- Events are supported by the **java.awt.event** package.
- The modern approach to handling events is based on the *delegation event model*, which defines standard and consistent mechanisms to generate and process events.
- A *source* generates an event and sends it to one or more *listeners*.
- An *event* is an object that describes a state change in a source.
- Events may also occur that are not directly caused by interactions with a user interface.
- A *source* is an object that generates an event. This 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. Each type of event has its own registration method.
  the general form:
  public void add*Type*Listener(*Type*Listener *el*)
- 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

  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*)
- A *listener* is an object that is notified when an event occurs. It has two major requirements.
- First, it must have been registered with one or more sources to receive notifications about specific types of events.
  Second, it must implement methods to receive and process these notifications.
- The methods that receive and process events are defined in a set of interfaces found in **java.awt.event**.
- 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*)**
- **EventObject** contains two methods: **getSource( )** and **toString( )**.
- The **getSource( )** method returns the source of the event. **Object getSource( )**
- **toString( )** returns the string equivalent of the event.
- The class **AWTEvent**, defined within the **java.awt** package, is a subclass of **EventObject**. It is the superclass (either directly or indirectly) of all AWT-based events used by the delegation event model.

- **getID( )** method can be used to determine the type of the event.
  int getID( )

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

- 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 super class 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.

- 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.

- 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*)

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

- The **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( )

- Timestamps were added by **ActionEvent** to help support the improved input focus subsystem
- An **AdjustmentEvent** is generated by a scroll bar.
- 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.

- **AdjustmentEvent** constructor:
  AdjustmentEvent(Adjustable *src*, int *id*, int *type*, int *data*)

- The **getAdjustable( )** method returns the object that generated the event. **Adjustable getAdjustable( )**

- The type of the adjustment event may be obtained by the **getAdjustmentType( )** method.
- It returns one of the constants defined by **AdjustmentEvent**.
  int getAdjustmentType( )

- The amount of the adjustment can be obtained from the **getValue( )** method **int getValue( )**

- 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.

- Constructors:
  ComponentEvent(Component *src*, int *type*)

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

- The **getComponent( )** method returns the component that generated the event. **Component getComponent( )**

- 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** and **COMPONENT_REMOVED**.

- **ContainerEvent** is a subclass of **ComponentEvent**

- constructor:
  ContainerEvent(Component *src*, int *type*, Component *comp*)

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

- The **getChild( )** method returns a reference to the component that was added to or removed from the container
  Component getChild( )

- 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**.

- **FocusEvent** is a subclass of **ComponentEvent**

- constructors:
  FocusEvent(Component *src*, int *type*)
  FocusEvent(Component *src*, int *type*, boolean *temporaryFlag*)
  Focus Event(Component *src*, int *type*, boolean *temporaryFlag*, Component *other*)

- The argument *temporaryFlag* is set to **true** if the focus event is temporary.
- 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.

- You can determine the other component by calling **getOppositeComponent( )**, **Component getOppositeComponent( )**

- The **isTemporary( )** method indicates if this focus change is temporary **boolean isTemporary( )**

- 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.

- **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

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

- **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.
  int getModifiers( )

- You can obtain the extended modifiers by called **getModifiersEx( )**, **int getModifiersEx( )**
- 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

- DESELECTED :The user deselected an item.
  SELECTED: The user selected an item.

- **ItemEvent** defines one integer constant, **ITEM_STATE_CHANGED**, that signifies a change of state.

- constructor:
  ItemEvent(ItemSelectable *src*, int *type*, Object *entry*, int *state*)

- The **getItem( )** method can be used to obtain a reference to the item that generated an event.
  Object getItem( )

- The **getItemSelectable( )** method can be used to obtain a reference to the **ItemSelectable** object that generated an event.
  ItemSelectable getItemSelectable( )

- Lists and choices are examples of user interface elements that implement the **ItemSelectable** interface.
- The **getStateChange( )** method returns the state change (i.e., **SELECTED** or **DESELECTED**) for the event.

int getStateChange( )

- 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**.
- The first two events are generated when any key is pressed or released. The last event occurs only when a character is generated.

- 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_ENTER**
- **VK_ESCAPE**
- **VK_CANCEL**
- **VK_UP**
- **VK_DOWN**
- **VK_LEFT**

- **VK_RIGHT**
- **VK_PAGE_DOWN**
- **VK_PAGE_UP**
- **VK_SHIFT**
- **VK_ALT**
- **VK_CONTROL**

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

- **KeyEvent** is a subclass of **InputEvent**.

- 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*)**

- **getKeyChar( )**, which returns the character that was entered, and **getKeyCode( )**, which returns the key code.
- **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**.

