AWT programming

Unit VI

Why Does a Program or Application Need to be Event Driven?

- Before event handling came into the picture, a program had to collect all the user information itself to know what it was doing at a given point in time.
- This means that after being run or initialized, a program was always in a big repeat loop that was waiting for the user to do something.
- The Abstract Window Toolkit or AWT has gone ahead and struck association with a different working model for solving the issue discussed above.
- This new model is event-driven programming.
- With AWT, there is no need for the program to look out for user-generated events.
- It is the Java run time that does this job.
- It intimates the program as soon as an event takes place.
- It saves a valuable resource from exhaustion and handles user interaction better.

What is an Event?

- Change in the state of an object is known as event i.e. event describes the change in state of source.
- Events are generated as result of user interaction with the graphical user interface components.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

What is Event Handling?

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.
- This mechanism have the code which is known as event handler that is executed when an event occurs.
- Java Uses the Delegation Event Model to handle the events.
- This model defines the standard mechanism to generate and handle the events.

- **Event source**:
- In Java, nearly everything is an object.
- The button you press is an object too.
- Source is the object which generates an event.
- In other words, a source is an object which undergoes state change.
- An event source that generates an event is mostly an AWT component.

- **Event classes:** In Java, these are classes that account for almost every component that has anything to do with the generation of events.
- These are also called event types. Here are a few of the most common event classes:
  - ***ActionEvent***: This event class or event type represents an event that involves the clicking of a graphical element, such as a button or a list item. The listener related to this class is ActionListener.
  - ***KeyEvent***: This event class represents an event that involves the pressing and releasing of a key. The listener associated with this class is KeyListener.

- ***ContainerEvent***: This event type represents an event that happens with the GUI container. This class is associated with any event where user action involves the addition or removal of object(s) from the GUI. The related listener for this class is ContainerListener.
- ***MouseEvent***: This class represents all those events that involve the clicking or pressing of the mouse. The listener for this class is MouseListener.
- ***WindowEvent***: This event class or type represents events that involve any action related to a window. Closing, activating, or deactivating a window come under this class. The related listener for this class is WindowListener

- **Listeners**
- Listeners are also called as event handlers as they are the ones responsible to handle events occurring at the source.
- Listeners are interfaces and different types of listeners are used according to the event
- The job of event listeners involves listening for events and then processing them appropriately when they take place
- Almost every component in Java has a dedicated listener that handles any event that that component generates

Important Event Classes and Interface

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener

Important Event Classes and Interface

Event Classes	Description	Listener Interface
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

Steps involved in event handling

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- the method is now get executed and returns.

Registration Methods

- For registering the component with the Listener, many classes provide the registration methods. For example:
- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java ActionListener Interface

- The Java ActionListener is notified whenever you click on the button or menu item.
- It is notified against ActionEvent.
- The ActionListener interface is found in java.awt.event package.
- It has only one method: actionPerformed().
- The actionPerformed() method is invoked automatically whenever you click on the registered component.

public abstract void actionPerformed(ActionEvent e);

How to write ActionListener

1. Implement the ActionListener interface in the class:

   public class ActionListenerExample Implements ActionListener

2. Register the component with the Listener:

   component.addActionListener(instanceOfListenerclass);

3. Override the actionPerformed() method:

   ```
   public void actionPerformed(ActionEvent e){
           //Write the code here
   }
   ```

Java MouseListener Interface

- The Java MouseListener is notified whenever you change the state of mouse.
- It is notified against MouseEvent.
- The MouseListener interface is found in java.awt.event package.
- It has five methods.

1. *public abstract void mouseClicked(MouseEvent e);*
2. *public abstract void mouseEntered(MouseEvent e);*
3. *public abstract void mouseExited(MouseEvent e);*
4. *public abstract void mousePressed(MouseEvent e);*
5. *public abstract void mouseReleased(MouseEvent e);*

Java ItemListener Interface

- The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent.
- The ItemListener interface is found in java.awt.event package.
- It has only one method: itemStateChanged().

public abstract void itemStateChanged(ItemEvent e);

Java KeyListener Interface

- The Java KeyListener is notified whenever you change the state of key.
- It is notified against KeyEvent.
- The KeyListener interface is found in java.awt.event package.
- It has three methods.

