CS205 Object Oriented Programming in Java

Module 4 - **Advanced features of Java**

(Part 8)

Prepared by

Renetha J.B.

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

Topics

- ☑ **Event handling:**
  - ❑ Sources of Events
  - ❑ Event Listener Interfaces
  - ❑ Using the Delegation Model

Sources of Events

- The components in user interface that can generate the events are the **sources of the event.**
- Event Source are:
  - ☑ Button
  - ☑ Check box
  - ☑ Choice
  - ☑ List
  - ☑ Menu Item
  - ☑ Scroll bar
  - ☑ Text components
  - ☑ Window

Event Source & Description

Button	• Generates **action events** when the button is **pressed.**
Check box	• Generates **item events** when the check box **is selected or deselected.**
Choice	• Generates **item events** when the choice is **changed**.
List	• Generates **action events** when an item is **double-clicked**; generates **item events** when an item is **selected or deselected.**
Menu Item	• Generates **action events** when a **menu item is selected**; generates **item events** when a **checkable menu item is selected or deselected.**
Scroll bar	• Generates **adjustment events** when the scroll bar is **manipulated**.
Text components	• Generates **text events** when the user **enters a character.**
Window	• Generates **window events** when a window is **activated, closed, deactivated, deiconified, iconified, opened, or quit.**

Sources of Events(contd.)

- Any class derived from **Component,** such as Applet, can generate events.
  - For example, we can receive key and mouse events from an applet.

Event Listener Interfaces

- Listeners are created by **implementing** one or more of the **interfaces** defined by the **java.awt.event** package,
- When an event occurs,
  - the event source invokes the appropriate method defined by the listener and provides an event object as its argument

Event class	**Event Listener interface**
ActionEvent	ActionListener
AdjustmentEvent	AdjustmentListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener
InputEvent	
ItemEvent	ItemListener
KeyEvent	KeyListener
MouseEvent	MouseListener
	MouseMotionListener
MouseWheelEvent	MouseWheelListener
TextEvent	TextListener
WindowEvent	WindowFocusListener
	WindowListener

Event listeners & Description

ActionListener	• Defines one method to receive **action events.**
AdjustmentListener	• Defines one method to receive **adjustment events.**
ComponentListener	• Defines four methods to recognize when a component is **hidden, moved, resized, or shown**.
ContainerListener	• Defines two methods to recognize when a **component** is **added to or removed** from a container.
FocusListener	• Defines two methods to recognize when a component **gains or loses keyboard focus**.
ItemListener	• Defines one method to recognize when the **state of an item changes.**

Event listeners & Description(contd.)

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.**
MouseMotionListener	• Defines two methods to recognize when the mouse is **dragged or moved.**
MouseWheelListener	• Defines one method to recognize when the **mouse wheel is moved.**
TextListener	• Defines one method to recognize when **a text value changes.**
WindowFocusListener	• Defines two methods to recognize when a **window gains or loses input focus**.
WindowListener	• Defines seven methods to recognize when a window is **activated, closed, deactivated, deiconified, iconified, opened, or quit.**

ActionListener Interface

- ActionListener interface defines the **actionPerformed( ) method that is invoked when an action event** occurs.
  - **button is pressed**
  - **a list item is double-clicked**
  - **menu item is selected.**

void **actionPerformed**(**ActionEvent** *ae*)

AdjustmentListener Interface

- The **AdjustmentListener Interface defines** the **adjustmentValueChanged( )** method that is invoked when an **adjustment event** occurs.
  - **scroll bar is manipulated**.

```
void adjustmentValueChanged(AdjustmentEvent ae)
```

ComponentListener Interface

- **The ComponentListener Interface** defines four methods that are invoked when a component is resized, moved, shown, or hidden.

void component**Resized**(**ComponentEvent** *ce)*

void component**Moved**(**ComponentEvent** *ce)*

void component**Shown**(**ComponentEvent** *ce)*

void component**Hidden**(**ComponentEvent***ce)*

ContainerListener Interface

- The **ContainerListener Interface** contains two methods.
- When a component is added to a container,
  - **componentAdded( ) is invoked.**
- When a component is removed from a container,
  - **componentRemoved( ) is invoked.**

```
void componentAdded(ContainerEvent ce)
```

```
void componentRemoved(ContainerEvent ce)
```

