pressed.

+*mouseMoved(e: MouseEvent): void*

Invoked when a mouse button is moved without a button pressed.



# MouseEvent

---

## java.awt.event.InputEvent

+getWhen(): long  
+isAltDown(): boolean  
+isControlDown(): boolean  
+isMetaDown(): boolean  
+isShiftDown(): boolean

Returns the timestamp when this event occurred.

Returns whether or not the Alt modifier is down on this event.

Returns whether or not the Control modifier is down on this event.

Returns whether or not the Meta modifier is down on this event

Returns whether or not the Shift modifier is down on this event.



## java.awt.event.MouseEvent

+getButton(): int  
+getClickCount(): int  
+getPoint(): java.awt.Point  
+getX(): int  
+getY(): int

Indicates which mouse button has been clicked.

Returns the number of mouse clicks associated with this event.

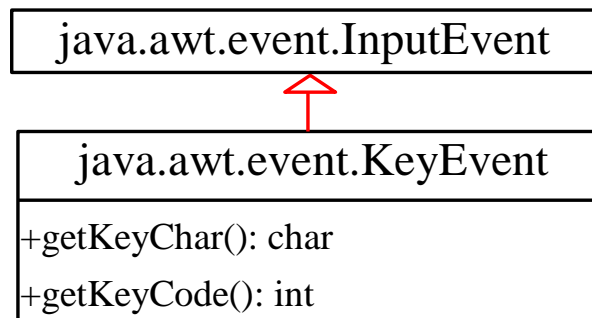
Returns a Point object containing the x and y coordinates.

Returns the x-coordinate of the mouse point.

Returns the y-coordinate of the mouse point.

# Handling Keyboard Events

- ▶ To process a keyboard event, use the following handlers in the **KeyListener** interface:
  - ▶ `keyPressed(KeyEvent e)` - Called when a key is pressed.
  - ▶ `keyReleased(KeyEvent e)` - Called when a key is released.
  - ▶ `keyTyped(KeyEvent e)` - Called when a key is pressed and then released.
  - ▶ Corresponding Adapter – **KeyAdapter**



Keys:

Home	VK_HOME
End	VK_END
Page Up	VK_PGUP
Page Down	VK_PGDN
etc...	

Defined as some  
constant values in  
the class

Returns the character associated with the key in this event.

Returns the integer **keyCode** associated with the key in this event.

KeyEventDemo

# The Timer Class

- ▶ Some non-GUI components can fire events. The `javax.swing.Timer` class is a source component that fires an `ActionEvent` at a predefined rate.
- ▶ The Timer class can be used to control animations. For example, you can use it to display a moving message.

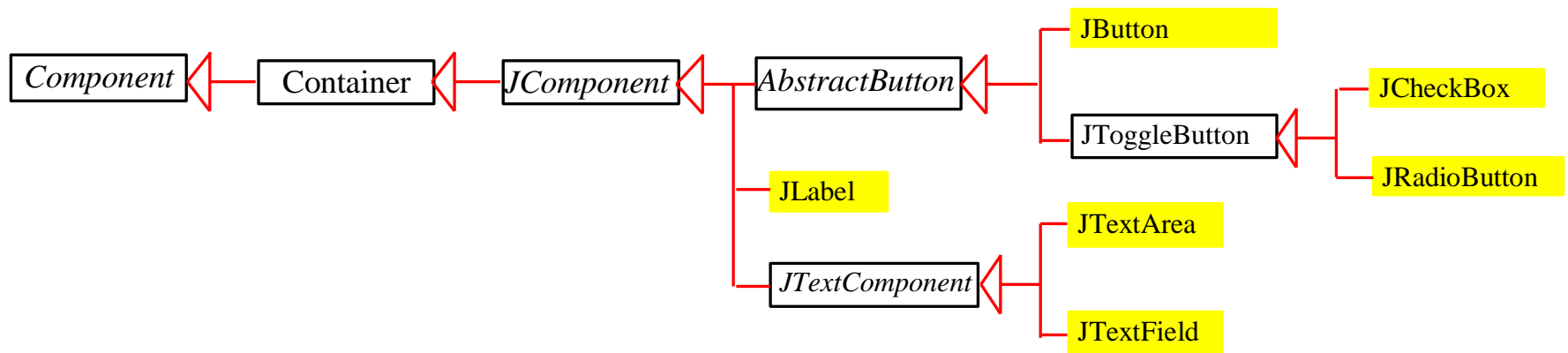
javax.swing.Timer	
+Timer(delay: int, listener: ActionListener)	Creates a Timer with a specified delay in milliseconds and an ActionListener.
+addActionListener(listener: ActionListener): void	Adds an ActionListener to the timer.
+start(): void	Starts this timer.
+stop(): void	Stops this timer.
+setDelay(delay: int): void	Sets a new delay value for this timer.

