Event-Driven Programming in Java

Event-Driven Programming - Motivation

Write a program that animates a rising flag, as shown.

- How do you accomplish the task?

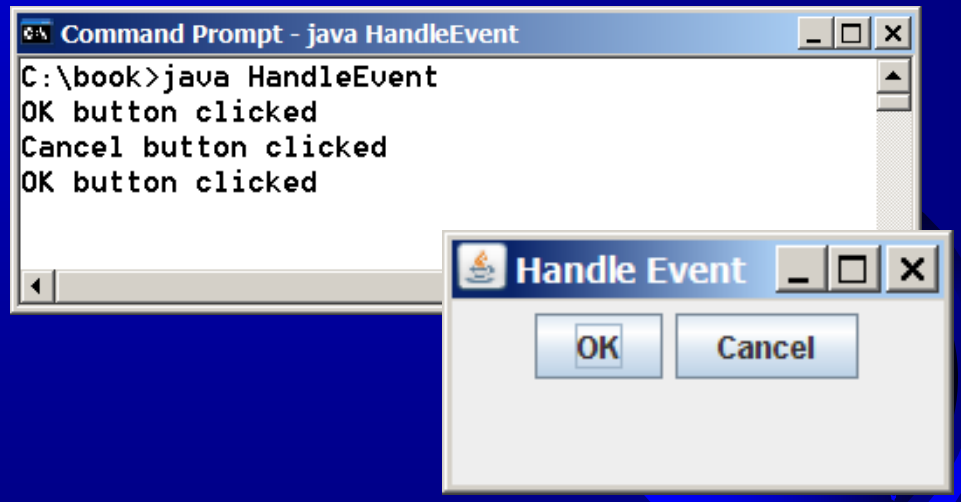
There are several solutions to this problem. An effective way to solve it is to use event-driven programming.

Procedural vs. Event-Driven Programming

- *Procedural programming* is executed in procedural order.
- In event-driven programming, code is executed upon activation of events.

Taste of Event-Driven Programming (Example – *HandleEvent*)

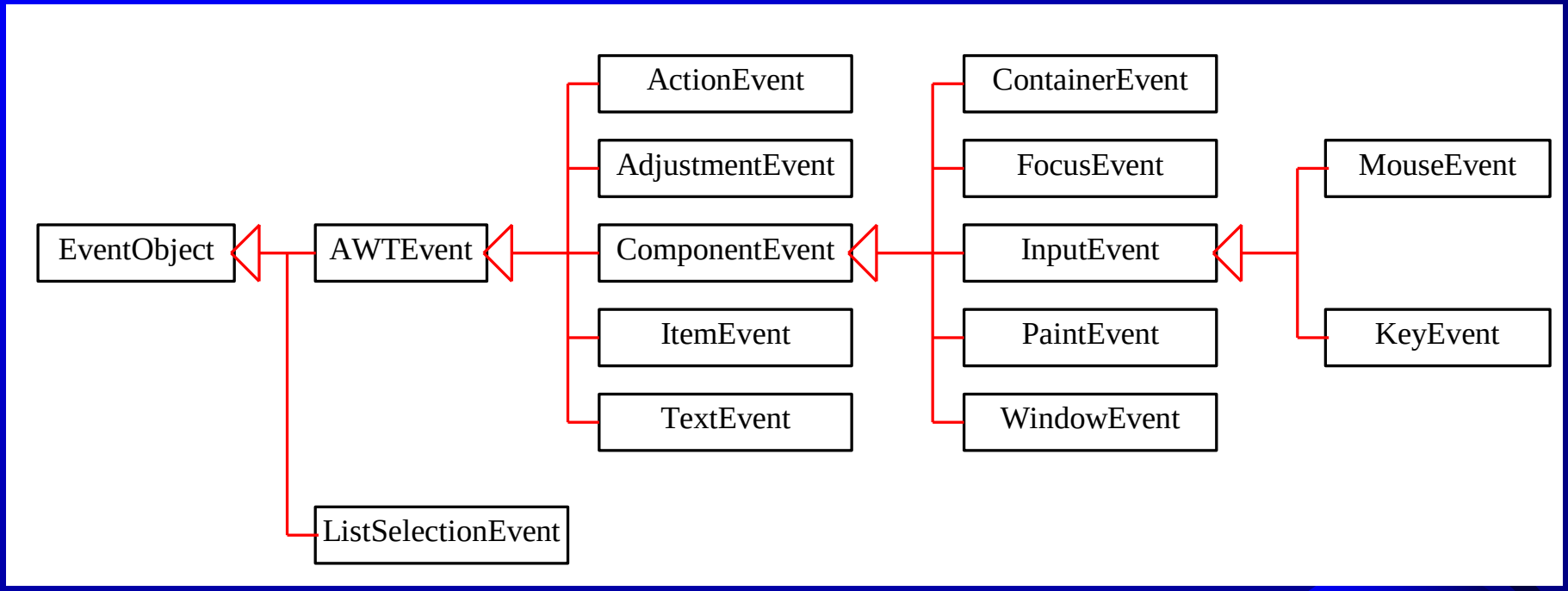
- The example displays a button in the frame.
- A message is displayed on the console when a button is clicked.