FocusListener Interface

- **The FocusListener Interface** defines two methods.
- When a component obtains keyboard focus,
  - **focusGained( )** is invoked.
- When a component loses keyboard focus,
  - **focusLost( )** is called.

void focus**Gained**(**FocusEvent** *fe)*

void focus**Lost**(**FocusEvent** *fe)*

ItemListener Interface

- The **ItemListener Interface** defines the **itemStateChanged( )** method that is invoked when the state of an item changes.

void itemStateChanged(**ItemEvent** *ie)*

KeyListener Interface

- The **KeyListener Interface** defines three methods.
- When a key is pressed , **keyPressed( )** method is invoked
- When a key is released, **keyReleased( )** method is invoked
- When a character has been entered **keyTyped( )** method is invoked
- For example, if a user presses and releases the letter **A** key in keyboard, three events are generated in sequence:
  - key pressed
  - key typed,
  - key released.
- If a user presses and releases the **HOME** key in keyboard, two key events are generated in sequence:
  - key pressed
  - key released

KeyListener Interface(contd.)

- The general forms of methods are :

```
void keyPressed(KeyEvent ke)
```

```
void keyReleased(KeyEvent ke)
```

```
void keyTyped(KeyEvent ke)
```

MouseListener Interface

- **The MouseListener 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.**
  - When the mouse is pressed **the mousePressed( ) method is invoked**
  - When the mouse is released, **mouseReleased( ) methods is invoked**

void mouse**Clicked**(**MouseEvent** *me)*

void mouse**Entered**(**MouseEvent** *me)*

void mouse**Exited**(**MouseEvent** *me)*

void mouse**Pressed**(**MouseEvent** *me)*

void mouse**Released**(**MouseEvent** *me)*

MouseMotionListener Interface

- The **MouseMotionListener Interface** defines two methods.

void **mouseDragged**(**MouseEvent** *me*)

void **mouseMoved**(**MouseEvent** *me*)

mouseDragged( )	• **called <u>multiple times</u>** as the mouse is **dragged**.
mouseMoved( )	• **called <u>multiple times</u> as the mouse** is **moved**.

MouseWheelListener Interface

- The **MouseWheelListener Interface** defines the **mouseWheelMoved( ) method** that is invoked **when the mouse** wheel is **moved**.

```
void mouseWheelMoved(MouseWheelEvent mwe)
```

TextListener Interface

- The **TextListener Interface** defines the **textChanged( )** method that is invoked **when a change occurs** in a **text area** or **text field.**

```
void textChanged(TextEvent te)
```

WindowFocusListener Interface

- The **WindowFocusListener Interface** defines two methods:
  - **windowGainedFocus( ) -** called when a window gains input focus
  - **windowLostFocus( ) -** called when a window loses input focus

void window**GainedFocus**(**WindowEvent** *we)*

void window**LostFocus**(**WindowEvent** *we)*

WindowListener Interface

- **The WindowListener Interface** defines seven methods.

void window**Activated**(**WindowEvent** *we)*

void window**Closed**(**WindowEvent** *we)*

void window**Closing**(**WindowEvent***we)*

void window**Deactivated**(**WindowEvent** *we)*

void window**Deiconified**(**WindowEvent** *we*)

void window**Iconified**(**WindowEvent** *we)*

void window**Opened**(**WindowEvent** *we)*

WindowListener Interface(contd.)

windowActivated( )	• invoked when a window is **activated**
windowDeactivated( )	• invoked when a window is **deactivated**
windowIconified( )	• invoked if a window is **iconified**,
windowDeiconified( )	• Invoked when a window is **deiconified**
windowOpened( )	• When a window is **opened**
windowClosed( )	• Invoked when a window is **closed**
windowClosing( )	• Invoked when a window is **being closed**.

Simple applet program

```
import java.awt.*;
import java.applet.*;
/*
<applet  code="Sampleapplet"   width=300  height=50>
</applet>
*/
public class Sampleapplet extends Applet{

String msg;
public void init()
 {
setBackground(Color.cyan);
setForeground(Color.red);

}
public void paint(Graphics g) {
msg ="Welcome to first applet";
g.drawString(msg, 10, 30);
}
}
```

```
//TO COMPILE....
 javac Sampletest.java
//TO RUN........
appletviewer Sampletest.java
```

Using the Delegation Event Model

- To use the delegation event model follow two steps:

1. **Implement the appropriate interface in the listener** so that it will receive the type of event desired.
2. **Implement code to register and unregister** (if necessary) **the listener** as a recipient for the event notifications.

