EVENT HANDLING

Prepared by

Dr. Rajesh Kumar Ojha
Asst. Prof., CSE, Silicon University

Delegation Event Model

- A **source** generates an event and sends it to one or more **listeners**.
- Listener simply waits until it receives an event. Once received, the listener processes the event and then returns.
- Listeners must register with a source in order to receive an event notification.
- An event is an object that describes a state change in a source.It can be generated as a consequence of a person interacting with the elements in a graphical user interface.

Event Sources

- A source is an object that generates an event. This occurs when the internal state of that object changes in some way.

- A source must register listeners in order for the listeners to receive notifications about a specific type of event.

```
public void addTypeListener(TypeListener el)
```

- For example, the method that registers a keyboard event listener is called addKeyListener( ). The method that registers a mouse motion listener is called addMouseMotionListener( ).

- A source allows a listener to unregister an interest in a specific type of event.

public void remove*Type*Listener(*Type*Listener *el*)

Event Classes

- At the root of the Java event class hierarchy is EventObject, which is in java.util.

- Its one constructor is:

  EventObject(Object *src*)

- EventObject contains two methods : **getSource( )** and **toString( ).**

  Object getSource( )

- toString( ) returns the string equivalent of the event.

- EventObject is a superclass of all events.

- AWTEvent is a superclass of all AWT events that are handled by the delegation event model.

ActionEvent Class

- ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.

ActionEvent(Object *src*, int *type*, String *cmd*)
ActionEvent(Object *src*, int *type*, String *cmd*, int *modifiers*)
ActionEvent(Object *src*, int *type*, String *cmd*, long *when*, int *modifiers*)

- *src* is a reference to the object that generated this event. The type of the event is specified by *type*, and its command string is *cmd*.

- *modifiers* indicates which modifier keys (ALT, CTRL, META, and/or SHIFT) were pressed when the event was generated.

- *ActionEvent* object by using the *getActionCommand( )* method.

```
String getActionCommand( )
```

- *getModifiers( )* method returns a value that indicates which modifier keys (ALT, CTRL, META, and/or SHIFT) were pressed when the event was generated.

```
int getModifiers( )
```

- *getWhen( )* that returns the time at which the event took place. This is called the event's *timestamp*.

```
long getWhen( )
```

KeyEvent Class

- KeyEvent is generated when keyboard input occurs.
- There are three types of key events :**KEY_PRESSED, KEY_RELEASED,** and **KEY_TYPED.**
- KeyEvent is a subclass of InputEvent. Here are two of its constructors:

KeyEvent(Component *src*, int *type*, long *when*, int *modifiers*, int *code*)
KeyEvent(Component *src*, int *type*, long *when*, int *modifiers*, int *code*, char *ch*)

- KeyEvent class defines several methods:

char getKeyChar( )
int getKeyCode( )

MouseEvent Class

- Eight types of mouse events.

- The MouseEvent class defines the following integer constants

MOUSE_CLICKED	The user clicked the mouse.
MOUSE_DRAGGED	The user dragged the mouse.
MOUSE_ENTERED	The mouse entered a component.
MOUSE_EXITED	The mouse exited from a component.
MOUSE_MOVED	The mouse moved.
MOUSE_PRESSED	The mouse was pressed.
MOUSE_RELEASED	The mouse was released.
MOUSE_WHEEL	The mouse wheel was moved (Java 2, v1.4).

- MouseEvent is a subclass of InputEvent.

```
MouseEvent(Component src, int type, long when, int modifiers,
           int x, int y, int clicks, boolean triggersPopup)
```

- coordinates of the mouse are passed in *x and y*.

- click count is passed in *clicks*.

- *triggersPopup* flag indicates if this event causes a pop-up menu to appear on this platform.

- We can use the getPoint( ) method to obtain the coordinates

```
Point getPoint( )
```

- translatePoint( ) method changes the location of the event.

  void translatePoint(int *x*, int *y*)

- *getClickCount( )* method obtains the number of mouse clicks for this event.

  int getClickCount( )

Event Listener Interfaces

- Listeners are created by implementing one or more of the interfaces defined by the **Java.awt.event** package.

ActionListener	Defines one method to receive action events.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.

ActionListener Interface

- This interface defines the actionPerformed( ) method that is invoked when an action event occurs.

```
void actionPerformed(ActionEvent ae)
```

KeyListener Interface

- This interface defines three methods.
- **keyPressed( )** and **keyReleased( )** methods are invoked when a key is pressed and released, respectively.
- **keyTyped( )** method is invoked when a character has been entered.

```
void keyPressed(KeyEvent ke)
void keyReleased(KeyEvent ke)
void keyTyped(KeyEvent ke)
```

MouseListener Interface

- This interface defines five methods.
- If the mouse is pressed and released at the same point, **mouseClicked( )** is invoked.
- When the mouse enters a component, the **mouseEntered( )** method is called. When it leaves**, mouseExited( ) is called.**
- **mousePressed( )** and **mouseReleased( )** methods are invoked when the mouse is pressed and released, respectively

```
void mouseClicked(MouseEvent me)
void mouseEntered(MouseEvent me)
void mouseExited(MouseEvent me)
void mousePressed(MouseEvent me)
void mouseReleased(MouseEvent me)
```

Using the Delegation Event Model

- Implement the appropriate interface in the listener so that it will receive the type of event desired.