AnimationDemo

# Introduction to Other Useful Components

# Frequently Used Components

- Introduces the frequently used GUI components

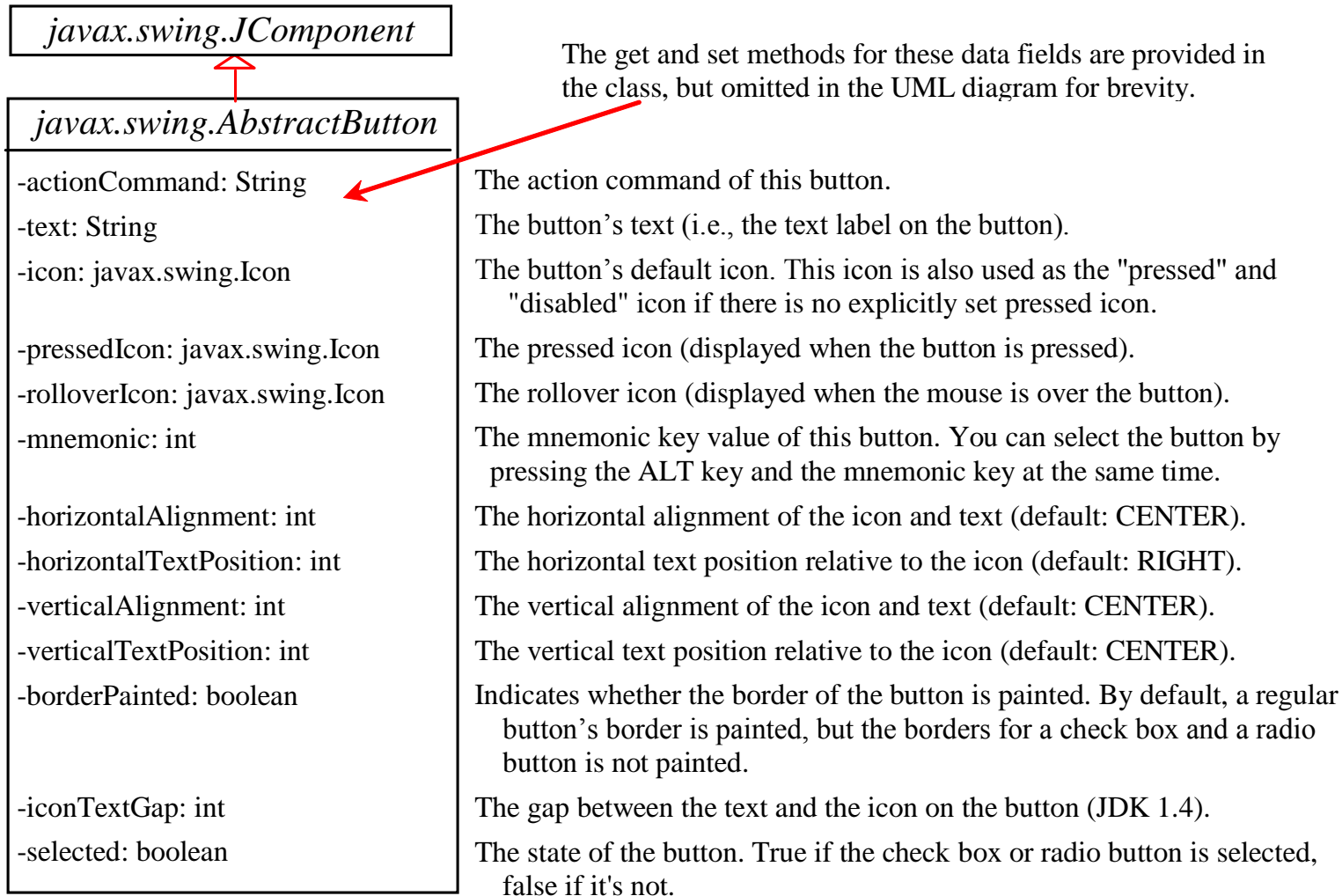


# Types of Button

---

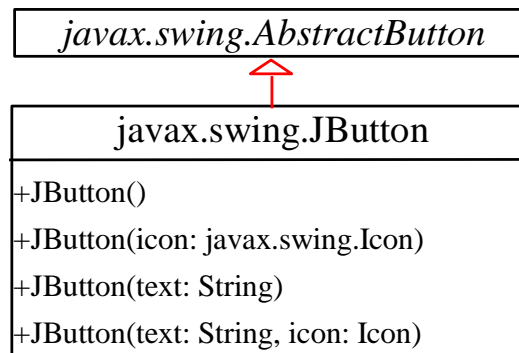
- ▶ A button is a component that triggers an action event when clicked.
- ▶ Swing provides regular buttons, toggle buttons (two-state button), check box buttons, and radio buttons.
- ▶ The common features of these buttons are generalized in `javax.swing.AbstractButton`.