Using the Delegation Event Model(contd.)

- A source may generate several types of events.
  - Each event must be registered separately.
- An object may register to receive several types of events,
  - but it must implement all of the interfaces that are required to receive these events

Handling Mouse Events

- To handle **mouse events**, we must **implement** the **MouseListener** and the **MouseMotionListener** interfaces.
  - We can implement **MouseWheelListener**, also.

Handling Mouse Events-Applet program example.

- Design an applet program with following features
  - It **displays the current coordinates of the mouse** in the applet's ***status window.***
  - Each time a **button** is **pressed**, the word "Down" is displayed at the location of the ***mouse pointer.***
  - Each time the **button is released**, the word "Up" is shown.
  - If a **button** is **clicked**, the message "Mouse clicked" is displayed in the upperleft corner of the ***applet display area***.
  - As the mouse **enters or exits** the applet window, a message is displayed in the upper-leftcorner of the ***applet display area***.
  - When **dragging the mouse,** a * is shown, which tracks with the mouse pointer as it is dragged.
  - **mouseX and mouseY** coordinates are then used by **paint( ) to** display output at the point of these occurrences.

Handling Mouse Events-Applet program example.

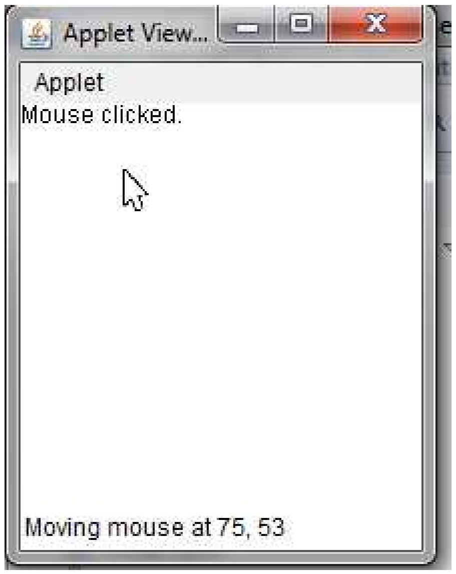
```
import java.awt.event.*;
import java.applet.*;
import java.awt.*;
/*
<applet code="MouseEvents" width=500 height=500>
</applet>
*/
public class MouseEvents extends Applet implements
    MouseListener, MouseMotionListener {
String msg = "";
int mouseX = 0, mouseY = 0;
public void init()
 {
addMouseListener(this);
addMouseMotionListener(this);
}
```

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

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

```
public void mouseExited(MouseEvent me)
        {
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse exited.";
        repaint();
        }
.
public void mousePressed(MouseEvent me)
    {
    mouseX = me.getX();
    mouseY = me.getY();
    msg = "Down";
    repaint();
    }
```

```
public void mouseReleased(MouseEvent me)
{
mouseX = me.getX();
mouseY = me.getY();
msg = "Up";
repaint();
}
public void mouseDragged(MouseEvent me)
{
mouseX = me.getX();
mouseY = me.getY();
msg= "*";
showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
repaint();
}
```

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

public void paint(Graphics g) {
g.drawString(msg, mouseX, mouseY);
}
}
```



```
Mouse clicked.
Moving mouse at 75, 53
```

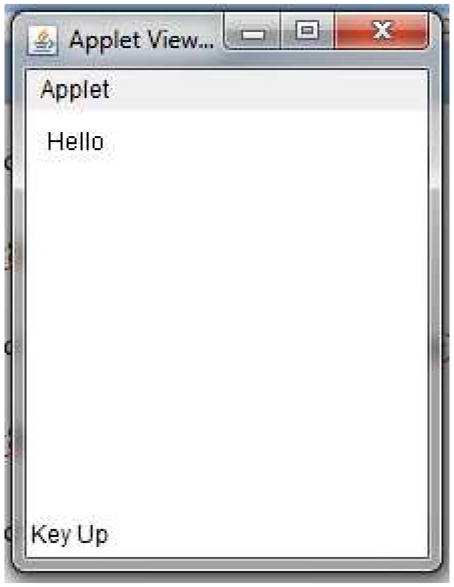
Handling Keyboard Events

- To handle **keyboardevents**, we must **implement** the **KeyListener interface**

Handling Keyboard Events- Applet Example

- When a key is pressed, a **KEY_PRESSED event is generated.**
  - This results in a call to the **keyPressed( )** event handler.
- When the key is released, a **KEY_RELEASED event is generated and**
  - **keyReleased( ) handler is executed.**
- If a character is generated by the keystroke, then **a KEY_TYPED event is sent and**
  - **keyTyped( ) handler is invoked.**
- Thus, each time the user presses a key, at least two and often three events are generated.
- Implement an applet program that echoes keystrokes to the applet window and shows the pressed/released status of each key in the status window.

