# Programming in Java

## *Lecture 20: Event Handling and AWT*

***Mahesh Kumar***
Assistant Professor (Adhoc)

Department of Computer Science
Acharya Narendra Dev College
University of Delhi

Course webpage
[ http://www.mkbhandari.com/mkwiki ]

# Outline

1. The AWT class hierarchy
2. The Delegation Event Model
3. Events
4. Event sources
5. Event classes
6. Event Listeners
7. Relationship between Event sources and Listeners
8. Creating GUI applications using AWT

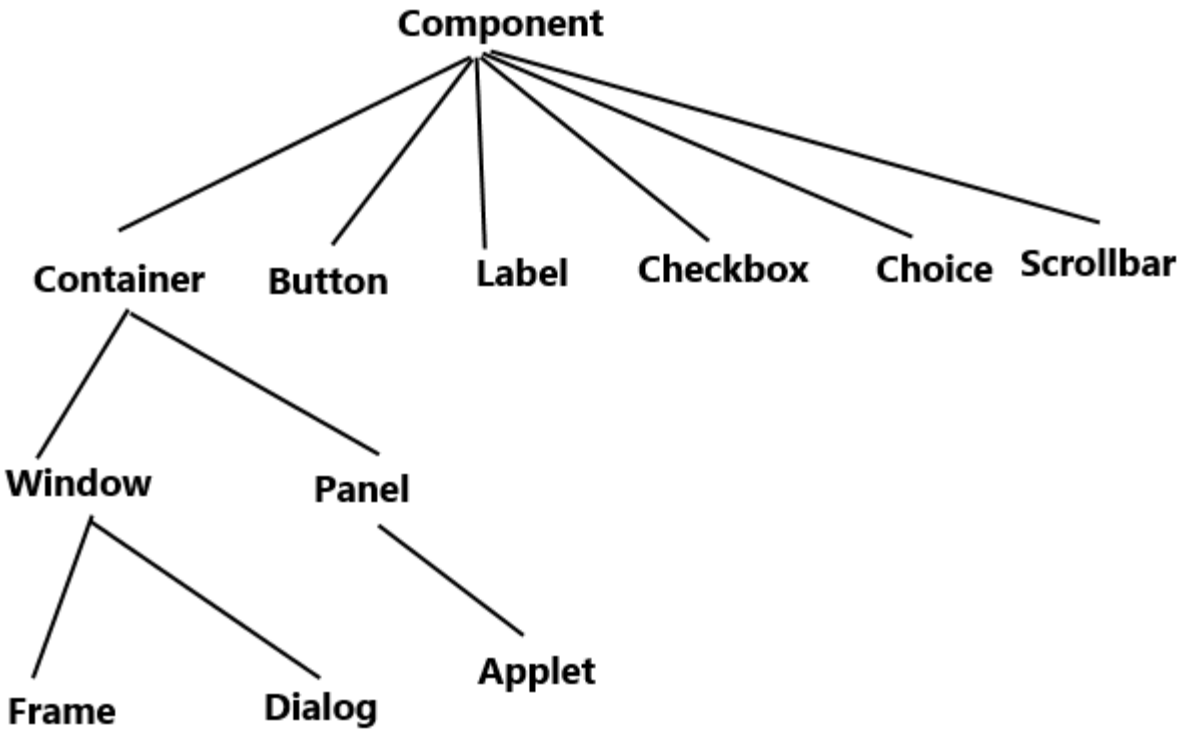
# The AWT(Abstract Window Toolkit)

- Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.

- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

- The **java.awt** package provides classes for AWT api such as **TextField**, **Label**, **TextArea**, **RadioButton**, **CheckBox, Choice**, **List** etc.

