11 GUI Event Handling

Objectives

- Write code to handle events that occur in a GUI
- Describe the concept of adapter classes, including how and when to use them
- Determine the user action that originated the event from the event object details
- Create the appropriate interface and event handler methods for a variety of event types

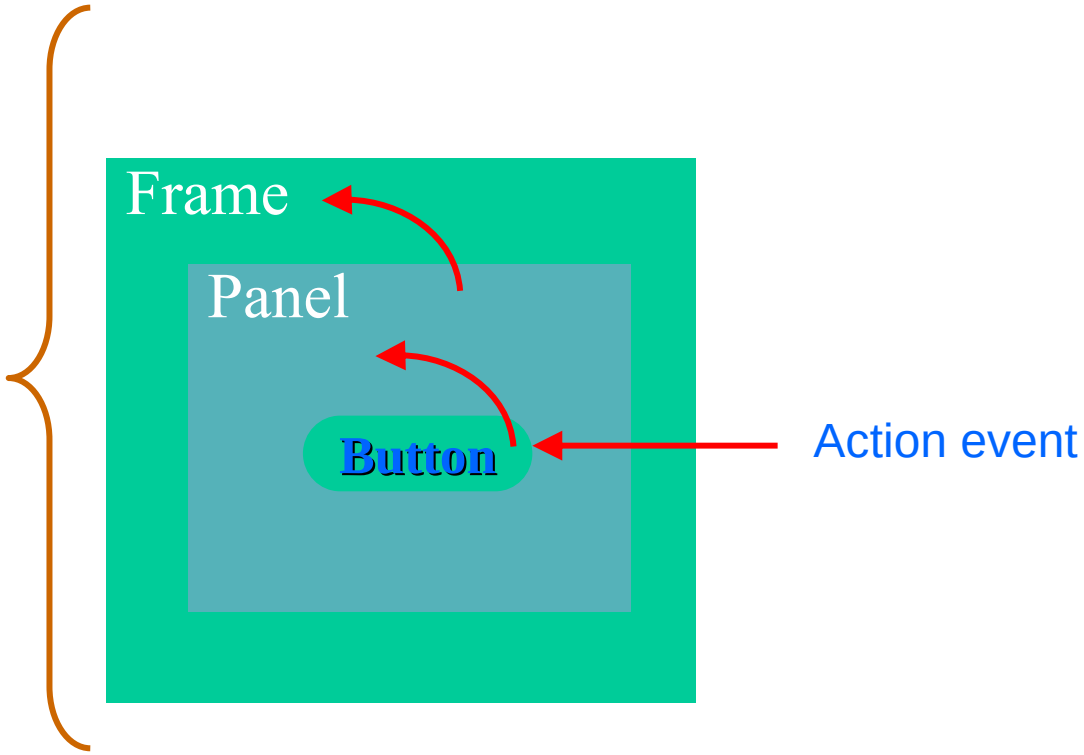
What is an Event?

- Events - Objects that describe what happened
- Event sources - The generator of an event
- Event handlers - A method that receives an event object, deciphers it, and processes the user's interaction

Hierarchical Model (JDK1.0)

- Is based on containment

action()
lostFocus()
mouseExit()
gotFocus()
mouseDown()
mouseMove()
keyDown()
mouseDrag()
mouseUp()
keyUp()
mouseEnter()



- Advantages

  - It is simple and well suited to an object-oriented programming environment.

- Disadvantages

  - An event can only be handled by the component from which it originated or by one of the containers of the originating component.

  - In order to handle events, you must either subclass the component that receives the event or create a handleEvent() method at the base container.

Delegation Model (JDK1.1)

► Events are sent to the component from which the event originated, but it is up to each component to propagate the event to one or more registered classes called listener. Listeners contain event handlers that receive and process the event. In this way, the event handler can be in an object separate from the component. Listeners are classes that implement the EventListener interface.