```
// Demonstrate the key event handlers.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="SimpleKey" width=300 height=100>
</applet>
*/
public class SimpleKey extends Applet  implements KeyListener {
String msg = "";
int X = 10, Y = 20;
public void init()
{
addKeyListener(this);
}
public void keyPressed(KeyEvent ke)
 {
showStatus("Key Down");
}
public void keyReleased(KeyEvent ke)
 {
showStatus("Key Up");
}
public void keyTyped(KeyEvent ke)
 {
msg += ke.getKeyChar();
repaint();
}
public void paint(Graphics g)
 {
g.drawString(msg, X, Y);
}
}
```



```
Hello
```

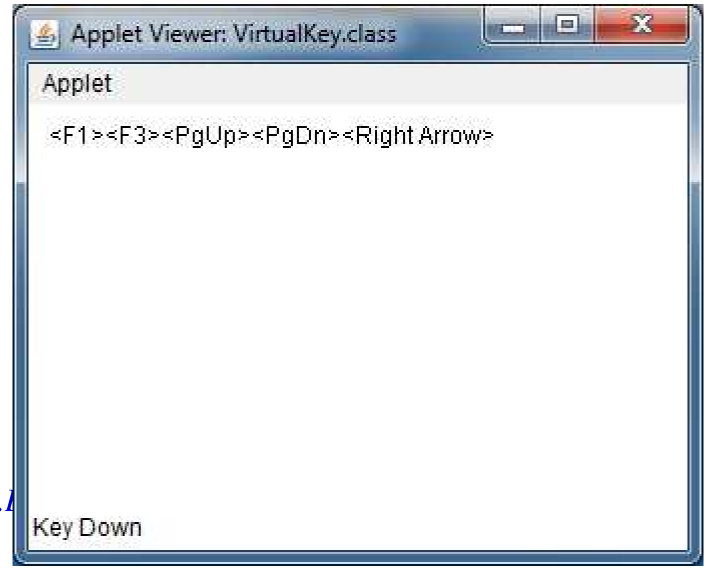
```
Key Up
```

Virtual key display

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="KeyEvents" width=300 height=100>
</applet>
*/
public class VirtualKey extends Applet  implements KeyListener {
String msg = "";
int X = 10, Y = 20;
public void init()
{
addKeyListener(this);
}
```

```
public void keyPressed(KeyEvent ke) {
showStatus("Key Down");
int key = ke.getKeyCode();
switch(key) {
case KeyEvent.VK_F1:
msg += "<F1>";
break;
case KeyEvent.VK_F2:
msg += "<F2>";
break;
case KeyEvent.VK_F3:
msg += "<F3>";
break;
case KeyEvent.VK_PAGE_DOWN:
msg += "<PgDn>";
break;
case KeyEvent.VK_PAGE_UP:
msg += "<PgUp>";
break;
case KeyEvent.VK_LEFT:
msg += "<Left Arrow>";
break;
case KeyEvent.VK_RIGHT:
msg += "<Right Arrow>";
break;
}
repaint();
}
public void keyReleased(KeyEvent ke)
{
showStatus("Key Up");
}
```

```
public void keyTyped(KeyEvent ke)
    {
msg += ke.getKeyChar();
repaint();
}
public void paint(Graphics g)
    {
    g.drawString(msg, X, Y);
    }
}
```



```
<F1><F3><PgUp><PgDn><Right Arrow>
Key Down
```

Reference

- Herbert Schildt, Java: **The Complete Reference, 8/e, Tata McGraw Hill, 2011**.