- Implement code to register and unregister (if necessary) the listener as a recipient for the event notifications.

Handling Mouse Events

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="MouseEvents" width=300 height=100>
  </applet>
*/

public class MouseEvents extends Applet
  implements MouseListener, MouseMotionListener {

  String msg = "";
  int mouseX = 0, mouseY = 0; // coordinates of mouse
```

```
public void init() {
    addMouseListener(this);
    addMouseMotionListener(this);
}

// Handle mouse clicked.
public void mouseClicked(MouseEvent me) {
  // save coordinates
  mouseX = 0;
  mouseY = 10;
  msg = "Mouse clicked.";
  repaint();
}
```

```
// Handle mouse entered.
public void mouseEntered(MouseEvent me) {
  // save coordinates
  mouseX = 0;
  mouseY = 10;
  msg = "Mouse entered.";
  repaint();
}

// Handle mouse exited.
public void mouseExited(MouseEvent me) {
  // save coordinates
  mouseX = 0;
  mouseY = 10;
  msg = "Mouse exited.";
  repaint();
}
```

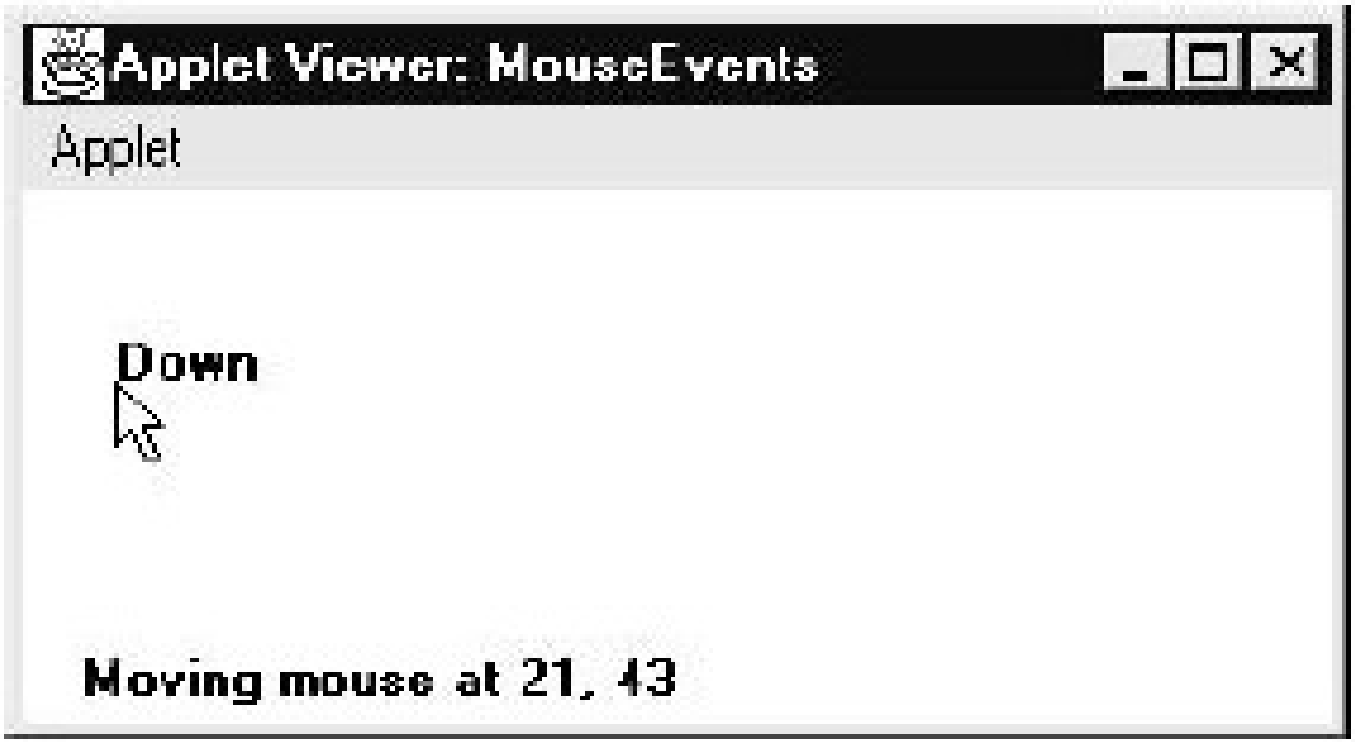
```
public void mousePressed(MouseEvent me) {
   // save coordinates
   mouseX = me.getX();
   mouseY = me.getY();
   msg = "Down";
   repaint();
}


// Handle button released.
public void mouseReleased(MouseEvent me) {
   // save coordinates
   mouseX = me.getX();
   mouseY = me.getY();
   msg = "Up";
   repaint();
}
```

```
// Handle mouse dragged.
public void mouseDragged(MouseEvent me) {
  // save coordinates
  mouseX = me.getX();
  mouseY = me.getY();
  msg = "*";
  showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
  repaint();
}

// Handle mouse moved.
public void mouseMoved(MouseEvent me) {
  // show status
  showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
}
```

```
  // Display msg in applet window at current X,Y location.
  public void paint(Graphics g) {
    g.drawString(msg, mouseX, mouseY);
  }
}
```



Thank you