# AbstractButton



# JButton Constructors and Properties

- ▶ JButton inherits AbstractButton and provides several constructors to create buttons.
- ▶ Constructor
  - ▶ JButton()
  - ▶ JButton(String text)
  - ▶ JButton(String text, Icon icon)
  - ▶ JButton(Icon icon)
- ▶ Properties
  - ▶ text
  - ▶ icon
  - ▶ Mnemonic (shortcut key)
  - ▶ 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



```
jbt.setPressedIcon(imageIcon);
```

TestButtonIcons

# Horizontal Alignments

---

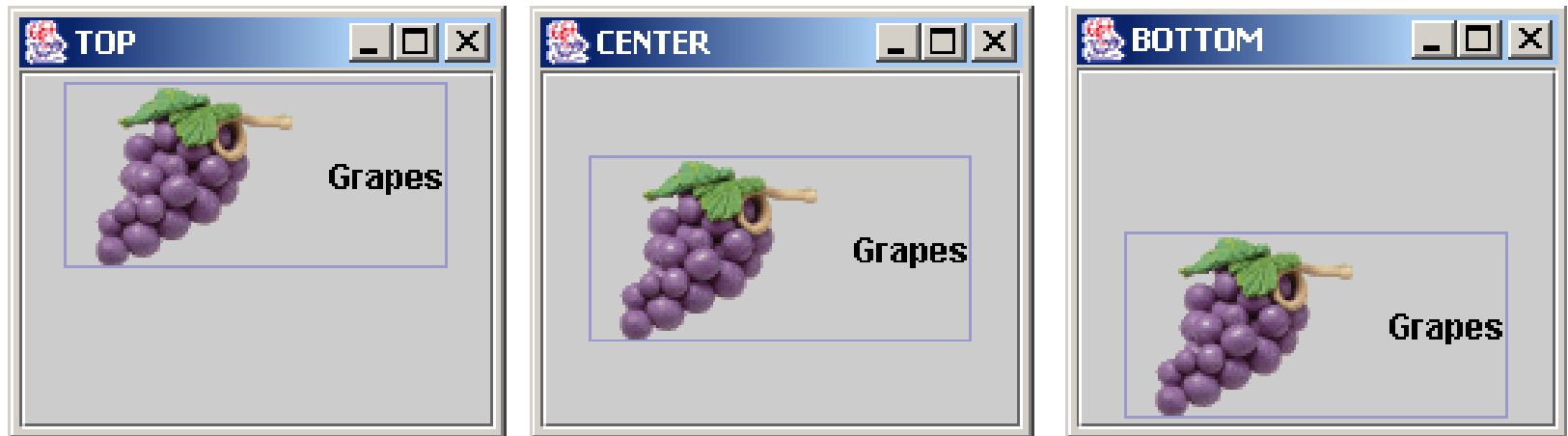
- ▶ Horizontal alignment specifies how the **icon and text** are placed horizontally on a button.
- ▶ You can set the horizontal alignment using one of the three constants: LEFT, CENTER, RIGHT.
- ▶ These constants are inherited from the *SwingConstants* interface.
- ▶ The default horizontal alignment is SwingConstants.CENTER.



```
jbt.setHorizontalAlignment(SwingConstants.CENTER);
```

# Vertical Alignments

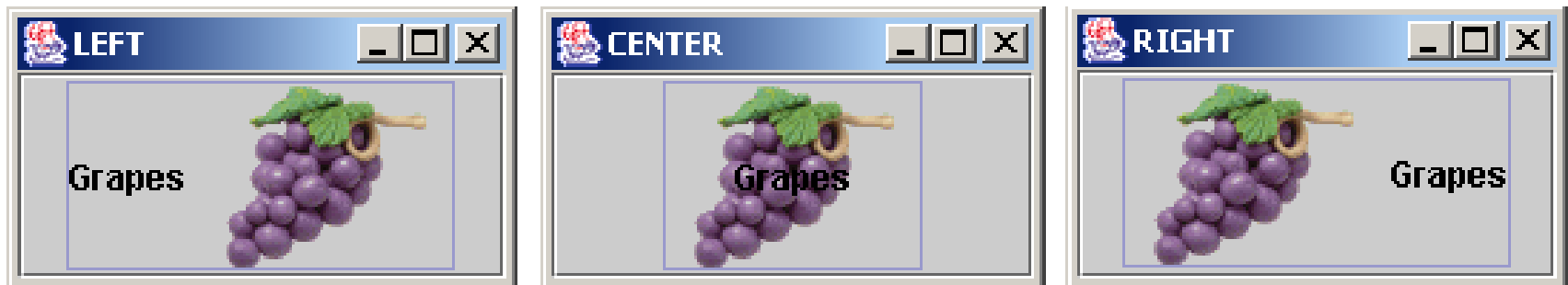
- ▶ Vertical alignment specifies how the icon and text are placed vertically on a button. You can set the vertical alignment using one of the three constants: `TOP`, `CENTER`, `BOTTOM`.
- ▶ The default vertical alignment is `SwingConstants.CENTER`.



```
jbt.setVerticalAlignment(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 three constants: LEFT, CENTER, RIGHT.
- ▶ The default horizontal text position is `SwingConstants.RIGHT`.