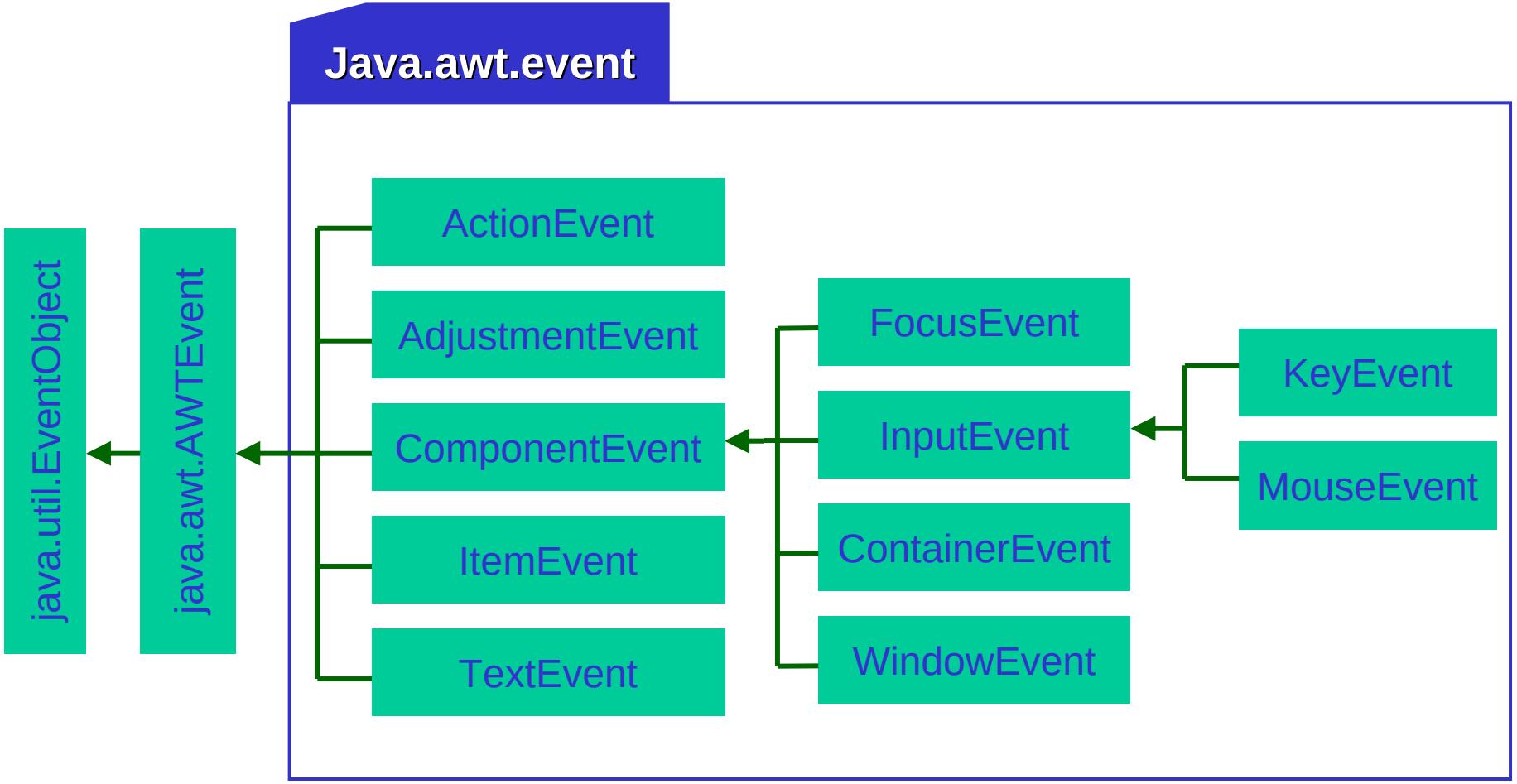
► Events are objects that are reported only to registered listeners. Every event has corresponding listener interface that mandates which methods must be defined in a class suited to receiving that type of event. The class that implements the interface defines those methods, and can be registered as a listener.

► Events from components that have no registered listeners are not propagated.

Delegation Model

- Client objects (handlers) register with a GUI component they want to observe.
- GUI components only trigger the handlers for the type of event that has occurred
  - Most components can trigger more than one type of event
- Distributes the work among multiple classes

Event Categories



Event Class	Listener Interface	Listener Methods
ActionEvent	ActionListener	actionPerformed()
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged()
ComponentEvent	ComponentListener	componentHidden()
		componentMoved()
		componentResized()
		componentShown()
ContainerEvent	ContainerListener	componentAdded()
		componentRemoved()
FocusEvent	FocusListener	focusGained()
		focusLost()
ItemEvent	ItemListener	itemStateChanged()
KeyEvent	KeyListener	keyPressed()
		keyReleased()
		keyTyped()

MouseEvent	MouseListener	mouseClicked()
		mouseEntered()
		mouseExited()
		mousePressed()
		mouseReleased()
MouseMotionEvent	MouseMotionListener	mouseDragged()
		mouseMoved()
TextEvent	TextListener	textValueChanged()
WindowEvent	WindowListener	windowActivated()
		windowClosed()
		windowClosing()
		windowDeactivated()
		windowDeiconified()
		windowIconified()
		windowOpened()

Component	Events Generated	Meaning
Button	ActionEvent	User clicked on the button
Checkbox	ItemEvent	User selected or deselected an item
CheckboxMenuItem	ItemEvent	User selected or deselected an item
Choice	ItemEvent	User selected or deselected an item
Component	ComponentEvent	Component moved, resized, hidden, or shown
	FocusEvent	Component gained or lost focus
	KeyEvent	User pressed or released a key
	MouseEvent	User pressed or released mouse button, mouse entered or exited component, or user moved or dragged mouse. Note: MouseEvent has two corresponding listeners, MouseListener and MouseMotion Listener.