# The AWT class Hierarchy

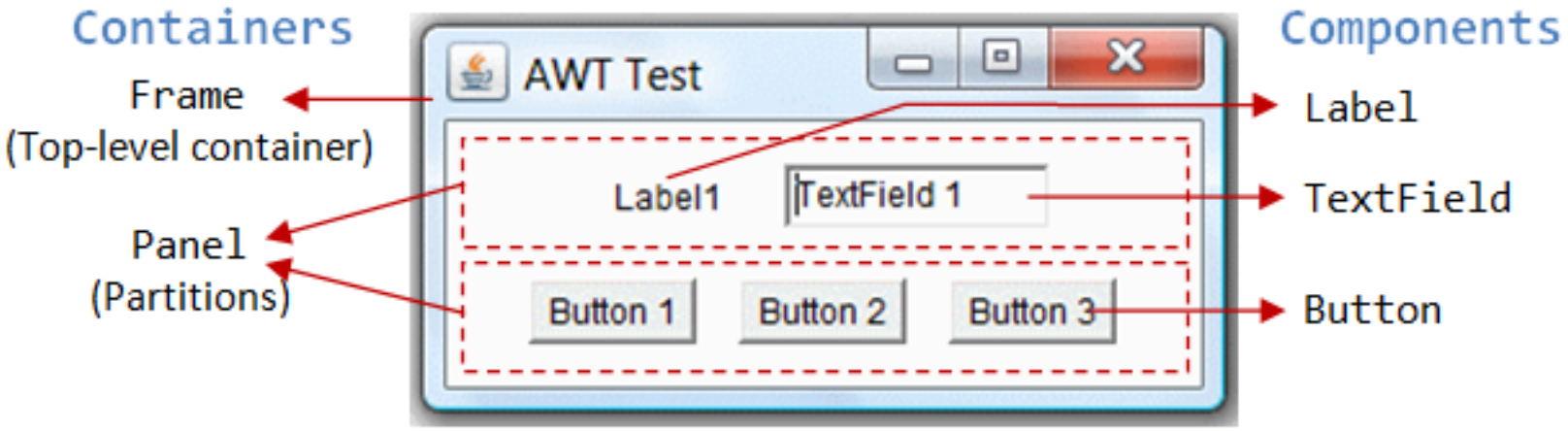


# The AWT class Hierarchy

1. **Component**
   - Components are elementary GUI entities, such as Button, Label, and TextField, etc.
   - In AWT we have classes for each component
   - To have everything placed on a screen to a particular position, we have to add them to a container.
2. **Container** ( a screen wherein we are placing components )
   - Containers, such as Frame and Panel, are used to hold components like buttons, text fields, checkbox etc., in a specific layout (such as FlowLayout or GridLayout)
   - In short a container contains and controls the layout of components.
   - There are four types of containers available in AWT
   - A container itself is a component (as shown in the AWT hierarchy diagram) thus we can add a container inside container.

# The AWT class Hierarchy

- Containers and Components



# The AWT class Hierarchy

- AWT provides many ready-made and reusable GUI **components** in package java.awt.

# The AWT class Hierarchy

3. **Window**
   - The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

4. **Panel**
   - The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

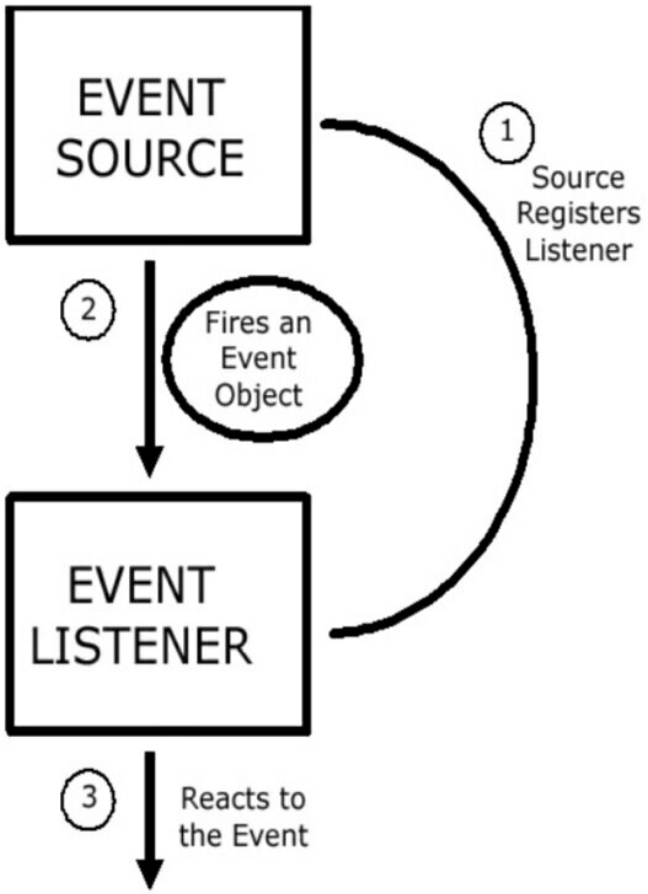
5. **Frame**
   - The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

6. **Dialog**
   - Dialog class has border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class.