```
jbt.setHorizontalTextPosition(SwingConstants.CENTER);
```

# Vertical Text Positions

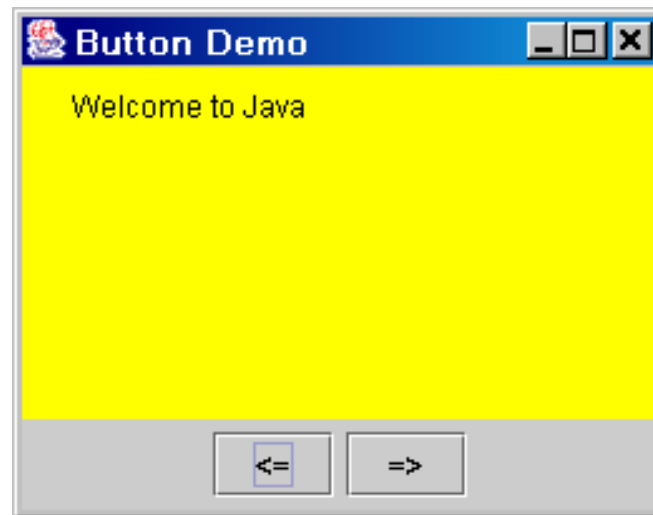
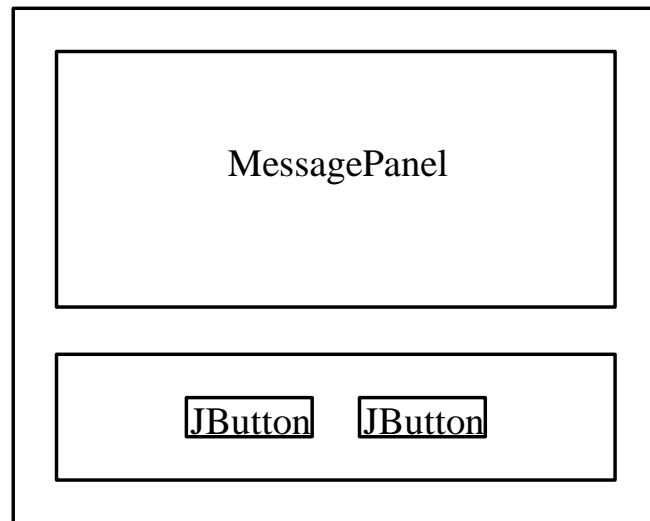
- ▶ Vertical text position specifies the vertical position of the **text relative to the icon**.
- ▶ You can set the vertical text position using one of the three constants: TOP, CENTER, BOTTOM.
- ▶ The default vertical text position is `SwingConstants.CENTER`.