Container	ContainerEvent	Component added to or removed from container
List	ActionEvent	User double-clicked on list item
	ItemEvent	User selected or deselected an item
MenuItem	ActionEvent	User selected a menu item
Scrollbar	AdjustmentEvent	User moved the scrollbar
TextComponent	TextEvent	User changed text
TextField	ActionEvent	User finished editing text
Window	WindowEvent	Window opened, closed, iconified, deiconified, or close requested

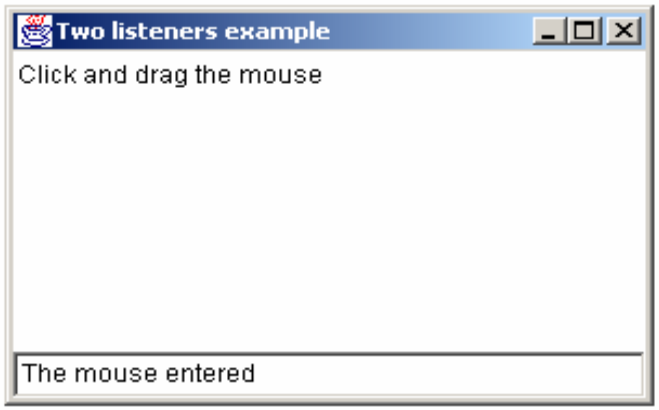
Example

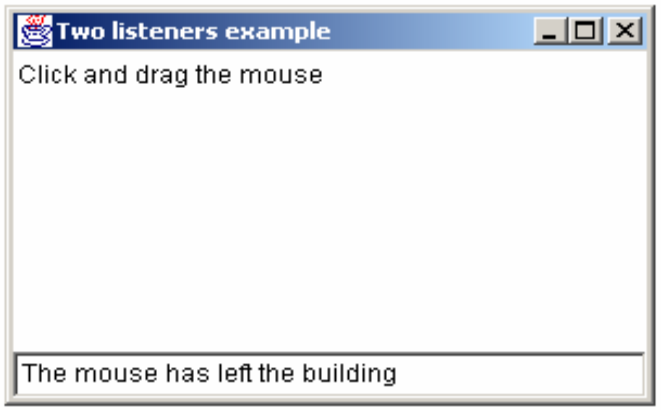
```
import java.awt.* ;

import java.awt.event.* ;

public class TwoListener implements MouseMotionListener,
MouseListener

{

      private Frame f   ;

      private TextField tf;


       public TwoListener() {

            f = new Frame("Two listeners example") ;

            tf = new TextField(30) ;

       }
```

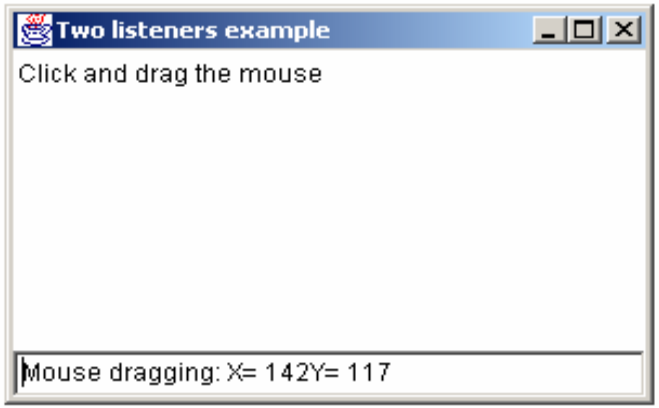
```
public void launchFrame()  {
        Label label = new Label("Click and drag the mouse") ;
        f.add(label, BorderLayout.NORTH) ;
        f.add(tf, BorderLayout.SOUTH) ;
        f.addMouseMotionListener(this) ;
        f.addMouseListener(this) ;
        f.setSize(300,200) ;
        f.setVisible(true) ;
    }
 // These are MouseMotionListener events
 public void mouseDragged( MouseEvent e)   {
        String s = "Mouse dragging: X= " + e.getX() + "Y= " + e.getY() ;
        tf.setText(s) ;
  }
```