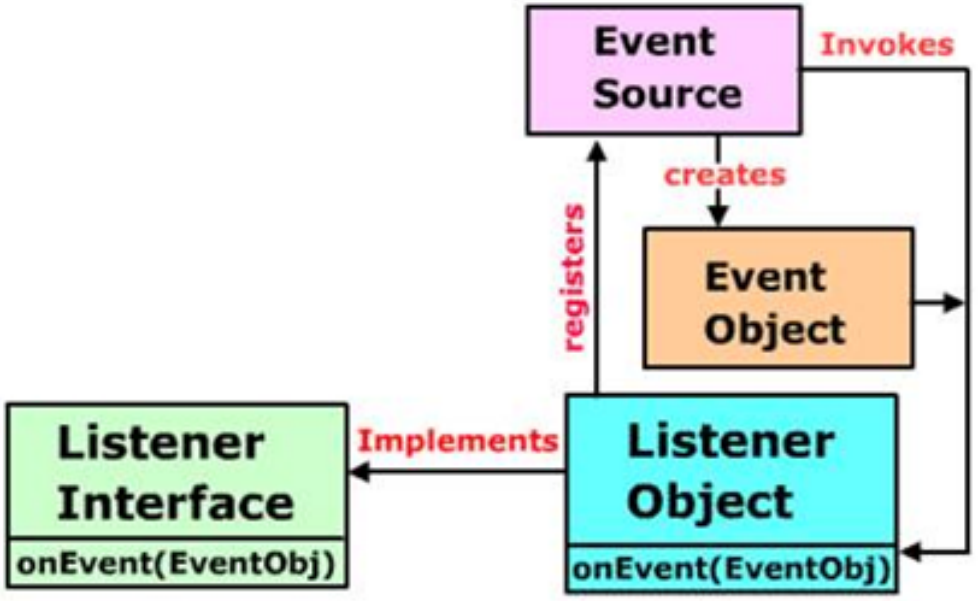
# The Delegation Event Model

- Java adopts the so-called "Event-Driven" (or "Event-Delegation") programming model for ***event-handling***, which defines standard and consistent mechanisms to generate and process events.
- Its concept is quite simple:
  - A ***source*** generates an event and sends it to one or more ***listeners***. In this scheme, the listener simply waits until it receives an event.
  - Once an event is received, the listener processes the event and then returns.
- The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.
- A user interface element is able to “delegate” the processing of an event to a separate piece of code.
- In the delegation event model, ***listeners*** must register with a ***source*** in order to receive an event notification (The benefit is notifications are sent only to listeners that want to receive them)

# The Delegation Event Model



# The Delegation Event Model



# Events

- In the delegation model, an ***event*** is an object that describes a state change in a ***source*** (changing the state of an object is known as an event).
- An event can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are:
  - Button press
  - Key press
  - Selecting an item in a list
  - Mouse click
- Events may also occur that are not directly caused by interactions with a user interface.
  - Timer expires
  - Counter exceed a value
  - Software/Hardware failure
  - Completion of an operation

# Event Sources

- A ***source*** is an object that generates an event. This occurs when the internal state of that object changes in some way.
- 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. Each type of event has its own registration method. Here is the general form:

**public void add*Type*Listener (*Type*Listener *el* )** ***-> multicasting****

- Here, ***Type*** is the name of the event, and ***el*** is a reference to the event listener. For example:
  - **addKeyListener( )** method registers a keyboard event listener.
  - **addMouseMotionListener( )** method registers a registers a mouse motion listener.

# Event Sources

- When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as ***multicasting*** the event.
- In all cases, notifications are sent only to listeners that register to receive them.
- 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**

***-> unicasting****

- Here, ***Type*** is the name of the event, and ***el*** is a reference to the event listener.
- When such an event occurs, the registered listener is notified. This is known as ***unicasting*** the event.

# Event Sources

- 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* )** ***-> unregister****

- Here, ***Type*** is the name of the event, and ***el*** is a reference to the event listener. For example:
  - **removeKeyListener( )** method is called to remove a keyboard listener.

- The methods that add or remove listeners are provided by the ***source*** that generates Events.

  For example, the **Component** class provides methods to add and remove keyboard and mouse event listeners.