```
jbt.setVerticalTextPosition(SwingConstants.CENTER);
```

# Example: Using Buttons

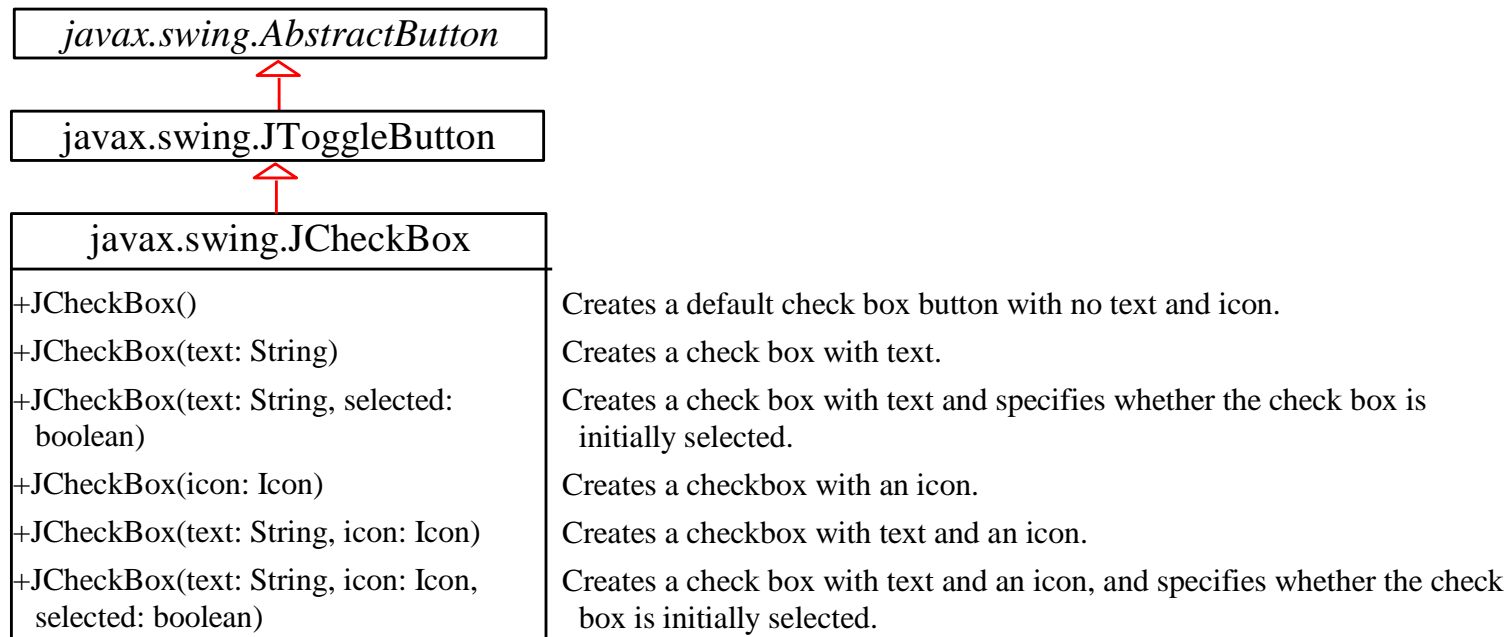
- ▶ Write a program that displays a message on a panel and uses two buttons, `<=` and `=>`, to move the message on the panel to the left or right.



ButtonDemo

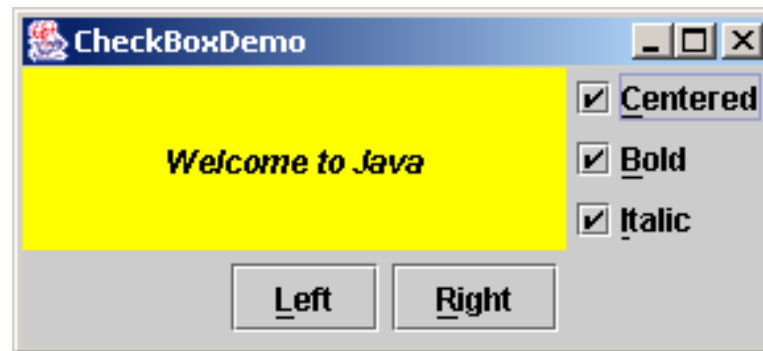
# JCheckBox

- ▶ 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.
- ▶ To detect if a box is checked, use *isSelected()*



# Example: Using Check Boxes

- ▶ Add three check boxes named **Centered**, **Bold**, and **Italic** into the previous example to let the user specify whether the message is centered, bold, or italic.

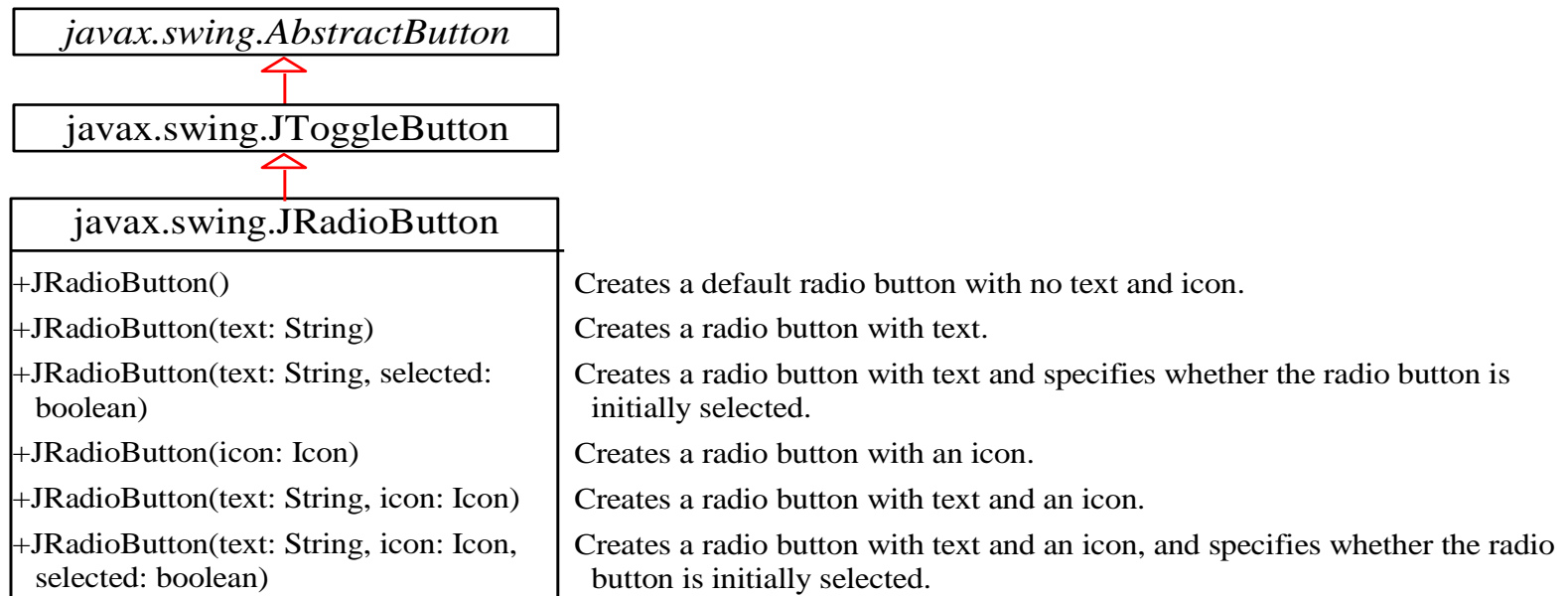


CheckBoxDemo



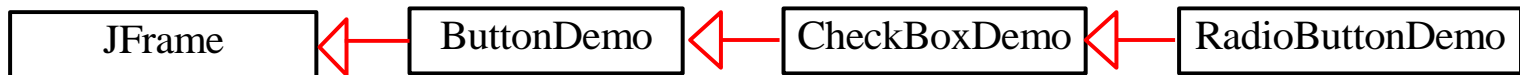
