# *Course: Object Based Modeling*
# *Code: CS-33105*
# *Branch: MCA-3*

## *Lecture 17: Event Handling*

*Dr. J Sathish Kumar (JSK)*
*(Faculty & Coordinator)*

Department of Computer Science and Engineering

Motilal Nehru National Institute of Technology Allahabad,
Prayagraj-211004

## Introduction

- Event handling is fundamental to Java programming because it is integral to the creation of many kinds of applications, including applets and other types of GUI-based programs.
- Events are supported by a number of packages, including **java.util**, **java.awt**, and **java.awt.event**.
- There are several types of events, including those generated by the mouse, the keyboard, and various GUI controls, such as a push button, scroll bar, or check box.

# Event Sources

- A *source* is an object that generates an event.
- A source must register listeners in order for the listeners to receive notifications about a specific type of event.
- Each type of event has its own registration method.
- Here is the general form:
  *public void addTypeListener (TypeListener el)*
  - *Type* is the name of the event, and *el* is a reference to the event listener.
- For example,
  - The method that registers a keyboard event listener is called **addKeyListener( )**.
  - The method that registers a mouse motion listener is called **addMouseMotionListener( )**.

# Event Sources

- Some sources may allow only one listener to register.
- The general form of such a method is this:
  public void add*Type*Listener(*Type*Listener *el*)
  throws java.util.TooManyListenersException
- When such an event occurs, the registered listener is notified.
- This is known as *unicasting* the event.
- A source must also provide a method that allows a listener to unregister an interest in a specific type of event.
- The general form of such a method is this:
  public void remove*Type*Listener(*Type*Listener *el*)
- Here, *Type* is the name of the event, and *el* is a reference to the event listener.
- For example, to remove a keyboard listener, you would call **removeKeyListener( )**.

# Event Classes

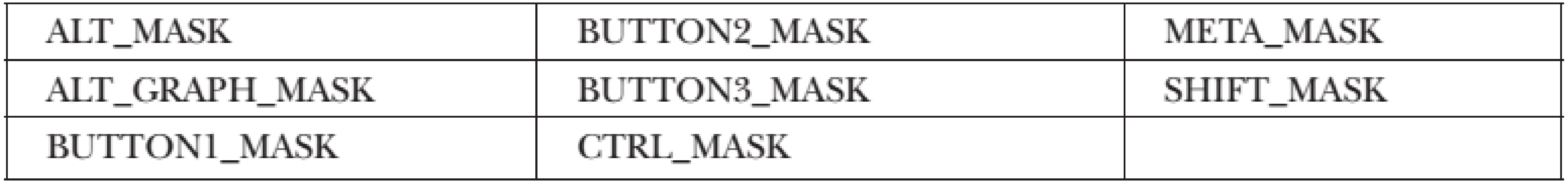
- The classes that represent events are at the core of Java’s event handling mechanism.
- The most widely used events at the time of this writing are those defined by the AWT and those defined by Swing.
- At the root of the Java event class hierarchy is **EventObject**, which is in **java.util**. It is the superclass for all events.
- **AWTEvent** is a superclass of all AWT events that are handled by the delegation event model.

## Commonly Used Event Classes in java.awt.event

| Event Class | Description |
|---|---|
| ActionEvent | Generated when a button is pressed, a list item is double-clicked, or a menu item is selected. |
| AdjustmentEvent | Generated when a scroll bar is manipulated. |
| ComponentEvent | Generated when a component is hidden, moved, resized, or becomes visible. |
| ContainerEvent | Generated when a component is added to or removed from a container. |
| FocusEvent | Generated when a component gains or loses keyboard focus. |
| InputEvent | Abstract superclass for all component input event classes. |
| ItemEvent | Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected. |
| KeyEvent | Generated when input is received from the keyboard. |
| MouseEvent | Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component. |
| MouseWheelEvent | Generated when the mouse wheel is moved. |
| TextEvent | Generated when the value of a text area or text field is changed. |
| WindowEvent | Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. |

# The InputEvent Class

- The abstract class **InputEvent** is a subclass of **ComponentEvent** and is the superclass for component input events.
- Its subclasses are **KeyEvent** and **MouseEvent**.
- **InputEvent** defines several integer constants that represent any modifiers, such as the control key being pressed, that might be associated with the event.
- Originally, the **InputEvent** class defined the following eight values to represent the modifiers:



| | | |
|---|---|---|
| ALT_MASK | BUTTON2_MASK | META_MASK |
| ALT_GRAPH_MASK | BUTTON3_MASK | SHIFT_MASK |
| BUTTON1_MASK | CTRL_MASK | |

