



GUI Programming II

Event-Driven Programming

Procedural and Event-Driven Programming

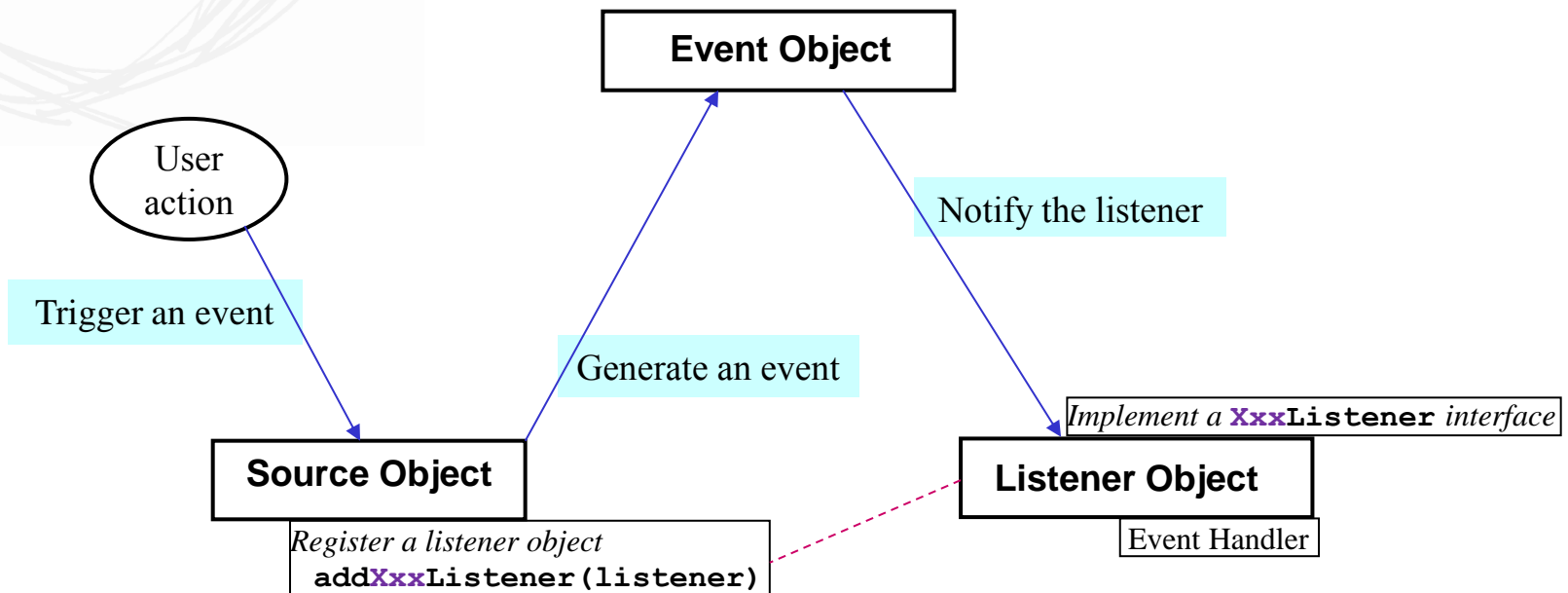
- ***Text-based applications***: object-oriented but executed in a procedural order
 - The program dictates the flow of execution and code is executed in procedural order
- ***GUI applications***: object-oriented and event-driven
 - code is executed upon activation of events: *a button click; a mouse movement* etc.

Events

- An **event** can be defined as a type of signal to the program that **something has happened**
- Any operating environment constantly monitors events and reports them to running programs. Each program decides what, if any, to do in response to these events.
- The event is generated by
 - External user actions
 - mouse movements, mouse button clicks, and keystrokes etc.
 - Internal program activities (or the operating system)
 - Timer
- Event Source
 - Source Object
 - the component on which an event is fired or generated
 - Event objects contain properties pertinent to the event

Java Event Delegation Model

An user action on a source object triggers an event, and an object interested (registered) in the event receives the event.



Event Handling

- 3 Parts - Model
 - Event Source
 - GUI components users interact with
 - Event Object
 - Encapsulates event information
 - Event Listener
 - Receives the event object when notified, then responds
- 2 Programming Tasks
 - Register **event listener** for event source
 - Implement event-handling methods (**handlers**)

