UNIT-3 [Around 5 marks]

Event Handling

this full page imp

⊛ Event Handling Concept: [Imp]

Why we need event handling, how event handling works or describe delegation event model?

An event can be defined as changing the state of an object or behaviour by performing actions. Actions can be a button click, cursor movement, mouse clicks, keypress through keyboard, page scrolling etc. Events are of two types:

Foreground Events: Events that require user interaction to generate are foreground events. Example: clicking a button, cursor movements etc.

Background Events: Events that don't require user interaction to generate are background events. Example: operating system failures/interrupts, operation completion etc.

Event Handling: It is a mechanism to control the events and to decide what should happen after an event occur. To handle the events, java follows Delegation Event Model. It has Sources and Listeners:

Source: Events are generated from the source. There are various sources like buttons, checkboxes, list, menu-item etc. to generate events.

Listeners: Listeners are used for handling the events generated from the source. Each of these listeners represents interfaces that are responsible for handling events.

To perform Event Handling, we need to register the Source with the Listener.

Syntax: addTypeListener()

where type represents the type of event.

For Example: For KeyEvent we use addKeyListener() to register, Similarly For ActionEvent we use addActionListener() to register.

⊛ Listener Interfaces:

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
FocusEvent	FocusListener

Registeration Methods:

i) For Button or MenuItem:
   ⇒ public void addActionListener(ActionListener a) { }

ii) For Checkbox or Choice:
   ⇒ public void addItemListener (ItemListener a) { }

iii) For TextArea:
   ⇒ public void addTextListener (TextListener a) { }

iv) For TextField:
   ⇒ public void addActionListener (ActionListener a) { }
   ⇒ public void addTextListener (TextListener a) { }

v) For List:
   ⇒ public void addActionListener (ActionListener a) { }
   ⇒ public void addItemListener (ItemListener a) { }

⊛ Adapter Classes: [Imp] ← At least concept of Adapter class theory

Java adapter classes provide the default implementation of listener interfaces. If we inherit the adapter class, we will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

Advantages of using Adapter classes:

→ It provides ways to use classes in different ways.

→ It increases transparency of classes.
→ It provides a way to include related patterns in the class.
→ It increases the reusability of the class.

Adapter class	Listener Interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
FocusAdapter	FocusListener
MouseMotionAdapter	MouseMotionListener

Java MouseAdapter Example:

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter{
    Frame f;
    MouseAdapterExample(){
        f=new Frame("Mouse Adapter");
        f.addMouseListener(this);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
public void mouseClicked(MouseEvent e){
        Graphics g=f.getGraphics();
        g.setColor(Color.BLUE);
        g.fillOval(e.getX(), e.getY(), 30,30);
}
public static void main(String args[]){
    new MouseAdapterExample();
}
}
```

⊛ Handling Action Events: [Imp]

For Action, Key, Mouse सबै events को example लगभग similar हुन्छ / फरक mainly tick(✓) लगाको line 3-4 वटा मात्र फरक हुन्छ Event को type अनुसार

```
import java.awt.*;
import java.awt.event.*;
✓ class AEvent extends Frame implements ActionListener{
    TextField tf;
    AEvent(){
        //create components
        tf=new TextField();
        tf.setBounds(60,50,170,20);
        Button b=new Button("click me");
        b.setBounds(100,120,80,30);

        //register listener
✓       b.addActionListener(this); //passing current instance

        //add components & set size, layout and visibility
        add(tf); add(b);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
✓ public void actionPerformed(ActionEvent e){
        tf.setText("Welcome");
    }
    public static void main(String args[]){
        new AEvent();
    }
}
```

⊛ Handling Key Events: [Imp]

माथिको भन्दा different हुने कुरा मात्र लेखिएको ६ वटा लाइनहरू को लागि

```
//imports same as action events above.
class KeyListenerExample extends Frame implements KeyListener{
    TextField tf; Label l;
    KeyListenerExample(){
        //create components same as above
        replacing ~~eliminating~~ button by label
        //register listener
        textArea.addKeyListener(this);
        //add components & set size, layout and visibility
        same as above.
    }
    public void keyPressed(KeyEvent e){
        l.setText("Key Pressed");
    }
//Finally call KeyListenerExample in main function as we did above.
```

Similarly we can replace for key released & key typed also this ~~key~~ as our need

⊛Handling Mouse Events: [Imp]

```
//imports same as action events
class MouseListenerExample extends Frame implements MouseListener{
        Label l;
        MouseListenerExample(){
                //create component label
                //add, set size, layout and visibility
                //register listener
                addMouseListener(this);
        }
        public void mouseClicked(MouseEvent e){
                l.setText("Mouse Clicked");
        }
//finally call MouseListenerExample from main function.
```

Similarly we can do for mouseEntered, mouseExited, mousePressed, mouseReleased

⊛Handling Window Event:

According to past questions window, focus, and item events are lesser important

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
class WindowExample extends Frame implements WindowListener{
        WindowExample(){
                //set size, layout, and visibility as in action.
                addWindowListener(this);
        }
        //call WindowExample from main method.
//overriding
public void windowActived(WindowEvent e){
        System.out.println("actived");
}
```

Similarly we can do for windowClosed, windowClosing, windowDeiconified, windowIconified, windowOpened, as per our need

⊛. Handling Item Event and Handling Focus are skipped, less important and lengthy, little bit harder examples.

⊛ Event Listener vs. Adapter class: [Imp]

Many of the listener interfaces have more than one method. When we implement a listener interface in any class then we must have to implement all the methods declared in that interface because all the methods in an interface are final and must be override in class which implement it.

Adapter class makes it easy to deal with this situation. An adapter class provides empty implementations of all methods defined by that interface. Adapter classes are very useful if we want to override only some of the methods defined by that interface.