# The InputEvent Class

- Because of possible conflicts between the modifiers used by keyboard events and mouse events, and other issues, the following extended modifier values were added:

| | | |
|---|---|---|
| ALT_DOWN_MASK | BUTTON2_DOWN_MASK | META_DOWN_MASK |
| ALT_GRAPH_DOWN_MASK | BUTTON3_DOWN_MASK | SHIFT_DOWN_MASK |
| BUTTON1_DOWN_MASK | CTRL_DOWN_MASK | |

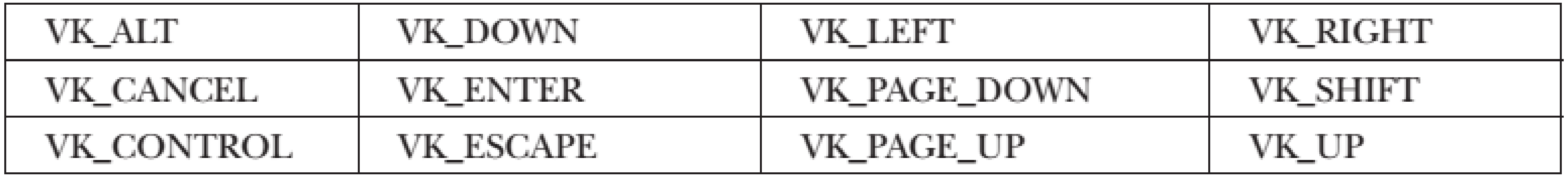
- To test if a modifier was pressed at the time an event is generated, use the **isAltDown( )**, **isAltGraphDown( )**, **isControlDown( )**, **isMetaDown( )**, and **isShiftDown( )** methods. All these methods are boolean.

# The ItemEvent Class

- An **ItemEvent** is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected.
- There are two types of item events, which are identified by the following integer constants:
  - DESELECTED The user deselected an item.
    SELECTED The user selected an item.
- In addition, **ItemEvent** defines one integer constant, **ITEM_STATE_CHANGED**, that signifies a change of state.
- **ItemEvent** has this constructor:
  *ItemEvent(ItemSelectable src, int type, Object entry, int state)*

## The KeyEvent Class

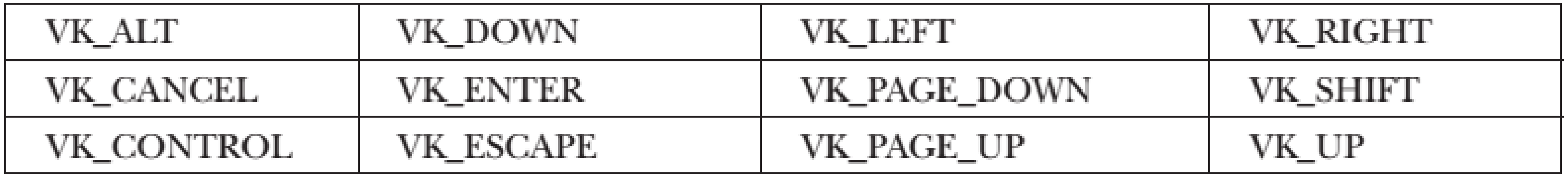
- A **KeyEvent** is generated when keyboard input occurs. There are three types of key events, which are identified by these integer constants: **KEY_PRESSED**, **KEY_RELEASED**, and **KEY_TYPED**.
- There are many integer constants that are defined by **KeyEvent**.
- For example, **VK_0** through **VK_9** and **VK_A** through **VK_Z** define the ASCII equivalents of the numbers and letters.
- The **VK** constants specify *virtual key codes* and are independent of any modifiers, such as control, shift, or alt.



| | | | |
|---|---|---|---|
| VK_ALT | VK_DOWN | VK_LEFT | VK_RIGHT |
| VK_CANCEL | VK_ENTER | VK_PAGE_DOWN | VK_SHIFT |
| VK_CONTROL | VK_ESCAPE | VK_PAGE_UP | VK_UP |