# JRadioButton

- ▶ Radio buttons are variations of check boxes. They are often used in the group, where only one button is checked at a time. They can be grouped with the following code:
  - ▶ `ButtonGroup btg = new ButtonGroup();`
  - ▶ `btg.add(jrb1);`
  - ▶ `btg.add(jrb2);`



# Example: Using Radio Buttons

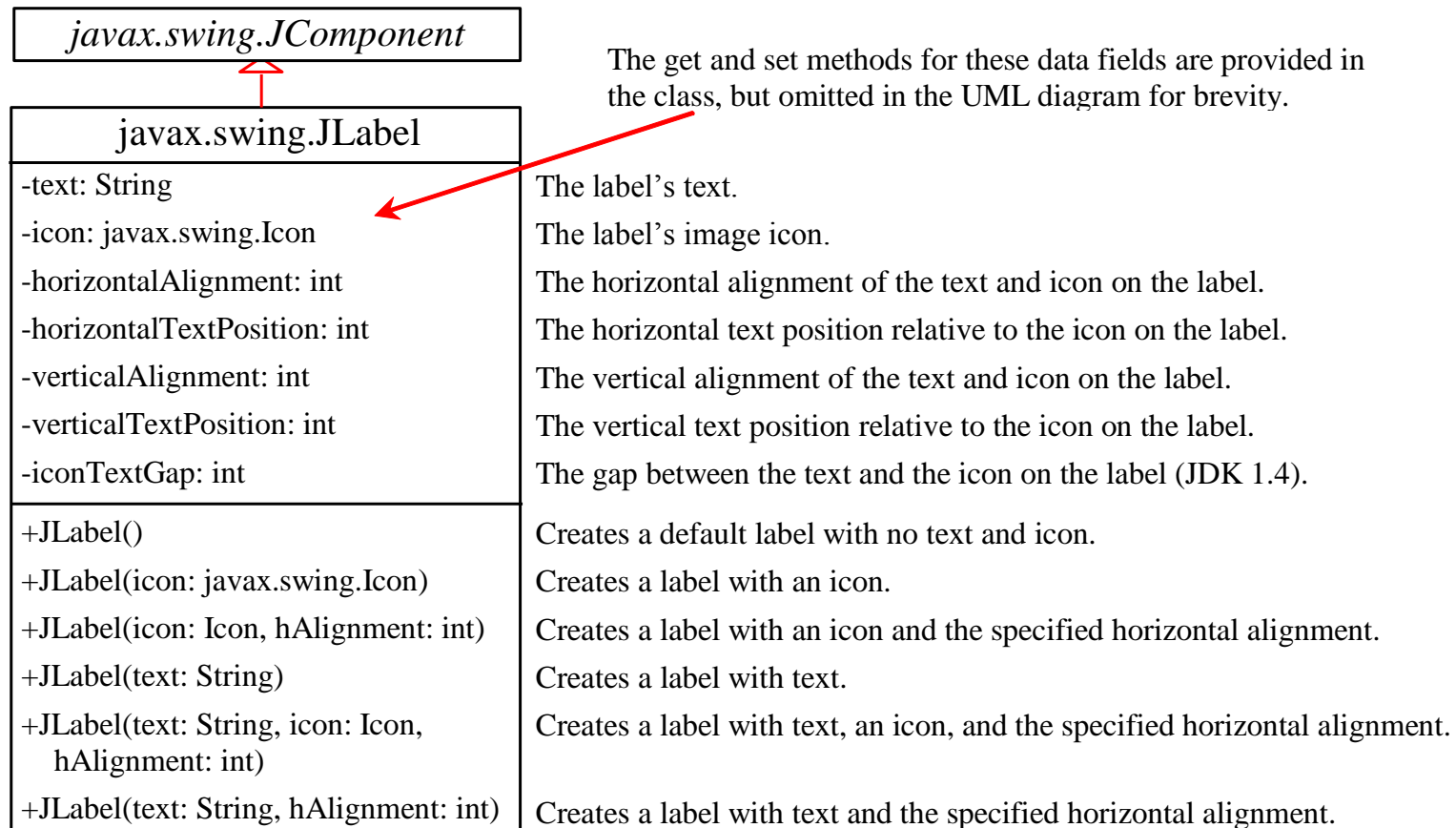
- ▶ Add three radio buttons named Red, Green, and Blue into the preceding example to let the user choose the color of the message.



RadioButtonDemo

# JLabel

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



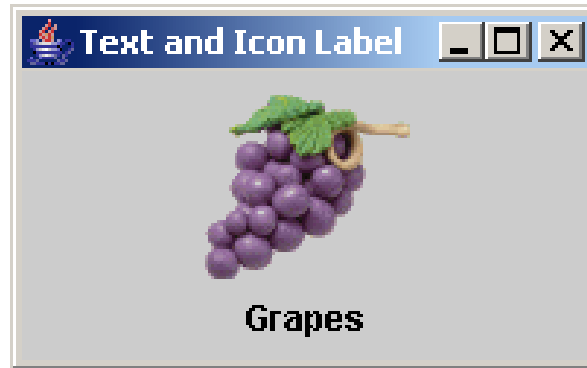
# JLabel Constructors & Properties

---

- ▶ 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 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

---



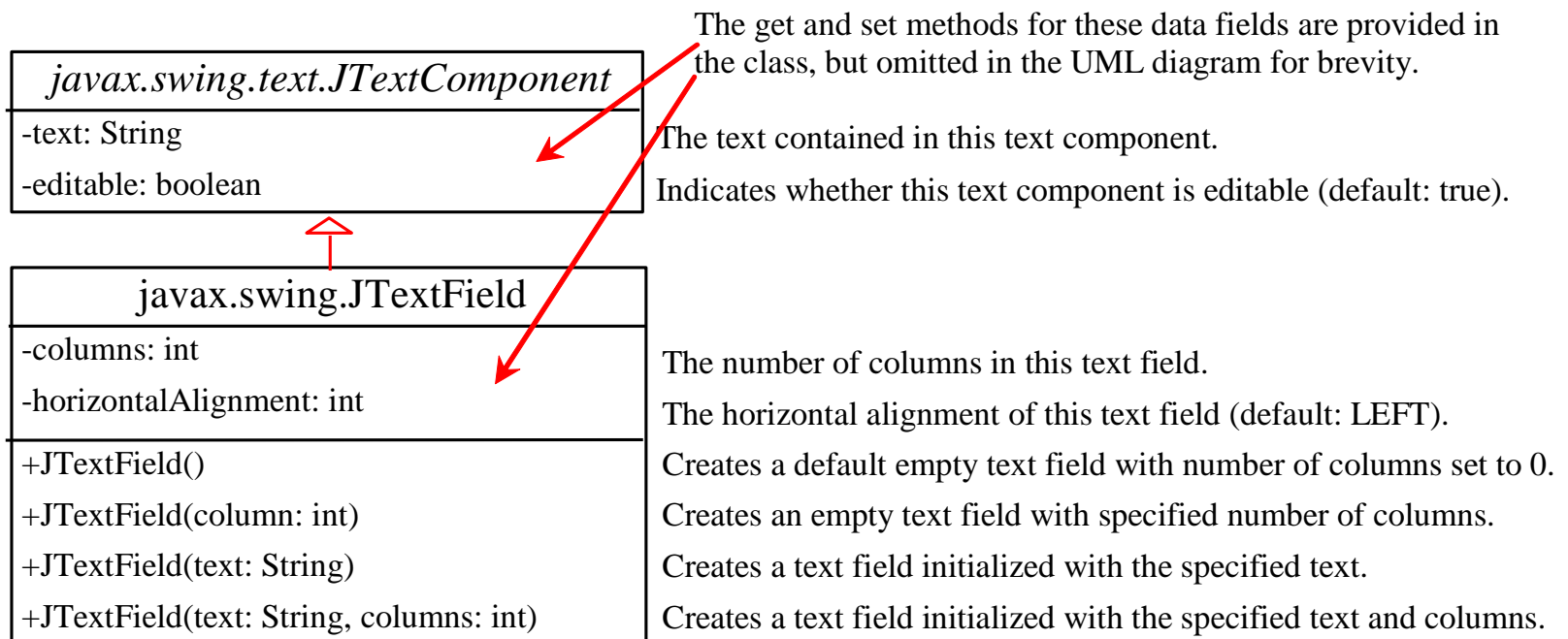
```
// Create an image icon from image file
ImageIcon icon = new ImageIcon("image/grapes.gif");