Events

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

Event Classes



Event Information

- An event object contains properties pertinent to the event.

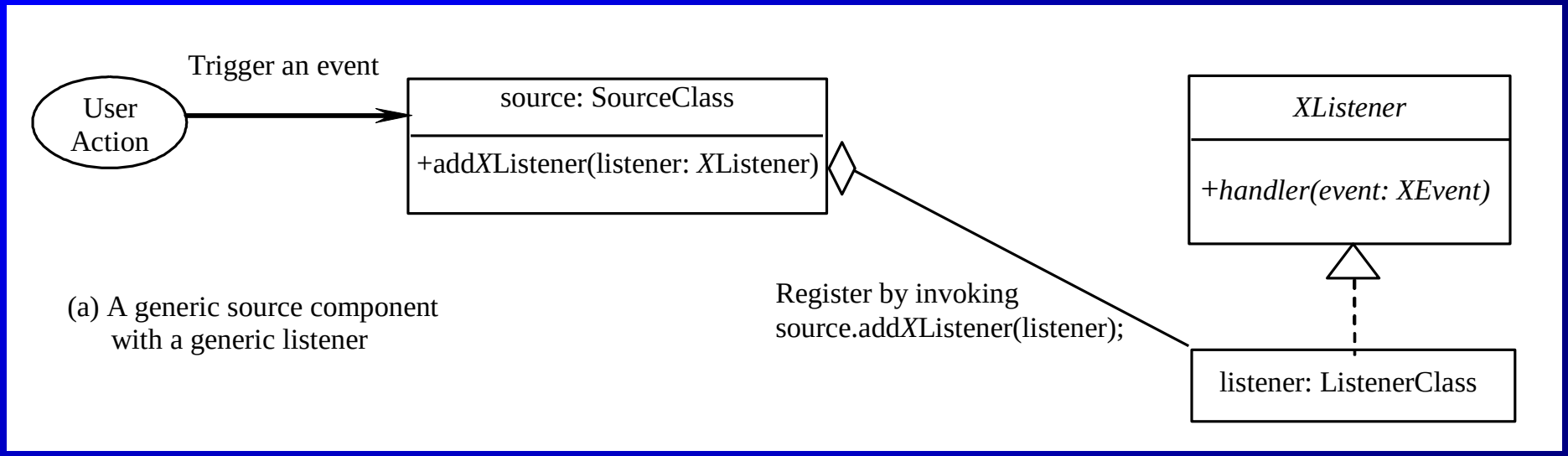
- You can identify the source object of the event using the getSource() instance method in the EventObject class.

- The subclasses of EventObject deal with special types of events, such as button actions, window events, component events, mouse movements, and keystrokes.