# The KeyEvent Class

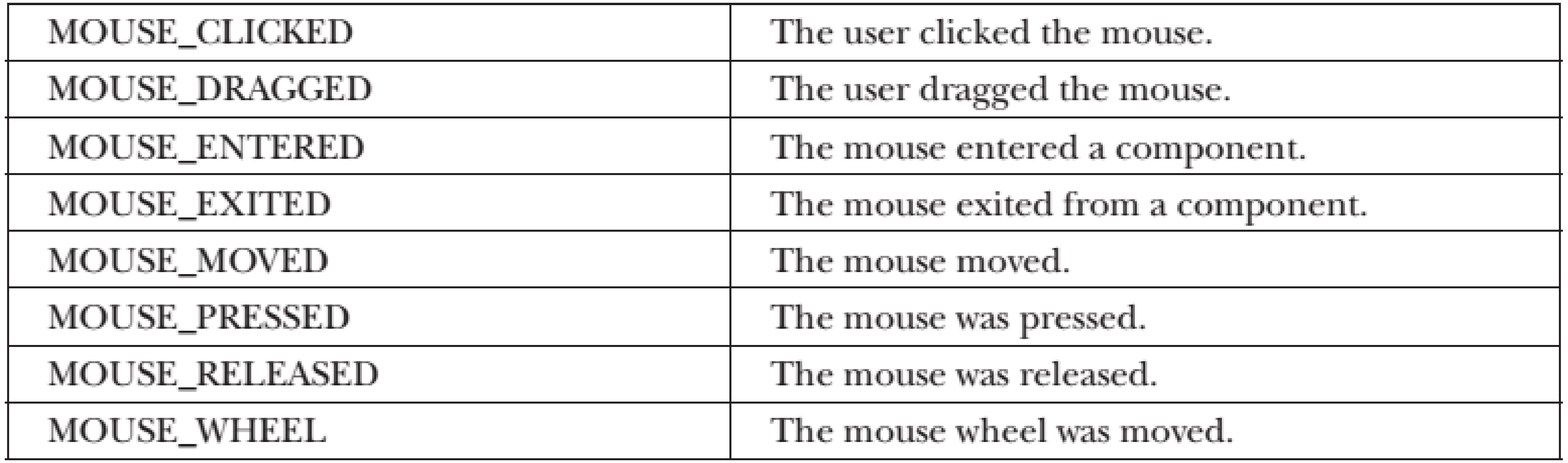
- A **KeyEvent** is generated when keyboard input occurs. There are three types of key events, which are identified by these integer constants: **KEY_PRESSED**, **KEY_RELEASED**, and **KEY_TYPED**.
- There are many integer constants that are defined by **KeyEvent**.
- For example, **VK_0** through **VK_9** and **VK_A** through **VK_Z** define the ASCII equivalents of the numbers and letters.
- The **VK** constants specify *virtual key codes* and are independent of any modifiers, such as control, shift, or alt.



| | | | |
|---|---|---|---|
| VK_ALT | VK_DOWN | VK_LEFT | VK_RIGHT |
| VK_CANCEL | VK_ENTER | VK_PAGE_DOWN | VK_SHIFT |
| VK_CONTROL | VK_ESCAPE | VK_PAGE_UP | VK_UP |

# The MouseEvent Class

- There are eight types of mouse events.
- **MouseEvent** is a subclass of **InputEvent**.
- The **MouseEvent** class defines the following integer constants that can be used to identify them



| | |
|---|---|
| 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. |

# The MouseEvent Class

- One of its constructors:
  MouseEvent(Component *src*, int *type*, long *when*, int *modifiers*,
  int *x*, int *y*, int *clicks*, boolean *triggersPopup*)
- Two commonly used methods in this class are **getX( )** and **getY( )**.
- These return the X and Y coordinates of the mouse within the component when the event occurred.
- Alternatively, you can use the **getPoint( )** method to obtain the coordinates of the mouse
- The **translatePoint( )** method changes the location of the event.
- The **getClickCount( )** method obtains the number of mouse clicks for this event.
- The **isPopupTrigger( )** method tests if this event causes a pop-up menu to appear on this platform.
- Also available are three methods that obtain the coordinates of the mouse relative to the screen rather than the component.
- They are Point getLocationOnScreen( ), int getXOnScreen( ), int getYOnScreen( )

# Example #1

```
// Demonstrate the mouse event handlers.
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();
}
```

# Example #1

```
// Handle mouse exited.

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

// Handle button pressed.
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);
  }
}
```

## Example #2

```
// 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; // output coordinates

   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();
   }

   // Display keystrokes.
   public void paint(Graphics g) {
     g.drawString(msg, X, Y);
   }
 }
```