// 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 type in variable data (such as a name or a description).



# JTextField Constructors & Properties

---

## ▶ Constructors

- ▶ `TextField(int columns)`
  - ▶ Creates an empty text field with the specified number of columns.
- ▶ `TextField(String text)`
  - ▶ Creates a text field initialized with the specified text.
- ▶ `TextField(String text, int columns)`
  - ▶ Creates a text field initialized with the specified text and the column size.

## ▶ Properties

- ▶ `text`
- ▶ `horizontalAlignment`
- ▶ `editable`
- ▶ `columns`

# JTextField Methods

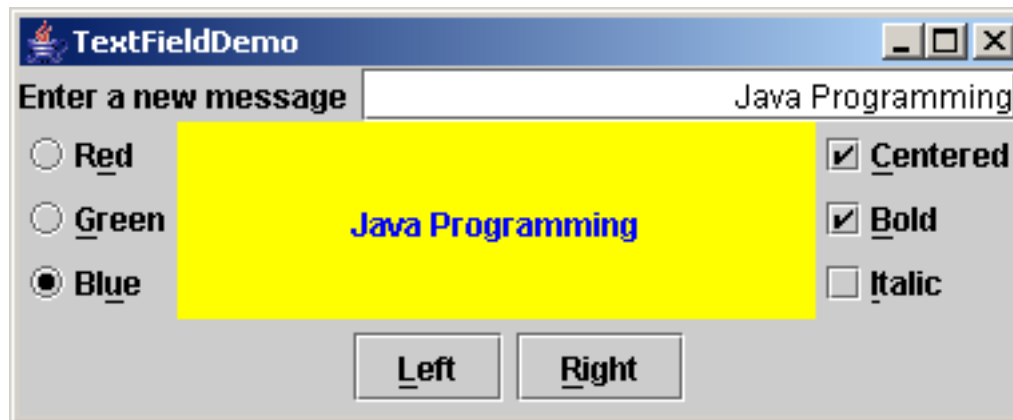
---

- ▶ **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.