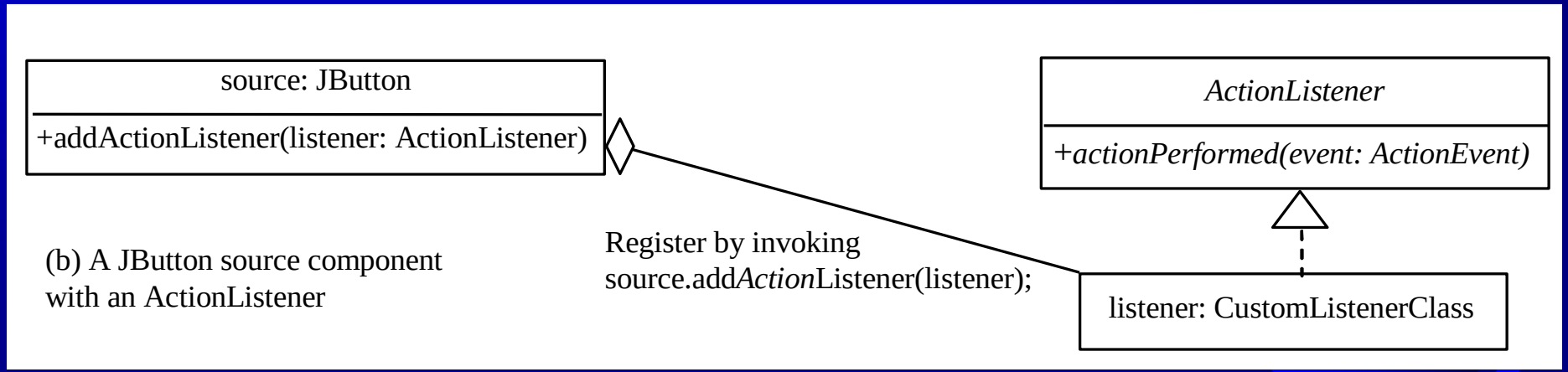
Selected User Actions

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

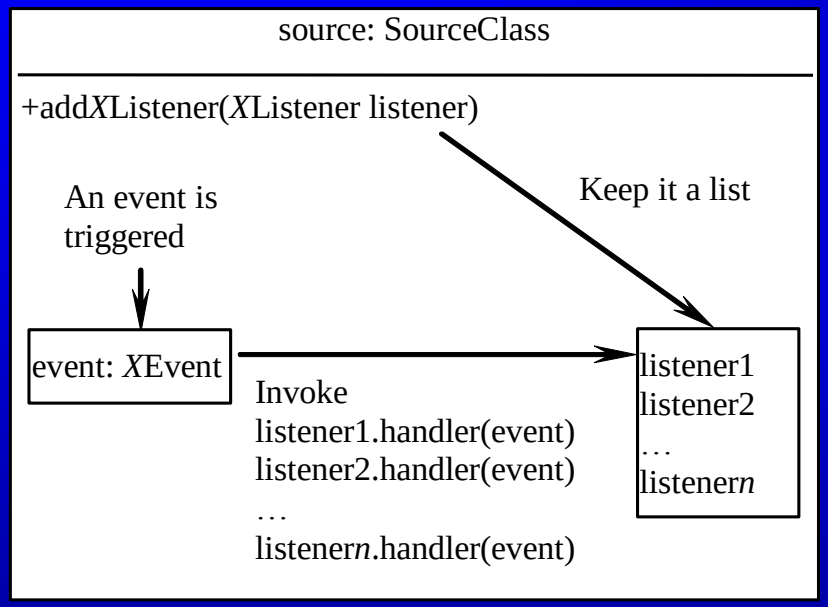


(a) A generic source component with a generic listener

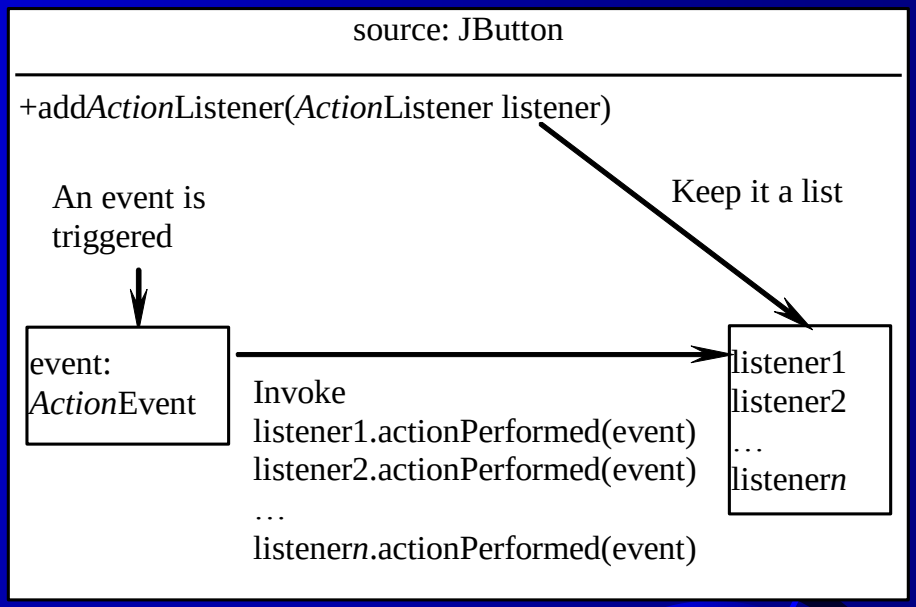


(b) A JButton source component with an ActionListener

Internal Function of a Source Component



(a) Internal function of a generic source object



(b) Internal function of a JButton object

The Delegation Model: Example

```
JButton jbt = new JButton("OK");

ActionListener listener = new OKListener();

jbt.addActionListener(listener);
```

Selected Event Handlers

Event Class	***Listener Interface***	***Listener Methods (Handlers)***
`ActionEvent`	`ActionListener`	`actionPerformed(ActionEvent)`
`ItemEvent`	`ItemListener`	`itemStateChanged(ItemEvent)`

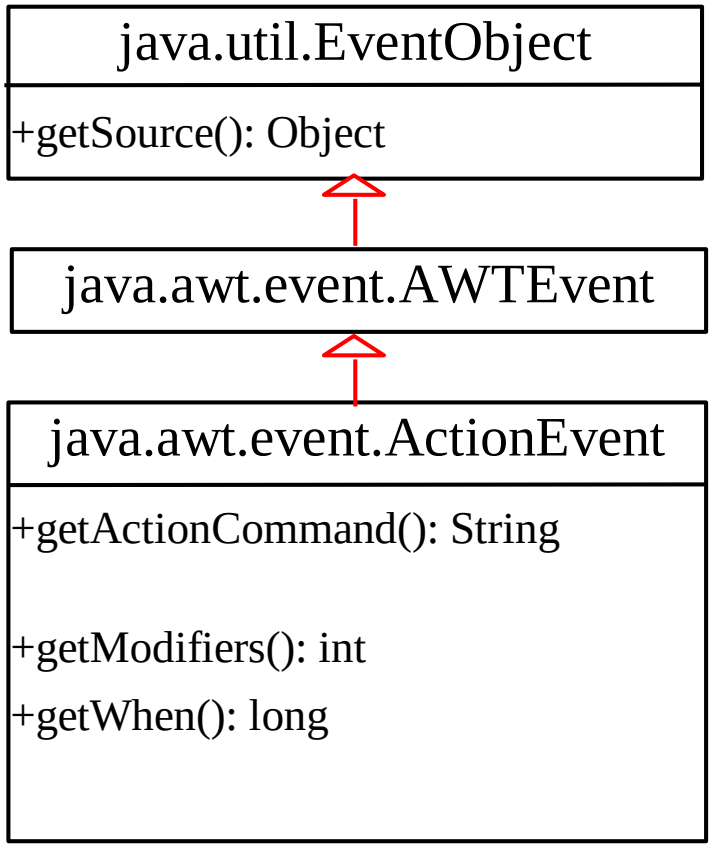
`WindowEvent`	`WindowListener`	`windowClosing(WindowEvent)`
		`windowOpened(WindowEvent)`
		`windowIconified(WindowEvent)`
		`windowDeiconified(WindowEvent)`
		`windowClosed(WindowEvent)`
		`windowActivated(WindowEvent)`
		`windowDeactivated(WindowEvent)`

`ContainerEvent`	`ContainerListener`	`componentAdded(ContainerEvent)` `componentRemoved(ContainerEvent)`

`MouseEvent`	`MouseListener`	`mousePressed(MouseEvent)`
		`mouseReleased(MouseEvent)`
		`mouseClicked(MouseEvent)`
		`mouseExited(MouseEvent)`
		`mouseEntered(MouseEvent)`

`KeyEvent`	`KeyListener`	`keyPressed(KeyEvent)`
		`keyReleased(KeyEvent)`
		`keyTypeed(KeyEvent)`

java.awt.event.ActionEvent (Example – *SimpleEventDemo*)



Returns the object on which the event initially occurred.

Returns the command string associated with this action. For a button, its text is the command string.

Returns the modifier keys held down during this action event.

Returns the timestamp when this event occurred. The time is the number of milliseconds since January 1, 1970, 00:00:00 GMT.

Inner Class Listeners

- 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.