Fundamentally important to GUI programming

Example 1: Button Frame

```
public class ButtonFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My Button Frame");
        ButtonPanel panel = new ButtonPanel();
        frame.add(panel);
        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

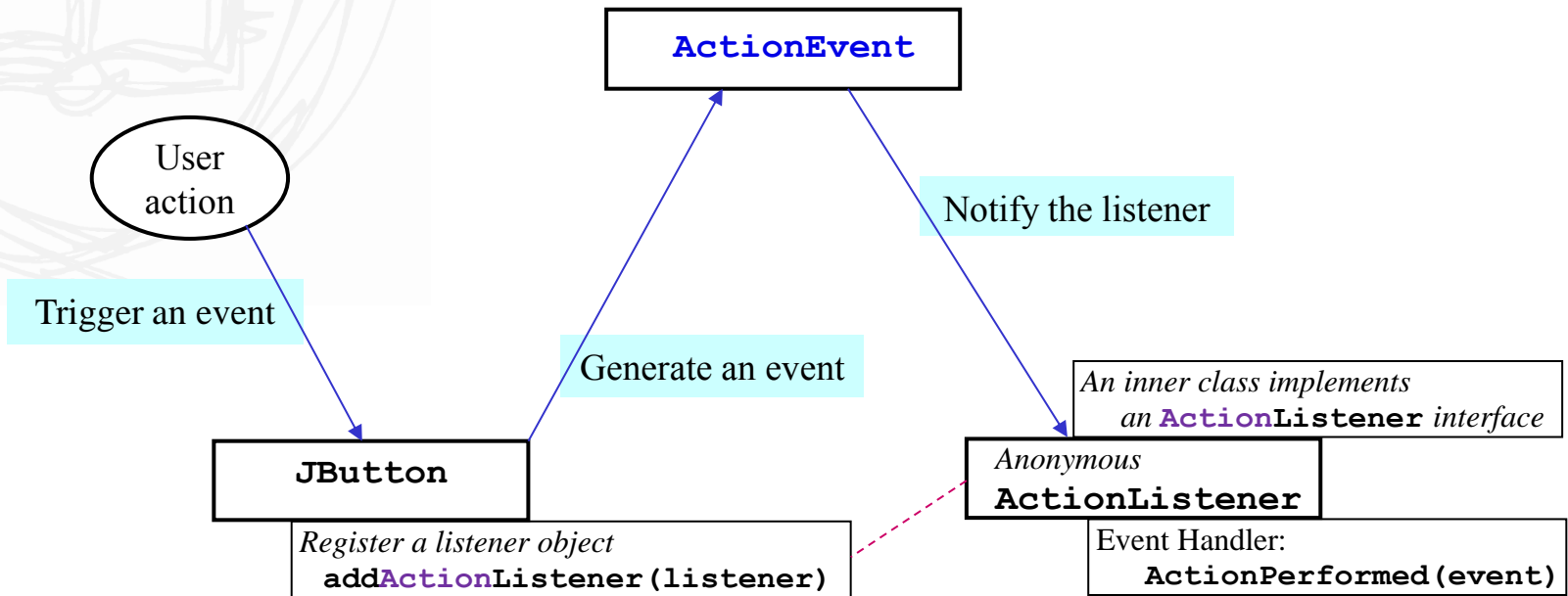
class ButtonPanel extends JPanel {
    ButtonPanel() {
        JButton redButton = new JButton("Red");
        add(redButton);
        redButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                setBackground(Color.RED);
            }
        });

        JButton blueButton = new JButton("Blue");
        add(blueButton);
        blueButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                setBackground(Color.BLUE);
            }
        });
    }
}
```

//add red button
//add button to panel
//register event listener

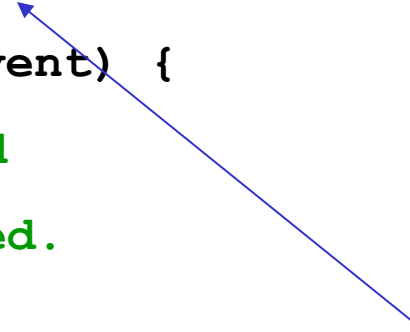
//add blue button
//add button to panel
//register event listener

Example 1: Button Frame



Learn a Java Idiom – Add Listener and Handler

```
SourceObject.addEventListener(new EventListener() {  
    public void EventHandler(Event event) {  
        /* Code here will be executed  
        * when the event is triggered.  
        */  
    }  
});
```



Anonymous Inner Class

Example



```
blueButton.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent event) {  
        setBackground(Color.BLUE);  
    }  
});
```