# Event Listeners

- A ***listener*** is an object that is notified when an ***event*** occurs.
- It has two major requirements.
  1. It must have been registered with one or more sources to receive notifications about specific types of events.
  2. It must implement methods to receive and process these notifications.
- The methods that receive and process events are defined in a set of interfaces, such as those found in ***java.awt.event.***

# Event Classes

- The classes that represent events are at the core of Java's event handling mechanism.
- At the root of the Java event class hierarchy is ***EventObject***, which is in **java.util**. It is the superclass for all events. Its one constructor is shown here:

  **EventObject(Object *src*)**

- Here, ***src*** is the object that generates this event.
- ***EventObject*** defines two methods:
  1. **getSource( )** method returns the source of the event. Its general form is shown here:

     **Object getSource( )**

  2. **toString( )** returns the string equivalent of the event.

# Event Classes

- The class ***AWTEvent***, defined within the **java.awt** package, is a subclass of ***EventObject.***
- The class ***AWTEvent,*** is the superclass (either directly or indirectly) of all **AWT-based** events used by **the delegation event model**.
- The **getID( )** method can be used to determine the type of the event. The signature of this method is shown here:

  **int getID( )**

- To summarize:
  - ***EventObject*** is a superclass of all events.
  - ***AWTEvent*** is a superclass of all AWT events that are handled by the delegation event model.
- The package **java.awt.event** defines many types of events that are generated by various user interface elements.

# Event Classes

| 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. |

**Table 24-1** Commonly Used Event Classes in **java.awt.event**

# 1. The ActionEvent Class

- An ***ActionEvent*** is generated **when a button is pressed**, a list item is double-clicked, or a menu item is selected.
- The ***ActionEvent*** class defines four integer constants that can be used to identify any modifiers associated with an action event:
  - ALT_MASK
  - CTRL_MASK
  - META_MASK, and
  - SHIFT_MASK.
- In addition, there is an integer constant, ACTION_PERFORMED, which can be used to identify action events.

***-> button press****

# 1. The ActionEvent Class

- ***ActionEvent*** has these three constructors:

  **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*)**

- Here, ***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***.
- The argument modifiers indicates which modifier keys (alt, ctrl, meta, and/or shift) were pressed when the event was generated
- The ***when*** parameter specifies when the event occurred.

***-> button press****

# 1. The ActionEvent Class

- You can obtain the command name for the invoking ***ActionEvent*** object by using the **getActionCommand( )** method, shown here:

  **String getActionCommand( )**

- For example, when a button is pressed, an action event is generated that has a command name equal to the label on that button.

- The **getModifiers( )** method returns a value that indicates which modifier keys (alt, ctrl, meta, and/or shift) were pressed when the event was generated. Its form is shown here:

  **int getModifiers( )**

- The method **getWhen( )** returns the time at which the event took place. This is called the event's *timestamp*. The **getWhen( )** method is shown here:

  **long getWhen( )**

***-> button press****

# 2. The AdjustmentEvent Class

- An ***AdjustmentEvent*** is generated **by a scroll bar**. There are five types of adjustment events. The ***AdjustmentEvent*** class defines integer constants that can be used to identify them.

| | |
|---|---|
| BLOCK_DECREMENT | The user clicked inside the scroll bar to decrease its value. |
| BLOCK_INCREMENT | The user clicked inside the scroll bar to increase its value. |
| TRACK | The slider was dragged. |
| UNIT_DECREMENT | The button at the end of the scroll bar was clicked to decrease its value. |
| UNIT_INCREMENT | The button at the end of the scroll bar was clicked to increase its value. |

- In addition, there is an integer constant, ADJUSTMENT_VALUE_CHANGED, that indicates that a change has occurred.

***-> scroll bar****

# 2. The AdjustmentEvent Class

- ***AdjustmentEvent*** class has one constructor:

  **AdjustmentEvent(Adjustable *src*, int *id*, int *type*, int *val*)**

- Here, ***src*** is a reference to the object that generated this event.
- The ***id*** specifies the event.
- The type of the adjustment is specified by ***type,*** and Its associated value is ***val***.