Inner Classes (Example – *ShowInnerClass*)

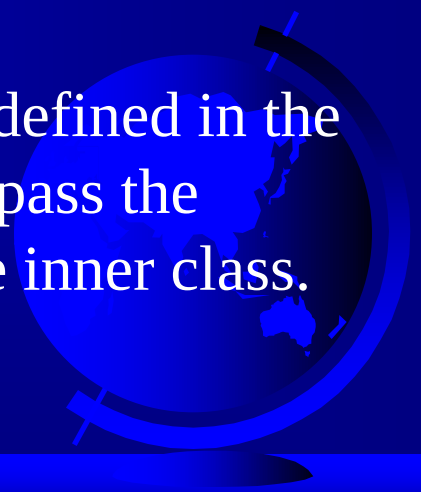
- Inner class

  - A class is a member of another class.

- Advantages

  - In some applications, you can use an inner class to make programs 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.



Inner Classes, cont.

```
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)

Inner Classes (cont.)

- Inner classes can make programs simple and concise.
- An inner class supports the work of its containing outer class and is compiled into a class named *OuterClassName*$*InnerClassName*.class.
- For example, the inner class InnerClass in OuterClass is compiled into *OuterClass*$*InnerClass*.class.

Inner Classes (cont.)

- An inner class can be declared public, protected, or private subject to the same visibility rules applied to a member of the class.
- An inner class can be declared static. A static inner class can be accessed using the outer class name. A static inner class cannot access nonstatic members of the outer class.

Revisiting SimpleEventDemo Using Inner Classes (Example – *SimpleEventDemoInnerClass*)

Anonymous Inner Classes

- An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit extends or implements clause.
- An anonymous inner class must implement all the abstract methods in the superclass or in the interface.
- An anonymous inner class always uses the no-arg constructor from its superclass to create an instance. If an anonymous inner class implements an interface, the constructor is Object().
- 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 Inner Classes (cont.)

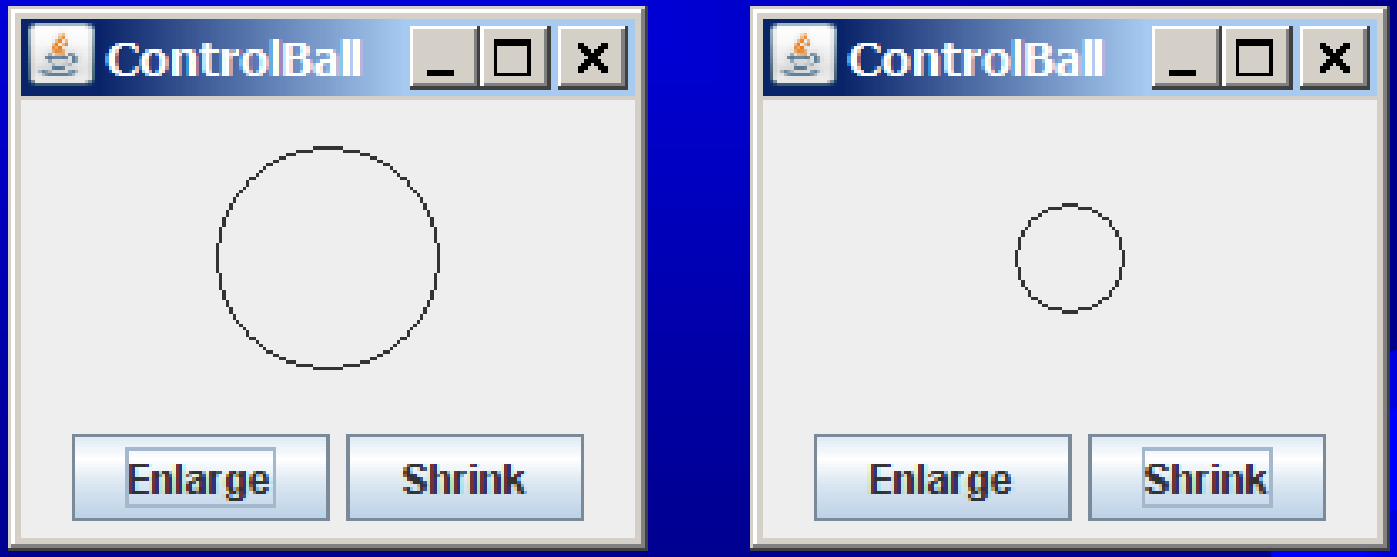
- Inner class listeners can be shortened using anonymous inner classes.
- An *anonymous inner class* is an inner class without a name. It combines declaring an inner class and creating an instance of the class in one step.
- An anonymous inner class is declared as follows:

```
new SuperClassName/InterfaceName() {
  // Implement or override methods in superclass or interface
  // Other methods if necessary
}
```

Revisiting SimpleEventDemo Using Anonymous Inner Classes (Example – *AnonymousListenerDemo*)

Example: Controlling Balls

- Objective: Create a *ControlBall* program using buttons to enlarge or shrink a ball.



Graphical User Interface

- A graphical user interface (GUI) makes a system user-friendly and easy to use.

- Creating a GUI requires creativity and knowledge of how GUI components work.

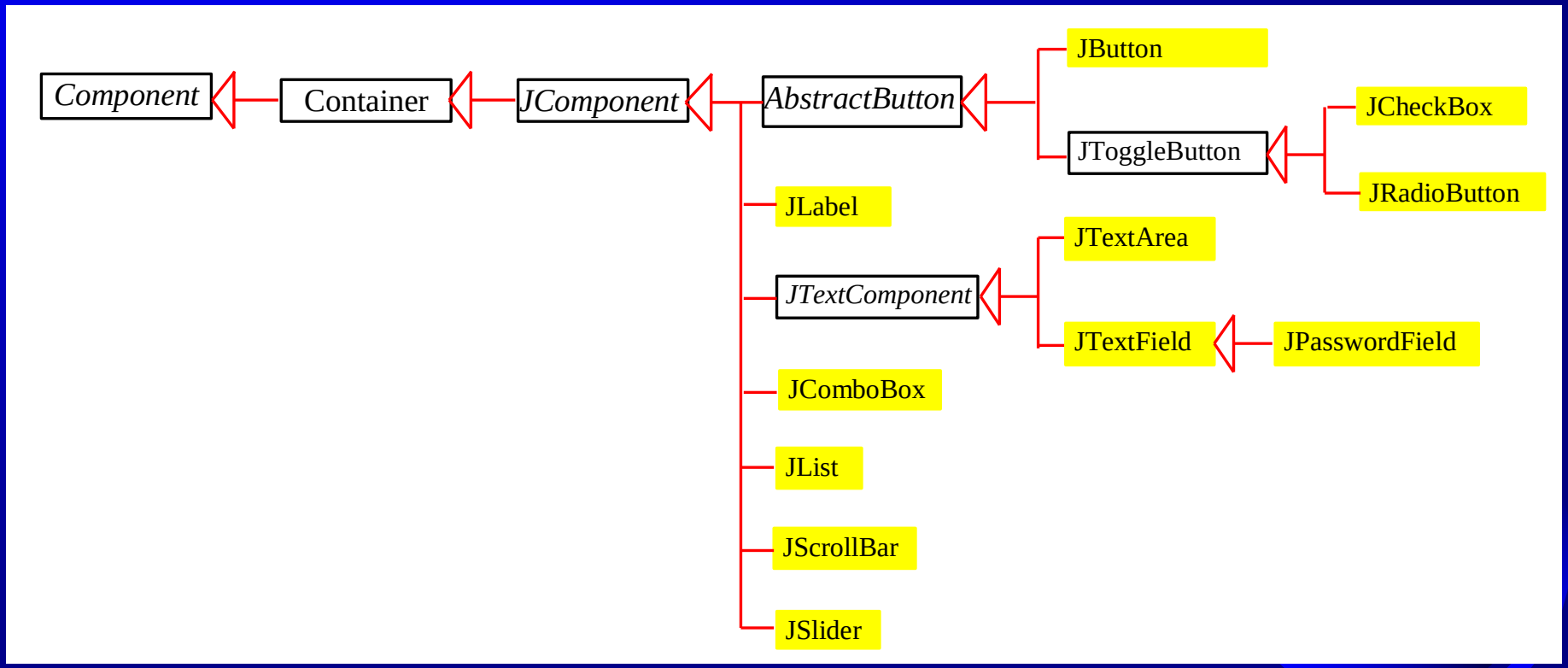
- GUI components in Java are very flexible and versatile, you can create a wide assortment of useful user interfaces.

Graphical User Interface

- Graphical user interfaces can be created:
  - using various user-interface components: JButton, JCheckBox, JRadioButton, JLabel, JTextField, JTextArea, JComboBox, JList, JScrollBar, and JSlider