# Example: Using Text Fields

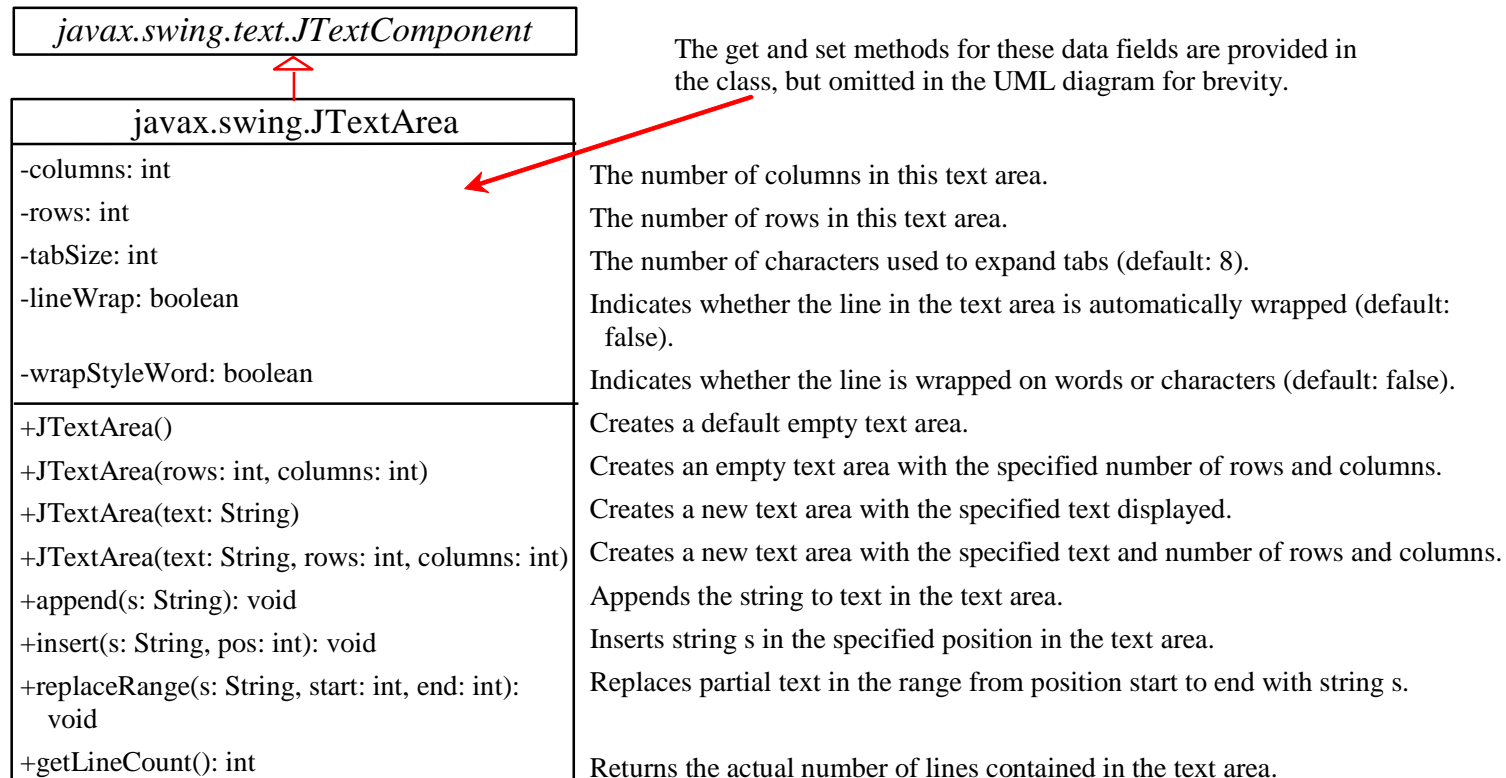
- ▶ Add a text field to the preceding example to let the user set a new message.



TextFieldDemo

# JTextArea

- ▶ If you want to let the user enter **multiple lines of text**, you cannot use text fields unless you create several of them. The solution is to use `JTextArea`, which enables the user to enter multiple lines of text.



# JTextArea Constructors & Properties

---

## ▶ Constructors

- ▶ `JTextArea(int rows, int columns)`
  - ▶ Creates a text area with the specified number of rows and columns.
- ▶ `JTextArea(String s, int rows, int columns)`
  - ▶ Creates a text area with the initial text and the number of rows and columns specified.

## ▶ Properties

- ▶ `text`
- ▶ `editable`
- ▶ `columns`
- ▶ `lineWrap`
- ▶ `wrapStyleWord`
- ▶ `rows`
- ▶ `lineCount`
- ▶ `tabSize`



TestTextArea