- 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**

- **MouseEvent** is a subclass of **InputEvent**.
- constructors.
- **MouseEvent(Component *src*, int *type*, long *when*, int *modifiers*, int *x*, int *y*, int *clicks*, boolean *triggersPopup*)**

- The most commonly used methods in this class are **getX( )** and **getY( )**. These return the X and Y coordinates of the mouse when the event occurred.
- **int getX( )**
- **int getY( )**

- **getPoint( )** method to obtain the coordinates of the mouse. **Point getPoint( )**

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

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

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

- **isPopupTrigger( )** method tests if this event causes a pop-up menu to appear on this platform.
  boolean isPopupTrigger( )

- int getButton( ) : It returns a value that represents the button that caused the event. The return value will be one of these constants defined by **MouseEvent**.
  NOBUTTON BUTTON1 BUTTON2 BUTTON3

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

The **MouseWheelEvent** class encapsulates a mouse wheel event. It is a subclass of **MouseEvent**

If a mouse has a wheel, it is located between the left and right buttons. Mouse wheels are used for scrolling. **MouseWheelEvent** defines these two integer constants.

- **WHEEL_BLOCK_SCROLL A page-up or page-down scroll event occurred.**
- **WHEEL_UNIT_SCROLL A line-up or line-down scroll event occurred.**

- constructor.
- **MouseWheelEvent(Component *src*, int *type*, long *when*, int *modifiers*, int *x*, int *y*, int *clicks*, boolean *triggersPopup*, int *scrollHow*, int *amount*, int *count*)**

- To obtain the number of rotational units, call **getWheelRotation( ), int getWheelRotation( )**

- If the value is positive, the wheel moved counterclockwise. If the value is negative, the wheel moved clockwise.

- To obtain the type of scroll, call **getScrollType( ) int getScrollType( )**

- If the scroll type is **WHEEL_UNIT_SCROLL**, you can obtain the number of units to scroll by calling **getScrollAmount( )**.
  int getScrollAmount( )

- Text Event 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**.

- Constructor:
  TextEvent(Object *src*, int *type*)

- The **TextEvent** object does not include the characters currently in the text component that generated the event.

- There are ten types of window events. The **WindowEvent** class defines integer constants that can be used to identify them.

- **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.**

- **WindowEvent** is a subclass of **ComponentEvent**

- **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*)

- **getWindow( )**. It returns the **Window** object that generated the event.
- **Window getWindow( )**

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

- **Button :Generates action events when the button is pressed.**
- **Checkbox: 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.**
- **Scrollbar :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.**

- **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.**

- ActionListener interface defines the **actionPerformed( )** method that is invoked when an action event occurs.
- **void actionPerformed(ActionEvent *ae*)**

- AdjustmentListener interface defines the **adjustmentValueChanged( )** method that is invoked when an adjustment event occurs.
- **void adjustmentValueChanged(AdjustmentEvent *ae*)**

- ComponentListener interface defines four methods that are invoked when a component is resized, moved, shown, or hidden.
- **void componentResized(ComponentEvent *ce*)**
- **void componentMoved(ComponentEvent *ce*)**
- **void componentShown(ComponentEvent *ce*)**
- **void componentHidden(ComponentEvent *ce*)**

- 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 defines two methods. When a component obtains keyboard focus, • **focusGained( )** is invoked. When a component loses keyboard focus, **focusLost( )** is called.
- **void focusGained(FocusEvent *fe*)**
- **void focusLost(FocusEvent *fe*)**

- ItemListener interface defines the **itemStateChanged( )** method that is invoked when the state of an item changes.
- **void itemStateChanged(ItemEvent *ie*)**

- KeyListener 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.

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

- 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. The **mousePressed( )** and **mouseReleased( )** methods are invoked when the mouse is pressed and released,
- **void mouseClicked(MouseEvent *me*)**
- **void mouseEntered(MouseEvent *me*)**
- **void mouseExited(MouseEvent *me*)**
- **void mousePressed(MouseEvent *me*)**
- **void mouseReleased(MouseEvent *me*)**

- MouseMotionListener 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.

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

- MouseWheelListener interface defines the **mouseWheelMoved( )** method that is invoked when the mouse wheel is moved.
- **void mouseWheelMoved(MouseWheelEvent *mwe*)**

- 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 defines two methods: **windowGainedFocus( )** and **windowLostFocus( )**. These are called when a window gains or losses input focus.

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

WindowListener 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.

- **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*)**

- *adapter class*, that can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface. You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.

- Adapter Class: Listener Interface
- ComponentAdapter: ComponentListener
- ContainerAdapter :ContainerListener
- FocusAdapter: FocusListener
- KeyAdapter :KeyListener
- MouseAdapter: MouseListener
- MouseMotionAdapter: MouseMotionListener
- WindowAdapter :WindowListener

- An *anonymous* inner class is one that is not assigned a name

- an *inner class* is a class defined within other class, or even within an expression.