Example 2: Create Listener Class

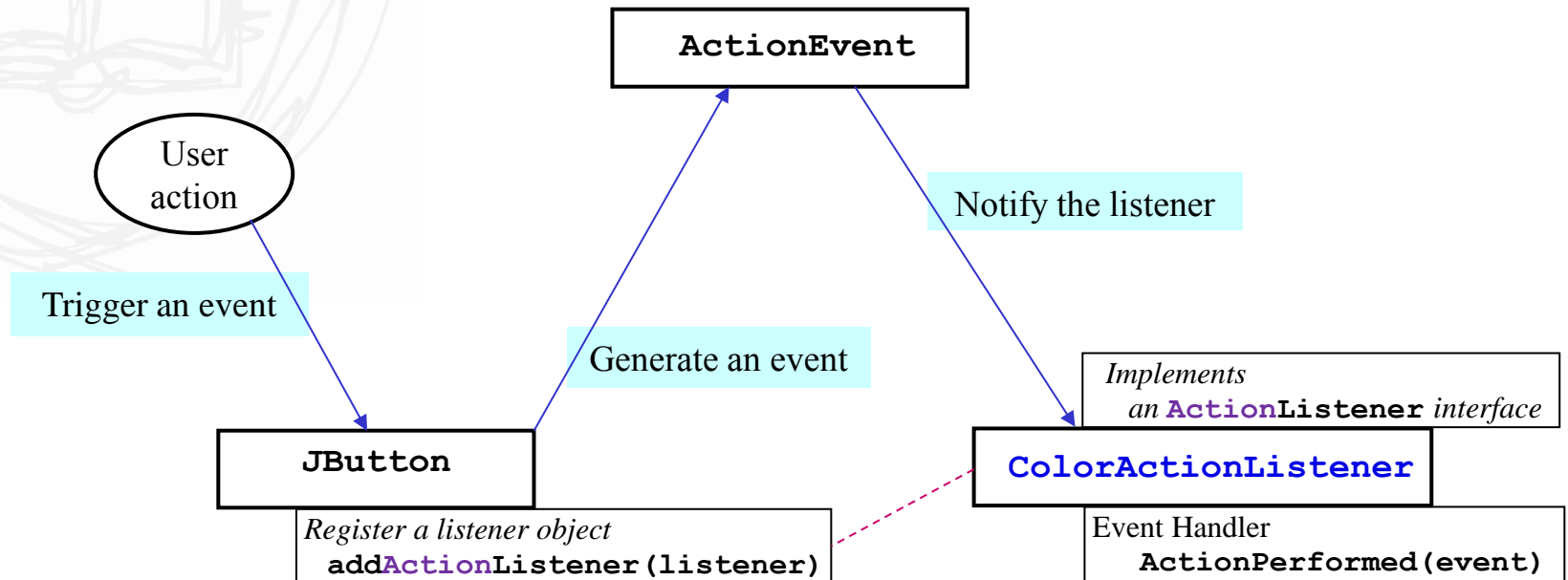
```
public class ListeningClass {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My Listening Class");
        MyButtonPanel panel = new MyButtonPanel();
        ...
    }
}

class MyButtonPanel extends JPanel {
    JButton yellowButton = new JButton("Yellow");
    JButton greenButton = new JButton("Green");

    MyButtonPanel() {
        ColorActionListener acitonListeningObject = new ColorActionListener();
        add(yellowButton); //add Yellow button to panel
        yellowButton.addActionListener(acitonListeningObject); //register event listener
        add(greenButton); //add Green button to panel
        greenButton.addActionListener(acitonListeningObject); //register event listener
    }
}

class ColorActionListener implements ActionListener{
    public void actionPerformed(ActionEvent event) { //implementing the(ActionEvent) handler
        if (event.getSource() == yellowButton) {
            setBackground(Color.YELLOW);
        } else if (event.getSource() == greenButton) {
            setBackground(Color.GREEN);
        }
    }
}
```

Example 2: Create Listener Class



Example 3: Turning Components into Event Listeners

```
public class TestListeningPanel {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My Listening Panel");
        ListeningPanel panel = new ListeningPanel();
        ...
    }
}

class ListeningPanel extends JPanel implements ActionListener {
    JButton orangeButton = new JButton("Orange");
    JButton pinkButton = new JButton("Pink");

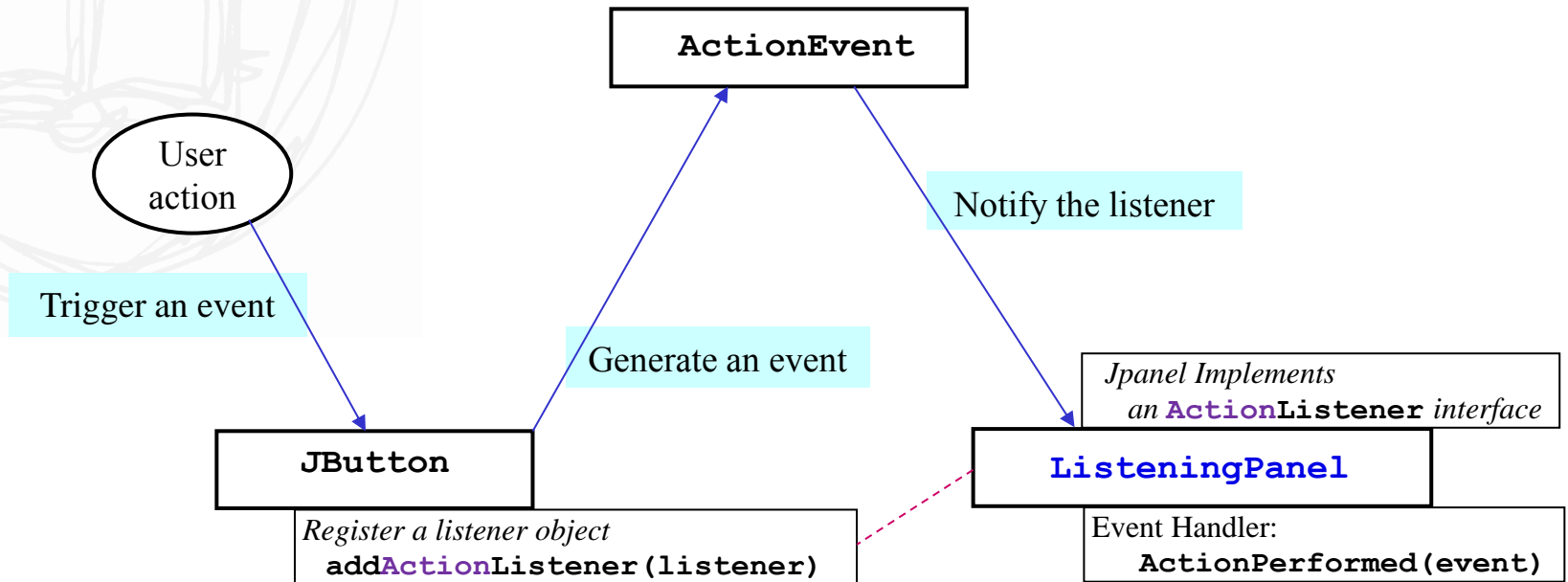
    ListeningPanel() {
        add(orangeButton);
        orangeButton.addActionListener(this);
        add(pinkButton);
        pinkButton.addActionListener(this);
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == orangeButton) {
            setBackground(Color.ORANGE);
        } else if (event.getSource() == pinkButton) {
            setBackground(Color.PINK);
        }
    }
}

//add Orange button to panel
//register event listener
//add Pink button to panel
//register event listener

//implementing the(ActionEvent) handler
```