- The **getAdjustable( )** method returns the object that generated the event. Its form is shown here:

  **Adjustable getAdjustable( )**

***-> scroll bar****

# 2. The AdjustmentEvent Class

- The type of the adjustment event may be obtained by the **getAdjustmentType( )** method. It returns one of the constants defined by ***AdjustmentEvent***. The general form is shown here:

  **AdjustmentEvent(Adjustable *src*, int *id*, int *type*, int *val*)**

- The amount of the adjustment can be obtained from the **getValue( )** method, shown here:

  **int getValue( )**

***-> scroll bar****

# 3. The ComponentEvent Class

- A ***ComponentEvent*** is generated **when the size, position, or visibility of a component is changed.**
- There are four types of component events. The ***ComponentEvent*** class defines integer constants that can be used to identify them.

| | |
|---|---|
| COMPONENT_HIDDEN | The component was hidden. |
| COMPONENT_MOVED | The component was moved. |
| COMPONENT_RESIZED | The component was resized. |
| COMPONENT_SHOWN | The component became visible. |

- ***ComponentEvent*** has one constructor:

  **ComponentEvent(Component *src*, int *type*)**

- Here, ***src*** is a reference to the object that generated this event. The type of the event is specified by ***type***.

***-> size, position, visibility****

# 3. The ComponentEvent Class

- ***ComponentEvent*** is the superclass either directly or indirectly of **ContainerEvent**, **FocusEvent**, **KeyEvent**, **MouseEvent**, and **WindowEvent**, among others.

- The **getComponent( )** method returns the component that generated the event. It is shown here:

  **Component getComponent( )**

***-> size, position, visibility****

# 4. The ContainerEvent Class

- A ***ContainerEvent*** is generated **when a component is added to or removed from a container.**
- There are two types of container events. The ***ContainerEvent*** class defines int constants that can be used to identify them:
  - COMPONENT_ADDED
  - COMPONENT_REMOVED

  indicate that a component has been **added to or removed** from the container.
- ***ContainerEvent*** is a subclass of ***ComponentEvent*** and has one constructor:

  **ContainerEvent(Component *src*, int *type*, Component *comp*)**
- Here, ***src*** is a reference to the container that generated this event.
- The type of the event is specified by ***type***, and the component that has been added to or removed from the container is ***comp***.

***-> added or removed****

# 4. The ContainerEvent Class

- You can obtain a reference to the container that generated this event by using the **getContainer( )** method, shown here:

  **Container getContainer( )**

- The **getChild( )** method returns a reference to the component that was added to or removed from the container. Its general form is shown here:

  **Component getChild( )**

***-> added or removed****

# 5. The FocusEvent Class

- A ***FocusEvent*** is generated **when a component gains or loses input focus**. These events are identified by the integer constants FOCUS_GAINED and FOCUS_LOST.

- A ***FocusEvent*** is a subclass of ***ComponentEvent*** and has these constructors:

  **FocusEvent(Component *src*, int *type*)**
  **FocusEvent(Component *src*, int *type*, boolean *temporaryFlag*)**
  **FocusEvent(Component src, int *type,* boolean *temporaryFlag*, Component *other*)**

- Here, ***src*** is a reference to the component that generated this event.
- The type of the event is specified by ***type.***
- The argument ***temporaryFlag*** is set to **true** if the focus event is temporary. Otherwise, it is set to **false**.
- The other component involved in the focus change, called the *opposite component*, is passed in ***other***. Therefore, if a FOCUS_GAINED event occurred, ***other*** will refer to the component that lost focus. Conversely, if a FOCUS_LOST event occurred, ***other*** will refer to the component that gains focus.

***-> Focus gain or lost****

# 5. The FocusEvent Class

- You can determine the other component by calling **getOppositeComponent( )**, shown here:

  **Component getOppositeComponent( )**

  The opposite component is returned.

- The **isTemporary( )** method indicates if this focus change is temporary. Its form is shown here:

  **boolean isTemporary( )**

  The method returns **true** if the change is temporary. Otherwise, it returns **false.**

***-> Focus gain or lost****

# 6. 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 | |

- Extended modifiers

| | | |
|---|---|---|
| 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 | |

***-> Focus gain or lost****

# 6. The InputEvent Class

- To test if a modifier was pressed at the time an event is generated, use the **isAltDown( )**, **isAltGraphDown( )**, **isControlDown( )**, **isMetaDown( )**, and **isShiftDown( )** methods. The forms of these methods are shown here:

  **boolean isAltDown( )**
  **boolean isAltGraphDown( )**
  **boolean isControlDown( )**
  **boolean isMetaDown( )**
  **boolean isShiftDown( )**

- You can obtain a value that contains all of the original modifier flags by calling the **getModifiers( )** method. It is shown here:

  **int getModifiers( )**

- You can obtain the extended modifiers by calling **getModifiersEx( )**, which is shown here:

  **int getModifiersEx( )**

***-> Focus gain or lost****

# 7. 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*)**

- Here, ***src*** is a reference to the component that generated this event.
- The type of the event is specified by ***type***.
- The specific item that generated the item event is passed in ***entry***.
- The current state of that item is in ***state***.