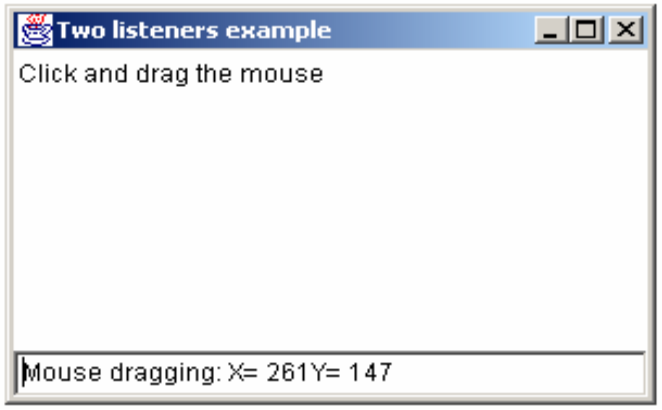
```
public void mouseEntered( MouseEvent e)  {
        String s = "The mouse entered" ;
        tf.setText(s) ;
}
public void mouseExited( MouseEvent e)  {
        String s = "The mouse has left the building" ;
        tf.setText(s) ;
}
// Unused MouseMotionListener method
// All methods of a listener must be present in the
// class even if they are not used
public void mouseMoved(MouseEvent e) { }
// Unused MouseListener methods
public void mousePressed(MouseEvent e) { }
public void mouseClicked(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
```

```
    public static void main(String[] args)          {
            TwoListener two = new TwoListener() ;
          two.launchFrame();
    }
}
```


Two listeners example
Click and drag the mouse
The mouse entered


Two listeners example
Click and drag the mouse
The mouse has left the building


Two listeners example
Click and drag the mouse
Mouse dragging: X= 142Y= 117


Two listeners example
Click and drag the mouse
Mouse dragging: X= 261Y= 147

Multiple Listeners

- Multiple listeners cause unrelated parts of a program to react to the same event.
- The handlers of all registered listeners are called when the event occurs

Event Adapters

- The listener classes that you define can extend adapter classes and override only the methods that you need.

- Example:

```
import java.awt.* ;
import java.awt.event.* ;
public class MouseClickHandler extends MouseAdapter {
        public void mouseClicked( MouseEvent e) {
           // do stuff with the mouse click...
        }
}
```

Event Handling Using Anonymous Classes

- You can include an entire class definition within the scope of an expression.
- This approach defines what is called an anonymous inner class and creates the instance all at once.
- For example:

↳next slide

```
import java.awt.* ;
import java.awt.event.* ;

public class TestAnonymous {
      private Frame f   ;
      private TextField tf;

      public TestAnonymous() {
                f = new Frame("Anonymous class example") ;
               tf = new TextField(30) ;
      }
      public void launchFrame() {
            Label label = new Label("Click and drag the mouse") ;
            f.add(label, BorderLayout.NORTH) ;
            f.add(tf, BorderLayout.SOUTH) ;
            f.addMouseMotionListener(new MouseMotionAdapter()
            {
               public void mouseDragged( MouseEvent e){
                     String s = "Mouse dragging: X= " + e.getX() +
                                    "Y= " + e.getY() ;
                     tf.setText(s) ;
}
            }) ;
```

```
        f.addMouseListener( new MouseClickHandler(tf) ) ;
        f.setSize(300,200) ;
        f.setVisible(true) ;
    }

    public static void main(String[] args)
    {
        TestAnonymous obj = new TestAnonymous() ;
        obj.launchFrame();
    }
}
```

```
import java.awt.* ;
import java.awt.event.* ;
public class MouseClickHandler extends MouseAdapter {
        private TextField tf ;
        public static int count = 0 ;
        public MouseClickHandler(TextField tf)  {
                this.tf = tf ;
        }
        public void mouseClicked( MouseEvent e) {
            count++;
            String s = "Mouse has been clicked " + count + " times so far." ;
tf.setText(s) ;
    }
}
```

Event Handling Using Inner Classes

- You can implement event handlers as inner class.
- This allows access to the private data of the outer class.
- For example:

➪next slide

```
import java.awt.* ;
import java.awt.event.* ;

public class TestInner {
      private Frame f   ;
      private TextField tf;

      public TestInner() {
            f = new Frame("Inner classes example") ;
            tf = new TextField(30) ;
      }
      public void launchFrame() {
            Label label = new Label("Click and drag the mouse") ;
            f.add(label, BorderLayout.NORTH) ;
            f.add(tf, BorderLayout.SOUTH) ;
            f.addMouseMotionListener(new MyMouseMotionListener()) ;
            f.addMouseListener(new MouseClickHandler(tf)) ;
            f.setSize(300,200) ;
            f.setVisible(true) ;
       }
```

```
class MyMouseMotionListener extends MouseMotionAdapter  {
        public void mouseDragged( MouseEvent e ) {
                String s = "Mouse dragging: X= " + e.getX() +
                                "Y= " + e.getY() ;
                tf.setText(s) ;
         }
         public static void main(String[] args)   {
             TestInner obj = new TestInner() ;
             obj.launchFrame();
        }
}
```