Example 3: Turning Components into Event Listeners



Completely free to designate **any object** of a class that implements the **EventListener** interface as an **event listener**

How Event Handling Works

- How do event handlers get *registered*
 - Through component's method `addEventListener()`
- How does a component know to call *EventHandler()*
 - Event is dispatched only to listeners of the appropriate type
 - Each event type has corresponding event-listener interface

Event Handling Programming

1. Implement a listener interface
2. Register the listener with an event source
3. Wait for the event source to call your event-handler method

User Action, Source Object and Event Type

User Action	Source Object	Event Type Generated
Click a button	JButton	ActionEvent
Press return on a text field	TextField	ActionEvent
Select a new item	JComboBox	ItemEvent, ActionEvent
Select item(s)	JList	ListSelectionEvent
Click a check box	JCheckBox	ItemEvent, ActionEvent
Click a radio button	JRadioButton	ItemEvent, ActionEvent
Select a menu item	JMenuItem	ActionEvent
Move the scroll bar	JScrollBar	AdjustmentEvent
Window opened, closed, iconified, deiconified or closing	Window	WindowEvent
Component added or removed from the container	Container	ContainerEvent
Component moved, resized, hidden or shown	Component	ComponentEvent
Component gained or lost focus	Component	FocusEvent
Key released or pressed	Component	KeyEvent
Mouse pressed, released, clicked, entered or exited	Component	MouseEvent
Mouse moved or dragged	Component	MouseEvent

Events, Event Listeners and Listener Methods

Event Class	Listener Interface	Listener Methods (Handler)
ActionEvent	ActionListener	actionPerformed(ActionEvent e)
ItemEvent	ItemListener	itemStateChanged(ItemEvent e)
WindowEvent	WindowListener (WindowAdapter)	windowClosing(WindowEvent e) windowOpened(WindowEvent e) windowIconified(WindowEvent e) windowDeiconified(WindowEvent e) windowClosed(WindowEvent e) windowActivated(WindowEvent e) windowDeactivated(WindowEvent e)
ContainerEvent	ContainerListener (ContainerAdapter)	componentAdded(ContainerEvent e) componentRemoved(ContainerEvent e)
ComponentEvent	ComponentListener (ComponentAdapter)	componentMoved(ComponentEvent e) componentHidden(ComponentEvent e) componentResized(ComponentEvent e) componentShown(ComponentEvent e)
FocusEvent	FocusListener (FocusAdapter)	foucsGained(FocusEvent e) foucsLost(FocusEvent e)
TextEvent	TextListener	textValueChanged(TextEvent e)
KeyEvent	KeyListener (KeyAdapter)	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)
MouseEvent	MouseListener (MouseAdapter) MouseMotionListener (MouseMotionAdapter)	mousePressed(MouseEvent e) mouseReleased(MouseEvent e) mouseEntered(MouseEvent e) mouseExited(MouseEvent e) mouseClicked(MouseEvent e) mouseDragged(MouseEvent e) mouseMoved(MouseEvent e)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent e)