***-> check box or list item****

# 7. The ItemEvent Class

- The **getItem( )** method can be used to obtain a reference to the item that changed. Its signature is shown here:

  **Object getItem( )**

- The **getItemSelectable( )** method can be used to obtain a reference to the **ItemSelectable** object that generated an event. Its general form is shown here:

  **ItemSelectable getItemSelectable( )**

- Lists and choices are examples of user interface elements that implement the **ItemSelectable** interface.

  **ItemSelectable getItemSelectable( )**

- The **getStateChange( )** method returns the state change (that is, **SELECTED** or **DESELECTED**) for the event. It is shown here:

  **int getStateChange( )**

***-> check box or list item****

# 8. 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 other 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. Here are some others:

| | | | |
|---|---|---|---|
| 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 **VK** constants specify ***virtual key codes*** and are independent of any modifiers, such as control, shift, or alt.

***-> keyboard input****

# 8. The KeyEvent Class

- ***KeyEvent*** is a subclass of ***InputEvent***. Here is one of its constructors:

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

- Here, ***src*** is a reference to the component that generated this event.
- The type of the event is specified by ***type***.
- The system time at which the key was pressed is passed in ***when.***
- The ***modifiers*** argument indicates which modifiers were pressed when this key event occurred.
- The virtual key code, such as VK_UP, VK_A, and so forth, is passed in ***code***.
- The character equivalent (if one exists) is passed in ***ch***. If no valid character exists, then ***ch*** contains CHAR_UNDEFINED. For KEY_TYPED events, code will contain VK_UNDEFINED.

***-> keyboard input****

# 8. The KeyEvent Class

- The ***KeyEvent*** class defines several methods, but probably the most commonly used ones are **getKeyChar( )**, which returns the character that was entered, and **getKeyCode( )**, which returns the key code. Their general forms are shown here:

  **char getKeyChar( )**
  **int getKeyCode( )**

- If no valid character is available, then **getKeyChar( )** returns CHAR_UNDEFINED. When a KEY_TYPED event occurs, **getKeyCode( )** returns VK_UNDEFINED.

***-> keyboard input****

# 9. The MouseEvent Class

- There are eight types of mouse events. 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. |

***-> mouse event****

# 9. The MouseEvent Class

- ***MouseEvent*** is a subclass of ***InputEvent***. Here is one of its constructors:

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

- Here, ***src*** is a reference to the component that generated the event.
- The type of the event is specified by ***type***.
- The system time at which the mouse event occurred is passed in ***when***.
- The ***modifiers*** argument indicates which modifiers were pressed when a mouse event occurred.
- The coordinates of the mouse are passed in ***x*** and ***y***.
- The click count is passed in ***clicks.***
- The ***triggersPopup*** flag indicates if this event causes a pop-up menu to appear on this platform.

***-> mouse event****

# 9. The MouseEvent Class

- 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. Their forms are shown here:

  **int getX( )**
  **int getY( )**

- The **getPoint( )** method is used to obtain the coordinates of the mouse. It is shown here:

  **Point getPoint( )**

  It returns a Point object that contains the X,Y coordinates in its integer members: x and y.

- The **translatePoint( )** method changes the location of the event. It is shown here:

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

  Here, the arguments x and y are added to the coordinates of the event.

***-> mouse event****

# 9. The MouseEvent Class

- The **getClickCount( )** method obtains the number of mouse clicks for this event. Its signature is shown here:

  **int getClickCount( )**

- The **isPopupTrigger( )** method tests if this event causes a pop-up menu to appear on this platform. Its form is shown here:

  **boolean isPopupTrigger( )**

- The **getButton( )** method returns a value that represents the button that caused the event.

  **int getButton( )**

  For most cases, the return value will be one of these constants defined by ***MouseEvent***: NOBUTTON, BUTTON1, BUTTON2, and BUTTON3

- The NOBUTTON value indicates that no button was pressed or released.

***-> mouse event****

# 9. The MouseEvent Class

- The three methods that obtain the coordinates of the mouse relative to the screen rather than the component. They are shown here:

  **Point getLocationOnScreen( )**
  **int getXOnScreen( )**
  **int getYOnScreen( )**

- The **getLocationOnScreen( )** method returns a **Point** object that contains both the X and Y coordinate. The other two methods return the indicated coordinate.