1. *public abstract void keyPressed(KeyEvent e);*
2. *public abstract void keyReleased(KeyEvent e);*
3. *public abstract void keyTyped(KeyEvent e);*

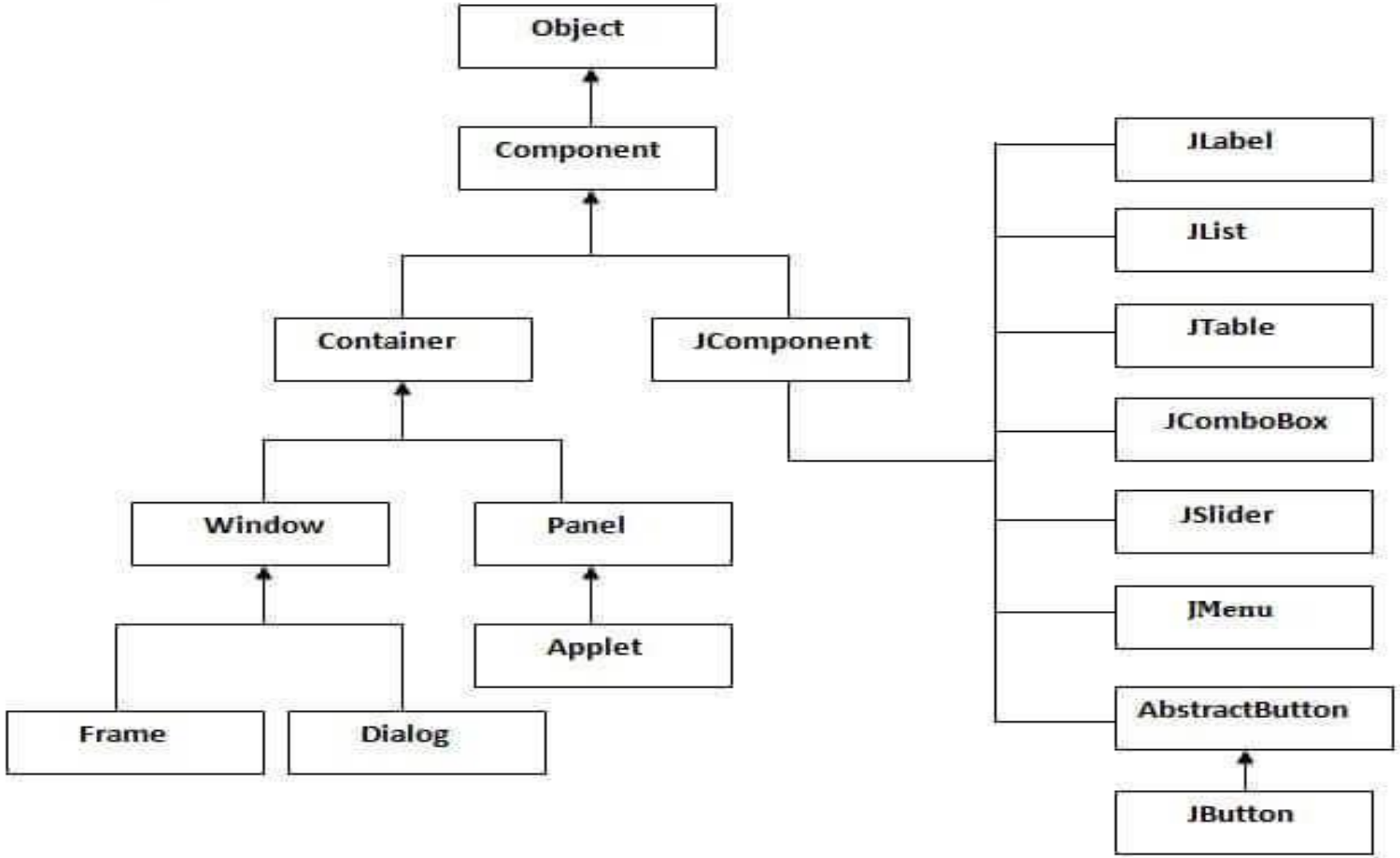
Java Swing

- It is *used to create window-based applications*.
- It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Hierarchy of Java Swing classes



Java JButton

- The JButton class is used to create a labeled button that has platform independent implementation

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Java JLabel

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Java JTextField

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Java JTextArea

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Java JPasswordField

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.
JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int	Construct a new JPasswordField initialized with the specified text and

Java JRadioButton

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

Java JComboBox

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Java JTable

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Java JOptionPane

Constructor	Description
JOptionPane()	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

Methods of JOptionPane class

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with given title and messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

Java JDialog

- The JDialog control represents a top level window with a border and a title used to take some form of input from the

Constructor	Description
JDialog()	It is used to create a modeless dialog without a title and without a specified Frame owner.
JDialog(Frame owner)	It is used to create a modeless dialog with specified Frame as its owner and an empty title.
JDialog(Frame owner, String title, boolean modal)	It is used to create a dialog with the specified title, owner Frame and modality.

Java JMenuBar, JMenu and JMenuItem

- JMenuBar class declaration
- **public class** JMenuBar **extends** JComponent **implements** MenuElement, Accessible
- JMenu class declaration
- **public class** JMenu **extends** JMenuItem **implements** Menu Element, Accessible
- JMenuItem class declaration
- **public class** JMenuItem **extends** AbstractButton **implemen ts** Accessible, MenuElement