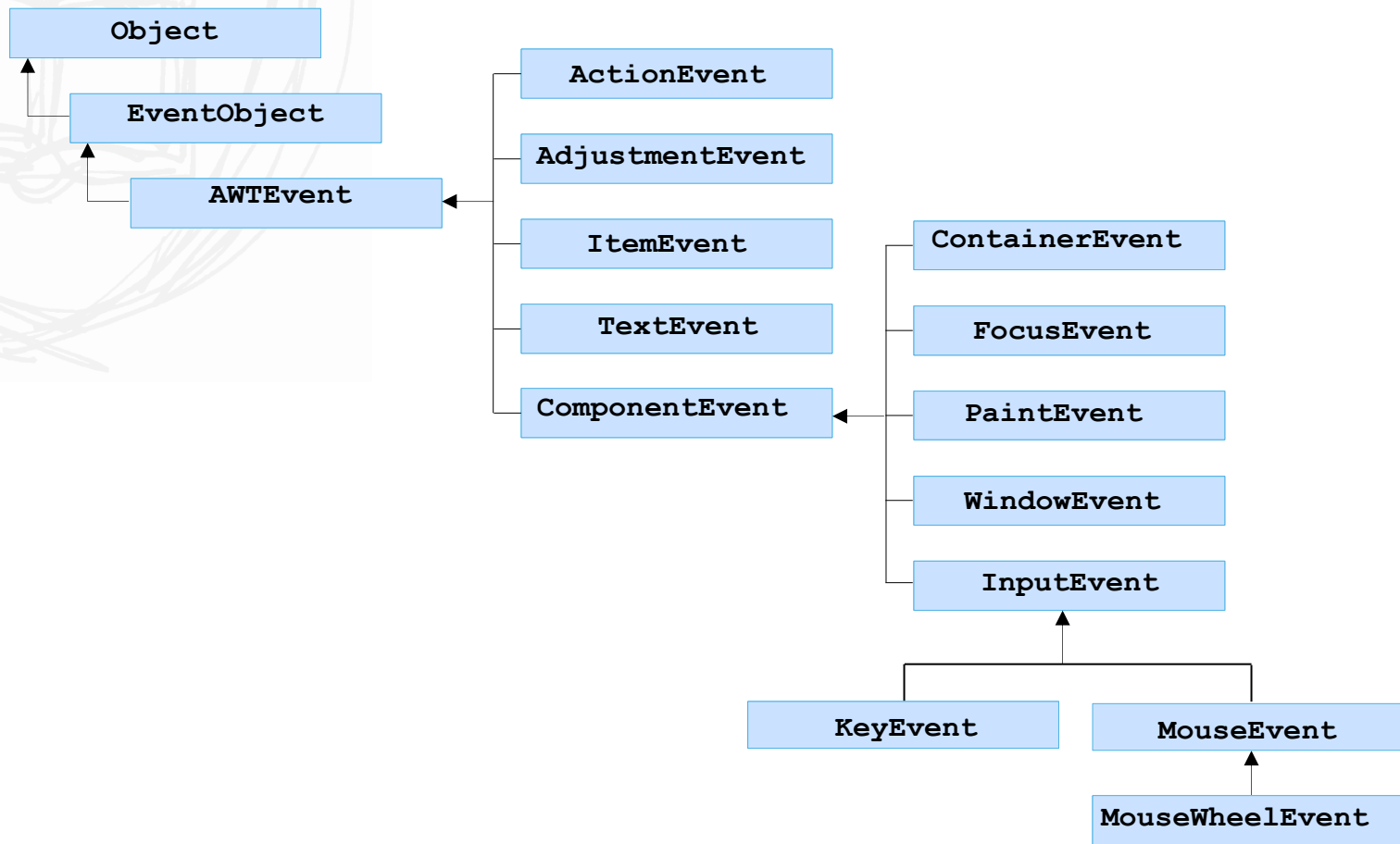
Semantic and Low-level Events

- Semantic Events:
what the user is doing – combination of low-level events
 - AcitonEvent
 - AdjustmentEvent
 - ItemEvent
 - TextEvent
- Low-level Events
window-system occurrences or low-level input
 - ComponentEvent
 - KeyEvent
 - MouseEvent
 - MouseWheelEvent (from SDK 1.4)
 - FocusEvent
 - WindowEvent
 - ContainerEvent

