Web Technology 15
Event Handling

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Delegation Event Model

- A *source* generates an event and sends it to one or more *listeners*
- The listener simply waits until it receives an event
- Once an event is received, the listener processes the event and then returns
- The event generating component delegates the responsibility of performing an event-based action to a separate event-performing component

Events

- It is an object that describes a state change in a source
- Event could be the occurrence of any activity such as mouse click or key press
- An event may be generated when a timer expires, s/w or h/w failure occurs or an operation is completed

Delegation Event Model...

Event Sources

- A *source* is an object that generates an event. It occurs when the internal state of that object changes. Sources may generate more than one type of event
- A source must register listeners in order for the listeners to receive notifications about a specific type of event
  public void add*Type*Listener (*Type*Listener el)
  where, *Type* is the event name and *el* is the reference to the event listener
  Ex: *addKeyListener()*
  addMouseMotionListener()
- Some sources may allow only one listener to register
  public void add*Type*Listener(*Type*Listener el) throws java.util.TooManyListenersException
- A source must provide mechanism to unregister listener
  public void remove*Type*Listener(*Type*Listener el)
  Ex: *removeKeyListener()*

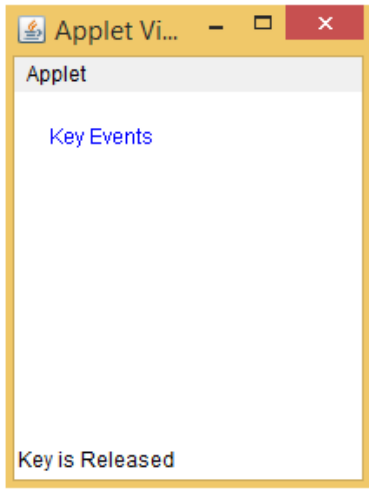
Delegation Event Model...

Event Listeners

- A *Listener* is an object which is notified when an event occurs
  - It must have been registered with one/more sources to receive notifications
  - It must implement methods to receive and process these notifications
- The methods that receive and process events are implemented from the corresponding listener interfaces in java.awt.event package

Delegation Event Model...

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*
<APPLET CODE="Ev" HEIGHT=200 WIDTH=200>
</APPLET>
*/
public class Ev extends Applet implements KeyListener{
        public void init(){
        addKeyListener(this);
        }
        public void keyReleased(KeyEvent e){
        showStatus("Key is Released");
        }
        public void keyTyped(KeyEvent e){
        showStatus("Key is Typed");
        }
        public void keyPressed(KeyEvent e){
        showStatus("Key is Pressed");
        }
        public void paint(Graphics g){
        g.setColor(Color.blue);
        g.drawString("Key Events", 20, 30);
        }
}
```



Event Classes

Event Classes

- All the events have corresponding event classes associated with them. Each class is derived from the super class EventObject in java.util package
  EventObject(Object src)
  where, src is the object that generates this event
- EventObject contains two methods:
  - **getSource()**: returns the source of the event
  - **toString()**: returns the string equivalent of the event
- The immediate subclass of EventObject is the AWTEvent class from which all the AWT-based event classes are derived
  - **getID()**: used to determine the event type

Event Classes...

Commonly Used Event Classes

- **ActionEvent**: button pressed, menu item selected, list item double-clicked
- **ItemEvent**: check box or list item clicked, choice selection made, checkable menu item selected
- **KeyEvent**: input receieved from keyboard
- **MouseEvent**: mouse dragged, moved, clicked, pressed or released
- **TextEvent**: value of text area or text field changed

Sources of Events

Sources of Events

- **Button**: generates *action events* when the button is pressed
- **Check Box**: generates *item events* when the check box is selected/deselected
- **Choice**: generates *item events* when the choice is changed
- **List**: generates *action events* when an item is double-clicked. It also generates *item events* when an item is selected/deselected
- **Menu Item**:generates *action events* when a menu item is selected. It also generates *item events* when a checkable menu item is selected/deselected
- **Text Components**: generates *text events* when the user enters a character

Event Listener Interfaces

Event Listener Interfaces

Event Listeners are created by implementing one or more interfaces defined by **java.awt.event** package

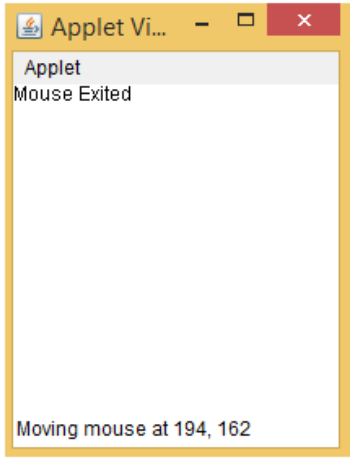
- **ActionListener**: defines a method to receive action events. *actionPerformed()*
- **ItemListener**: defines a method to recognize when the state of an item changes. *itemStateChanged()*
- **KeyListener**: defines methods to recognize when a key is pressed, released or typed. *keyPressed(), keyReleased(), keyTyped()*
- **MouseListener**: defines methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed or is released. *mouseClicked(), mouseExited(), mouseEntered(), mousePressed(), mouseReleased()*
- **MouseMotionListener**: defines methods to recognize when the mouse is dragged or moved. *mouseDragged(), mouseMoved()*
- **TextListener**: defines a method to recognize when a text value changes. *textChanged()*

Event Listener Interfaces...

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*
<APPLET CODE="Test" HEIGHT=200 WIDTH=200>
</APPLET>
*/
public class Test extends Applet implements MouseListener, MouseMotionListener{
String msg=null;
int x=0, y=0;
public void init(){
        addMouseListener(this);
        addMouseMotionListener(this);
        }
public void mouseClicked(MouseEvent m){
        x=0;  y=10;
        msg="Mouse Clicked";
        repaint();
        }
public void mouseEntered(MouseEvent m){
        x=0;  y=10;
        msg="Mouse Entered";
        repaint();
        }
public void mouseExited(MouseEvent m){
        x=0;  y=10;
        msg="Mouse Exited";
        repaint();
        }
```