***-> mouse event****

# 10. The TextEvent Class

- Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program.
- ***TextEvent*** defines the integer constant TEXT_VALUE_CHANGED.
- The one constructor for this class is shown here:

  **TextEvent(Object *src*, int *type*)**
- Here, ***src*** is a reference to the object that generated this event. The type of the event is specified by ***type***.

***-> text field/area****

# 11. The WindowEvent Class

- There are 10 types of window events, the ***WindowEvent*** class defines integer constants that can be used to identify them.
- The constants and their meanings are shown here:

| | |
|---|---|
| WINDOW_ACTIVATED | The window was activated. |
| WINDOW_CLOSED | The window has been closed. |
| WINDOW_CLOSING | The user requested that the window be closed. |
| WINDOW_DEACTIVATED | The window was deactivated. |
| WINDOW_DEICONIFIED | The window was deiconified. |
| WINDOW_GAINED_FOCUS | The window gained input focus. |
| WINDOW_ICONIFIED | The window was iconified. |
| WINDOW_LOST_FOCUS | The window lost input focus. |
| WINDOW_OPENED | The window was opened. |
| WINDOW_STATE_CHANGED | The state of the window changed. |

***-> window events****

# 11. The WindowEvent Class

- ***WindowEvent*** is a subclass of ***ComponentEvent***. It defines several constructors.

  **WindowEvent(Window *src*, int *type*)**
  **WindowEvent(Window *src*, int *type*, Window *other*)**
  **WindowEvent(Window *src*, int *type*, int *fromState*, int *toState*)**
  **WindowEvent(Window *src*, int *type*, Window *other*, int *fromState*, int *toState*)**

- Here, ***src*** is a reference to the object that generated this event. The type of the event is ***type***.
- Here, ***other*** specifies the opposite window when a focus or activation event occurs.
- The ***fromState*** specifies the prior state of the window, and ***toState*** specifies the new state that the window will have when a window state change occurs.

- A commonly used method in this class is ***getWindow( )***. It returns the ***Window*** object that generated the event. Its general form is shown here:

  **Window getWindow( )**

***-> text field/area****

# 11. The WindowEvent Class

- WindowEvent also defines methods that return the opposite window (when a focus or activation event has occurred), the previous window state, and the current window state. These methods are shown here:

**Window getOppositeWindow( )**
**int getOldState( )**
**int getNewState( )**

***-> text field/area****

# Sources of Events

- In addition to the graphical user interface elements, any class derived from ***Component***, such as ***Applet***, can generate events.

| 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. |

**Table 24-2** Event Source Examples

# Event Listener Interfaces

- The **delegation event model** has two parts: **sources** and **listeners.**

- ***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 Listener Interfaces

| Interface | 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. |
| 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. |

**Table 24-3** Commonly Used Event Listener Interfaces

# The specific methods in each interface

- **The ActionListener Interface**
  - This interface defines the **actionPerformed( )** method that is invoked when an action event occurs. Its general form is shown here:

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

- **The AdjustmentListener Interface**
  - This interface defines the **adjustmentValueChanged( )** method that is invoked when an adjustment event occurs. Its general form is shown here:

    **void adjustmentValueChanged(AdjustmentEvent *ae*)**

- **The ItemListener Interface**
  - This interface defines the **itemStateChanged( )** method that is invoked when the state of an item changes. Its general form is shown here:

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

# The specific methods in each interface

- **The ComponentListener Interface**
  - This interface defines four methods that are invoked when a component is resized, moved, shown, or hidden. Their general forms are shown here:

    **void componentResized(ComponentEvent *ce*)**
    **void componentMoved(ComponentEvent *ce*)**
    **void componentShown(ComponentEvent *ce*)**
    **void componentHidden(ComponentEvent *ce)***

- **The ContainerListener Interface**
  - This 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. Their general forms are shown here:

    **void componentAdded(ContainerEvent *ce*)**
    **void componentRemoved(ContainerEvent *ce*)**

# The specific methods in each interface

- **The FocusListener Interface**
  - This interface defines two methods. When a component obtains keyboard focus, **focusGained( )** is invoked. When a component loses keyboard focus, **focusLost( )** is called. Their general forms are shown here:

    **void focusGained(FocusEvent *fe*)**
    **void focusLost(FocusEvent *fe*)**

- **The KeyListener Interface**
  - This interface defines three methods. The **keyPressed( )** and **keyReleased( )** methods are invoked when a key is pressed and released, respectively. The **keyTyped( )** method is invoked when a character has been entered. The general forms of these methods are shown here:

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

# The specific methods in each interface

- **The MouseListener Interface**
  - This interface defines five methods. (1) If the mouse is pressed and released at the same point, **mouseClicked( )** is invoked. (2) When the mouse enters a component, the **mouseEntered( )** method is called. (3) When it leaves, **mouseExited( )** is called. (4 & 5) The **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*)**

# The specific methods in each interface

- **The MouseMotionListener Interface**
  - This interface defines two methods. The **mouseDragged( )** method is called multiple times as the mouse is dragged. The **mouseMoved( )** method is called multiple times as the mouse is moved. Their general forms are shown here:

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

- **The TextListener Interface**
  - This interface defines the **textValueChanged( )** method that is invoked when a change occurs in a text area or text field. Its general form is shown here:

    **void textValueChanged(TextEvent *te*)**

# The specific methods in each interface

- **The WindowFocusListener Interface**
  - This interface defines two methods: **windowGainedFocus( )** and **windowLostFocus( ).** These are called when a window gains or loses input focus. Their general forms are shown here:

    **void windowGainedFocus(WindowEvent *we*)**
    **void windowLostFocus(WindowEvent *we*)**

- **The WindowListener Interface**
  - This interface defines seven methods. The **windowActivated( )** and **windowDeactivated( )** methods are invoked when a window is activated or deactivated, respectively. If a window is iconified, the **windowIconified( )** method is called. When a window is deiconified, the **windowDeiconified( )** method is called. When a window is opened or closed, the **windowOpened( )** or **windowClosed( )** methods are called, respectively. The **windowClosing( )** method is called when a window is being closed. The general forms of these methods are:

(go to next slide)

# The specific methods in each interface

**void windowActivated(WindowEvent *we*)**

**void windowClosed(WindowEvent *we*)**

**void windowClosing(WindowEvent *we*)**

**void windowDeactivated(WindowEvent *we*)**

**void windowDeiconified(WindowEvent *we*)**

**void windowIconified(WindowEvent *we*)**

**void windowOpened(WindowEvent *we*)**

# Using the Delegation Event Model

- **Using the delegation event model is actually quite easy. Just follow these two steps:**

  1. Implement the appropriate interface in the listener so that it can receive the type of event desired.

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

- 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.

# Event Handling Few Sample Questions

1. Write different constants and their description available in AdjustmentEvent class.
2. Write different constants and their description available in ComponentEvent class.
3. Explain syntax of all constructors available in ContainerEvent and FocusEvent class.
4. Write the name of constants present in InputEvent class.
5. List all the constants present in KeyEvent class.
6. List all the constants present in MouseEvent class.
7. List all the constants and their meaning present in WindowEvent class.

# Creating GUI applications using AWT

- **Kindly attend following 7 lectures on the youtube for AWT**

1. https://www.youtube.com/watch?v=PDzltlkenhw&list=PLPHs5nuslqFQu2TVWHzeIoU-tf_FfPDAf&index=1
2. https://www.youtube.com/watch?v=eBhf79VAYfs&list=PLPHs5nuslqFQu2TVWHzeIoU-tf_FfPDAf&index=2
3. https://www.youtube.com/watch?v=tAbaDO0fgQs&list=PLPHs5nuslqFQu2TVWHzeIoU-tf_FfPDAf&index=3
4. https://www.youtube.com/watch?v=3kjUqmmSyzo&list=PLPHs5nuslqFQu2TVWHzeIoU-tf_FfPDAf&index=4
5. https://www.youtube.com/watch?v=SZ0qwp6FRLA&list=PLPHs5nuslqFQu2TVWHzeIoU-tf_FfPDAf&index=5
6. https://www.youtube.com/watch?v=OSRJ4rPDA5g&list=PLPHs5nuslqFQu2TVWHzeIoU-tf_FfPDAf&index=6
7. https://www.youtube.com/watch?v=q0M9qBqc3YU&list=PLPHs5nuslqFQu2TVWHzeIoU-tf_FfPDAf&index=7

# References

## Reference for this topic

- [Book: Java: The Complete Reference, Ninth Edition: Herbert Schildt ]
  https://www.amazon.in/Java-Complete-Reference-Herbert-Schildt/dp/0071808558

- [Web: GeeksforGeeks ]
  https://www.geeksforgeeks.org/java/

- [Web: Java T Point tutorial ]
  https://www.javatpoint.com/java-tutorial