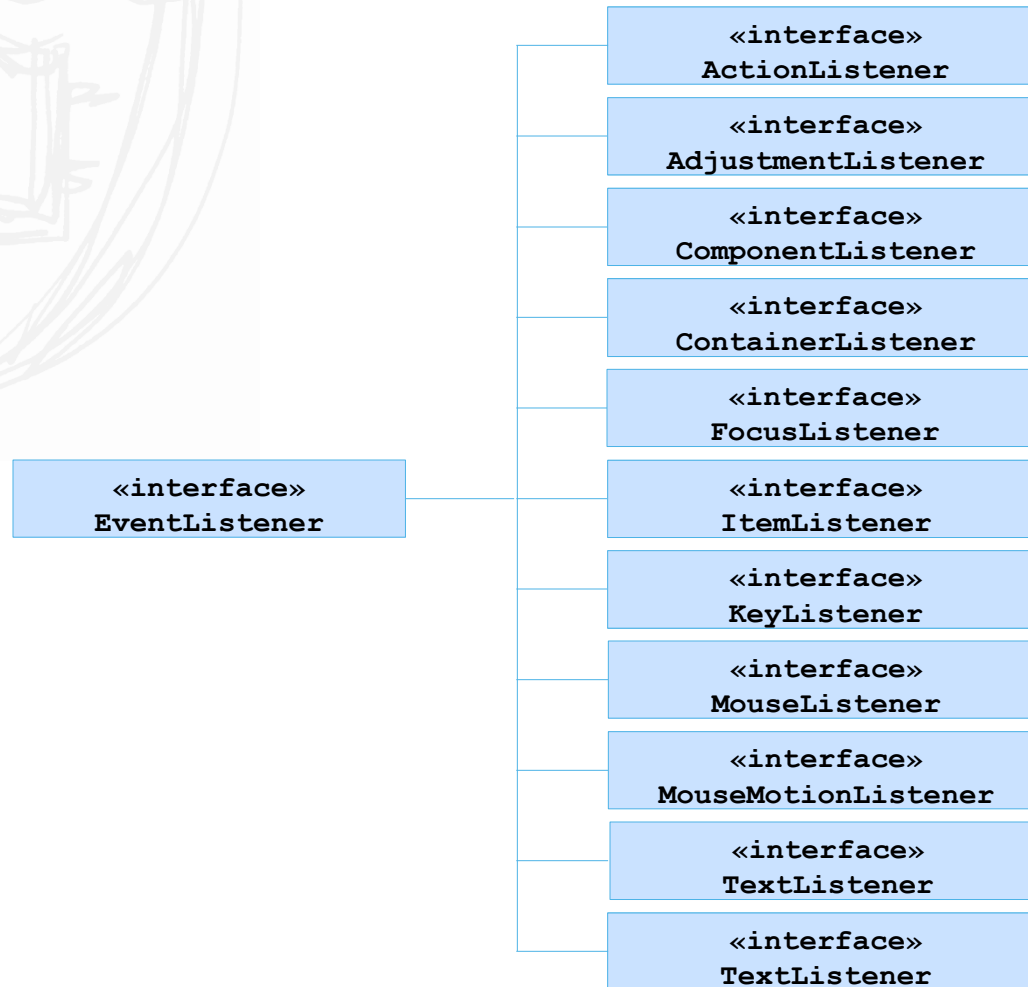
Event Objects

id	A number that identifies the event.
target	The source component upon which the event occurred.
arg	Additional information about the source components.
x, y coordinates	The mouse pointer location when a mouse movement event occurred
clickCount	The number of consecutive clicks for the mouse events. For other events, it is zero.
when	The time stamp of the event
key	The key that was pressed or released.

Some Event Classes



Event-listener Interfaces



Example: Do-nothing Methods

Monitor when the user tries to close the main frame because you don't want your users to lose unsaved work

```
WindowListener listener = . . . ;  
frame.addWindowListener(listener);
```

```
class Terminator implements WindowListener
```

```
{  
    public void windowClosing(WindowEvent e)  
    {  
        if (user agrees)  
            System.exit(0);  
    }  
    public void windowOpened(WindowEvent e) {}  
    public void windowClosed(WindowEvent e) {}  
    public void windowIconified(WindowEvent e) {}  
    public void windowDeiconified(WindowEvent e) {}  
    public void windowActivated(WindowEvent e) {}  
    public void windowDeactivated(WindowEvent e) {}  
}
```

```
public interface WindowListener  
{  
    void windowOpened(WindowEvent e);  
    void windowClosing(WindowEvent e);  
    void windowClosed(WindowEvent e);  
    void windowIconified(WindowEvent e);  
    void windowDeiconified(WindowEvent e);  
    void windowActivated(WindowEvent e);  
    void windowDeactivated(WindowEvent e);  
}
```

What a tedious busywork for nothing!

Event Adapter Classes

- Listener interfaces having more than one method come with a companion adapter class
- Implement the **Event** interface
- Provide default implementation (*do-nothing*) of each interface method
- Used when all methods in interface is not needed
- The listener classes that you define can extend adapter classes and override only the methods that you need

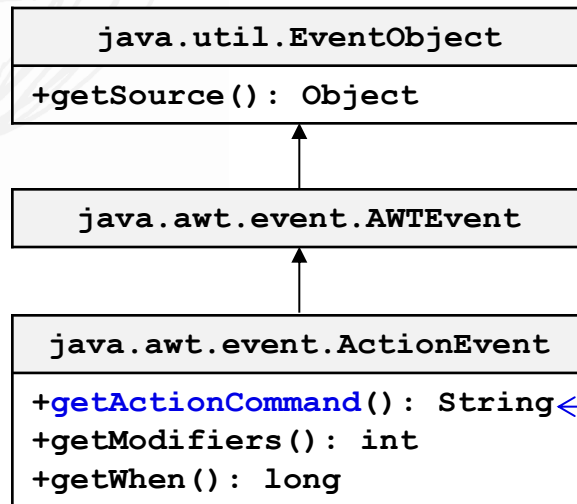
Convenient Adapters

Adapter	Interface
WindowAdapter	WindowListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
KeyAdapter	KeyListener
ContainerAdapter	ContainerListener
ComponentAdapter	ComponentListener
FocusAdapter	FocusListener

Previous example continues

```
class Terminator extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        //override only what you need
        if (user agrees)
            System.exit(0);
    }
}
```

The Action Event



*Returns the command string.
For a button, its text is the command string*

Mouse Events

`java.awt.event.InputEvent`

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

`java.awt.event.MouseEvent`

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

`<<interface>>`

`java.awt.event.MouseListener`

`+mousePressed (e: MouseEvent): void`
`+mouseReleased(e: MouseEvent): void`
`+mouseClicked (e: MouseEvent): void`
`+mouseEntered (e: MouseEvent): void`
`+mouseExited (e: MouseEvent): void`