Event Listener Interfaces...

```
public void mousePressed(MouseEvent m){
        x=m.getX();
        y=m.getY();
        msg="Mouse Down";
        repaint();
        }
public void mouseReleased(MouseEvent m){
        x=m.getX();
        y=m.getY();
        msg="Mouse Up";
        repaint();
        }
public void mouseDragged(MouseEvent m){
        x=m.getX();
        y=m.getY();
        msg="*";
        showStatus("Dragging mouse at "+ x + ", " + y);
        repaint();
        }
public void mouseMoved(MouseEvent m){
        x=m.getX();
        y=m.getY();
        showStatus("Moving mouse at "+ x + ", " + y);
        }
public void paint(Graphics g){
        g.drawString(msg, x,y);
        }
}
```



Event Handling using Adapter Class

Event Handling using Adapter Class

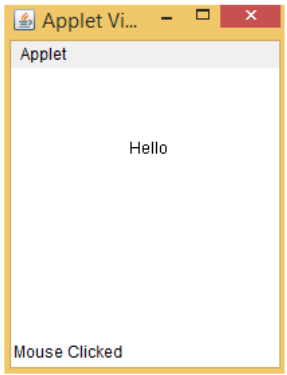
- Adapter class can simplify the creation of event handlers
- An adapter class provides an empty implementation of all methods in an event listener interface
- For listener interfaces containing more than one event handling methods, JDK defines corresponding adapter classes
  Ex: *MouseMotionAdapter class has two methods, mouseDragged() and mouseMoved()*
- As the adapter classes have already provided definitions with empty bodies, you don't have to provide implementations for all the methods again; i.e. we only need to override our methods of interest
- Commonly used adapter classes: **KeyAdapter, MouseAdapter, MouseMotionAdapter**

Event Handling using Adapter Class...

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<APPLET CODE="AdapterDemo" HEIGHT=200 WIDTH=200>
</APPLET>
*/
public class AdapterDemo extends Applet{
int x=0, y=0;
public void init(){
        addMouseListener(new MouseDemo(this));
        addMouseMotionListener(new MouseMotionDemo(this));
        }
public void paint(Graphics g){
        g.drawString("Hello", x, y);
        }
}

class MouseDemo extends MouseAdapter{
AdapterDemo d;
MouseDemo(AdapterDemo d){  this.d=d;  }
public void mouseClicked(MouseEvent m){
        d.showStatus("Mouse Clicked");
        }
}

class MouseMotionDemo extends MouseMotionAdapter{
AdapterDemo d;
MouseMotionDemo(AdapterDemo d){  this.d=d;  }
public void mouseMoved(MouseEvent m){
        d.x=m.getX();     d.y=m.getY();
        d.repaint();
        }
}
```

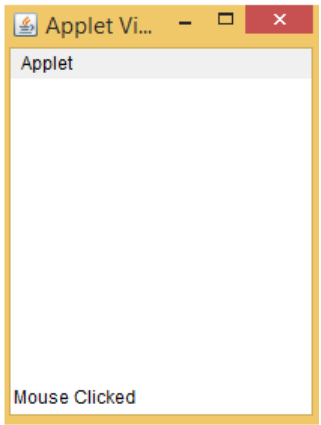


Event Handling using Inner Classes

- Inner class is a class defined within another class or even within an expression

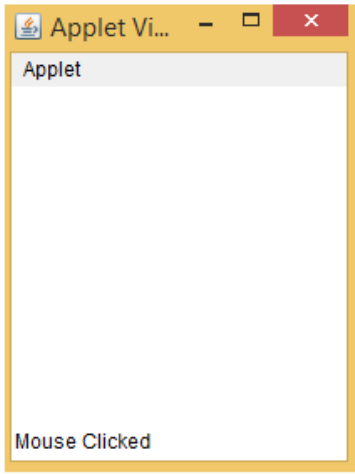
```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<APPLET CODE="InnerDemo" HEIGHT=200 WIDTH=200>
</APPLET>
*/
public class InnerDemo extends Applet{
int x=0, y=0;
public void init(){
        addMouseListener(new MouseDemo(this));
        }
}

class MouseDemo extends MouseAdapter{
InnerDemo id;
MouseDemo(InnerDemo id){
        this.id=id;
        }
public void mouseClicked(MouseEvent m){
        id.showStatus("Mouse Clicked");
        }
}
```



Event Handling using Inner Classes...

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<APPLET CODE="InnerDemo" HEIGHT=200 WIDTH=200>
</APPLET>
*/
public class InnerDemo extends Applet{
public void init(){
            addMouseListener(new MouseDemo());
            }
class MouseDemo extends MouseAdapter{
public void mouseClicked(MouseEvent m){
            showStatus("Mouse Clicked");
            }
}
}
```



Event Handling using Anonymous Inner Classes

Event Handling using Anonymous Inner Classes

- An anonymous inner class is one that is not assigned a name
- new MouseAdapter()... indicates to the compiler that the code between the braces defines an anonymous inner class. This anonymity helps in eliminating the unnecessary named objects

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<APPLET CODE="AInnerDemo" HEIGHT=200 WIDTH=200>
</APPLET>
*/
public class AInnerDemo extends Applet{
public void init(){
        addMouseListener(new MouseAdapter(){
                public void mouseClicked(MouseEvent m){
                        showStatus("Mouse Clicked");
                        }
                });
        }
}
```