  - using listeners for various types of events
  - using display of multiple windows in an application

GUI Components

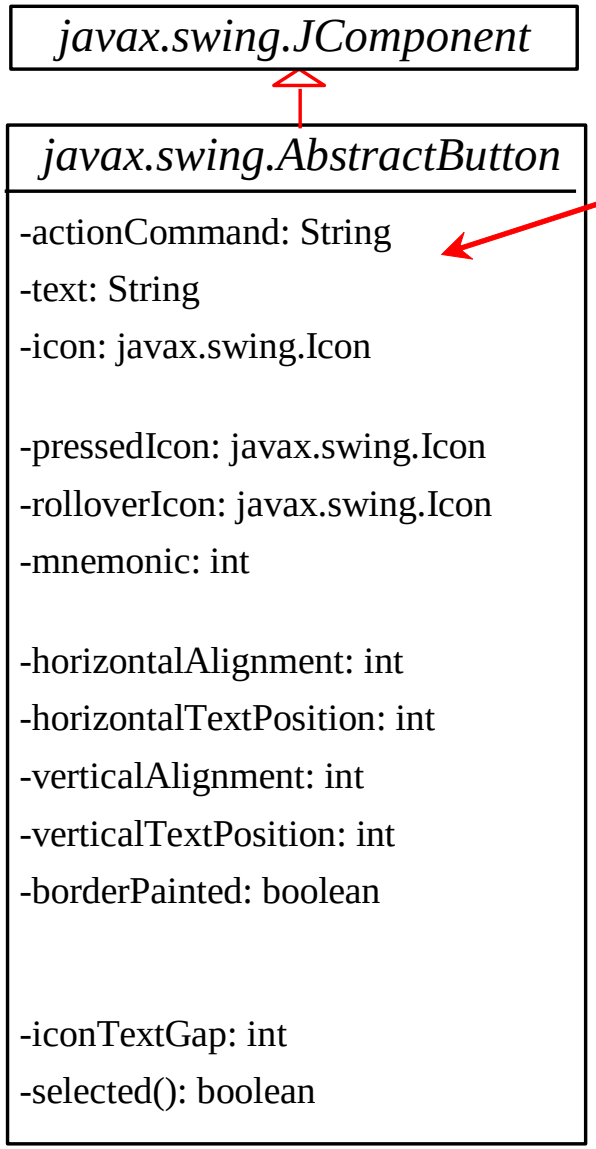
- Introduces the frequently used GUI components
- Uses borders and icons



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

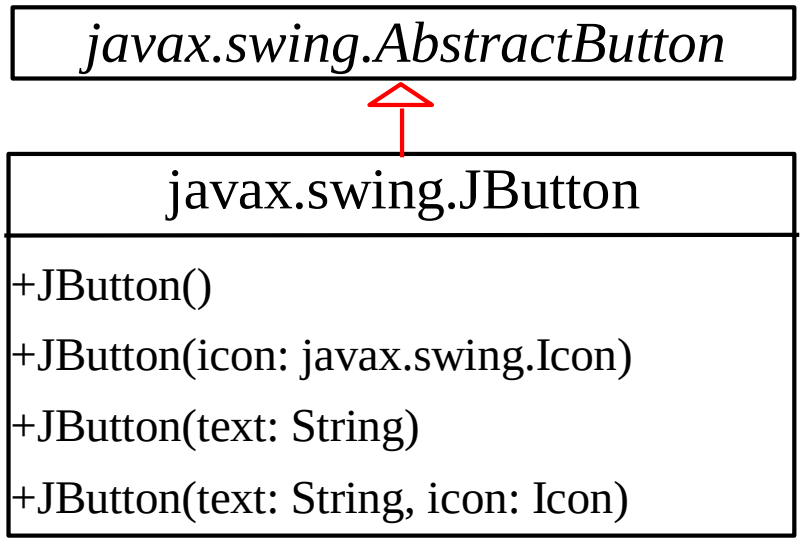


The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

- 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 explicitly set pressed icon.
- 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 is not painted.
- The gap between the text and the icon on the button (JDK 1.4).
- 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.



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.

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