`<<interface>>`

`java.awt.event.MouseMotionListener`

`+mouseDragged (e: MouseEvent): void`
`+mouseMoved (e: MouseEvent): void`

Example: Mouse Event

```
public class MouseFrame{
    public static void main(String[] args) {
        new Movinglabel("My Mouse Frame");
    }
}

class Movinglabel extends JFrame{

    JLabel label = new JLabel("You clicked @ here :-");

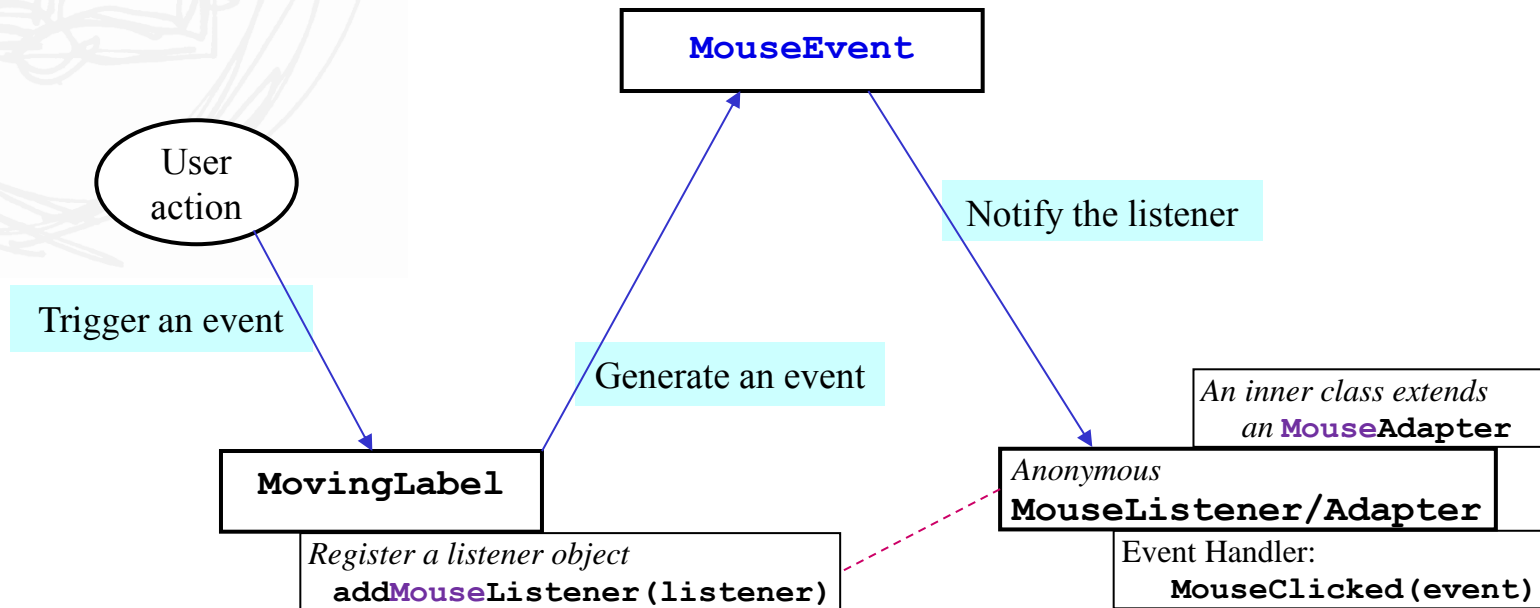
    Movinglabel(String message) {
        super(message);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(500, 400);
        setLocationRelativeTo(null);
        add(label);
        setVisible(true);

        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent event) {
                int x = event.getX();
                int y = event.getY();
                label.setBounds(x-8,y-30,150,10);
            }
        });
    }
}
```

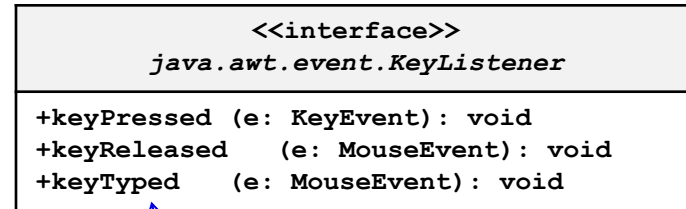
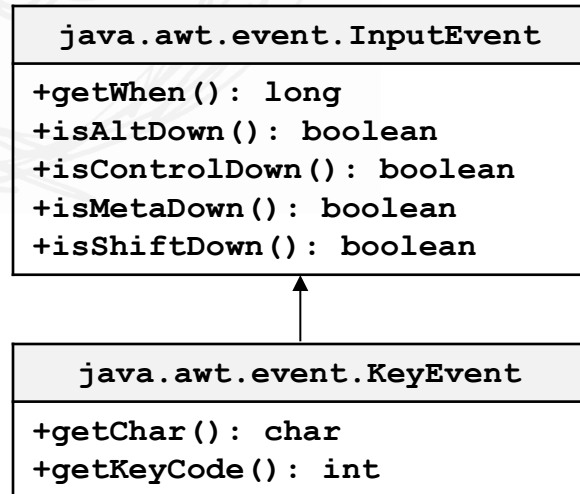
// add MouseListener
// handle mouse clicked

// relocating label

Example: Mouse Event



Key Events



keyTyped = keyPressed + keyReleased

Example: Key Event

```
public class KeyFrame{
    public static void main(String[] args) {
        new KeyLabel();
    }
}

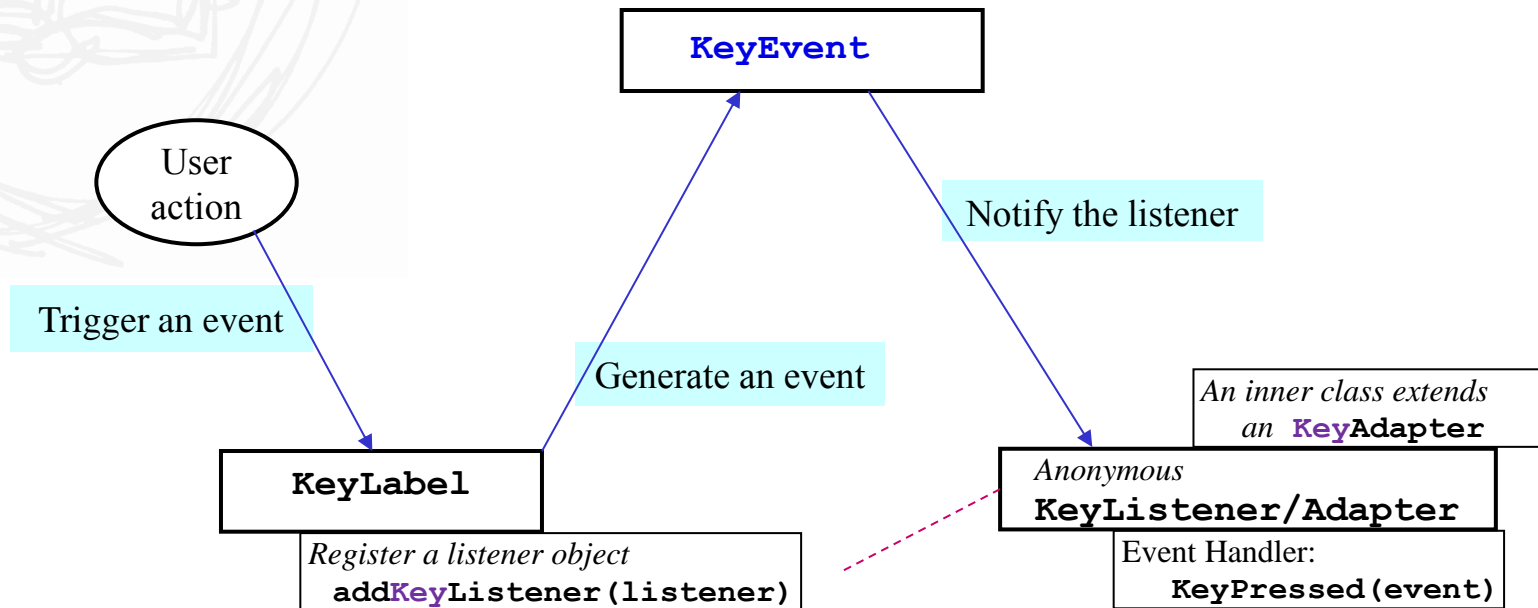
class KeyLabel extends JFrame{

    JLabel label = new JLabel("", JLabel.CENTER);

    KeyLabel() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(500, 400);
        setLocationRelativeTo(null);
        label.setForeground(Color.BLUE);
        label.setFont(new Font("Serif", Font.BOLD, 40));
        add(label);
        setVisible(true);

        addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent event) {
                label.setText(String.valueOf(label.getText() + event.getKeyChar()));
            }
        });
    }
}
```

Example: Key Event



Example: Timer Class

A Non-GUI Event Source

```
public class MovingFrame{  
    public static void main(String[] args) {  
        MovingLabel movingLabel = new MovingLabel("A Moving Label");  
        . . .  
    }  
}
```

```
class MovingLabel extends JFrame{  
    private int x = 0;  
    JLabel label = new JLabel("Hello again, again!");  
    MovingLabel(String message) {  
        label.setForeground(Color.BLUE);  
        label.setFont(new Font("Serif", Font.BOLD, 20));  
        add(label);  
    }  
}
```

A **Timer** object serves as the source of an **ActionEvent**

```
    Timer timer = new Timer(10, new TimerListener());  
    timer.start();  
}
```

```
class TimerListener implements ActionListener {  
    public void actionPerformed(ActionEvent event) {  
        if(x > getWidth()){  
            x = -150;  
        }  
        x += 1;  
        label.setBounds(x,166,160,30);  
    }  
}
```

// create a ActionListener
// handling ActionEvent

// relocating label

Notes on Event Handling

- The listener object's class must implement the corresponding event-listener interface and must be registered by the source object
- One source object may fire several types of events
- One event handler can respond to multiple event sources
 - the same listener is added to multiple event sources
 - **Action** interface that extends **ActionListener** interface; abstract class **AbstractAction** that implements **Action** interface; Your class extends the **AbstractAction** then add the **actionPerformed** method
- Multiple listeners can react to the same event
 - all register to the same event and their handlers are called when the event occurs