Unit-5

Event Handling

- event, event listener, event handler, source.
- event delegation model.
- Adapter class ★★
- Anony mous Inner Class

Q) what r events?

A
- when a user interacts w/ GUI program, events r generated.
- pgm responds to these events.

Q) What is event delegation model for event handling? ★★★★★ v. v. imp.

A
- Event handling means when any event is generated, how does the program responds to it.
- eg. of events, mouse button pressed
  " " released
  key pressed etc.

event delegation model :-

- There r source & listeners
- Src gens events & send it to 1 or more listeners
- listeners wait for events, when an event is received listener processes the event & returns.
- Here source is the UI & it delegates the processing of event to separate piece of code.
- Listener must register w/ a src in order to receive an event notification
- So notification sent only to listeners that want to recv them.

Adv. of event delegation model over the earlier inheritance model :- **

- Earlier event was propagated up the containment hierarchy until it was received by a component.
- This required component to receive the events that it did not process & it wasted valuable time.
- In ev. delegation - application logic that processes the event is separated from UI logic that generates the event.

- UI delegates the processing of event to separate piece of code.

Q) What is an event?
- an obj that describes state change in a src.
- Suppose the mouse moved, then the obj will contain the position of mouse.

Q) What is a src?
- obj that generate event.
- Src can generate several events.
- for eg A window can be src.
  A button " " "
- Src registers listener to receive notification.

Q) What is ev. listener?
- an obj that is notified when event occurs.
- ev. listener waits for event to occur.
- A listener must be registered w/ one or more src
- It must implement methods to receive & process events (it must b/c event handler(s)).

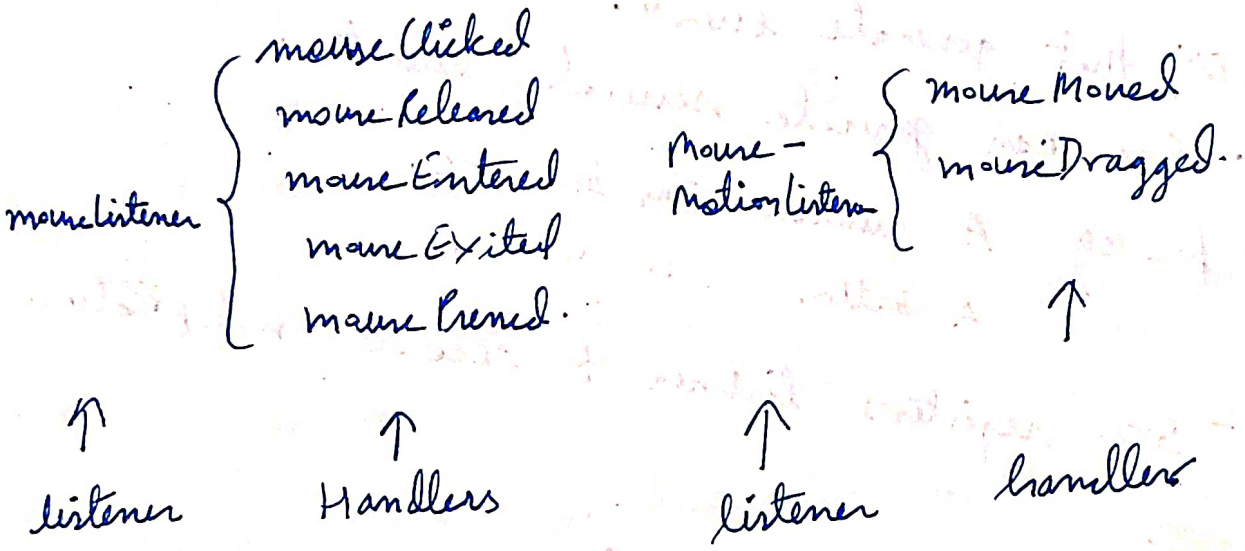
Q WAP to demo event handling

Handle mouse events like keypressed, keyreleased,

- When any mouse event occurs, it should be displayed on top left corner of a GUI window

A listeners for mouse, & their handlers.

- there r 2 mouselisteners - MouseListener & MouseMotionListener w/ the following handlers.



Structure of pgm :-

```
demo class {
    s: msg → to be printed        } var.
       x, y → coordinate of mouse
    ----------
    demo() { // constructor
        listenerss - addMouseListener      } listener
        listenerr  - addMouseMotionListener.
    }
    paint( ){ // paint fun.
        g.drawString(...);
    }
    main() {
        appl obj;
        appl.setSize()
        appl.setTitle()
        appl.setVisible(true);
    }
}
-----------------------------------
handler class{
    demo obj;
    handler(po){  // Constructor
        obj = po
    }
    handler func → mousePressed()    } //handlers
    //      fun → mouseReleased()
    //      fun → mouseMoved()
}
```

Code :-

```
import java.awt.*;
"     "  . "  . event.*;

class demo extends Frame{
    String msg = "";  // msg to be printed
    int x, y;         // Coordinate of Mouse.

    demo(){
    //create listeners
    addMouseListener(new myhandler(this));
                          ↑
                     handler class
    addMouseMotionListener(new myhandle(this));
    addWindowListener(new mywindow(this));
    }
    public void paint(Graphics g){
        g.drawString(msg, x, y);

    }

    public s v main(--){
        demo aff = new demo();
        aff.setSize(new Dimension(300, 300));
        aff.setTitle("demo");
        aff.setVisible(true);
    }
}
```

class myhandler implements MouseListener, MouseMotionListener {

demo obj;

myhandler(demo po) {
obj = po;
}

public void mouseClicked(MouseEvent me) {
// event handler for m clicked
obj.x = 10; // determine coordinate & msg.
obj.y = 100;
obj.msg = "mouse clicked";
obj.repaint(); // call repaint.
}

p v mousePressed(MouseEvent me) {
obj.x = 10; // handler-2
obj.y = 100;
obj.msg = "mouse pressed"; } // determine coordinate & msg
obj repaint(); // repaint().
}

p v mouseReleased(MouseEvent me) {
obj.x = 10;
obj.y = 100;
obj.msg = "mouse Released";
obj.repaint();
}

}

```
p v mouseEntered(MouseEvt me){
//handler for event when mouse enters the window
        obj.x=10;
        obj.y=100;
        msg = " m. entered";
        obj.repaint();

}
p v mouseExited(---){
        obj.x=10;
        obj.y=100;
        msg = "----";
        obj.repaint();
}
}
```

```
//This class is to handle window events like
  pressing close button.
class mywindow extends WindowAdapter{
    p v windowClosing(WindowEvent we){
//handler to close win when X button is
  pressed
        System.exit(0);
    }
}
```

Adapter Class :- ***

- In the previous eg- there r 2 event listeners
  MouseListener & MouseMotionListener
- The handler class implements these listeners so the handler class must implements all the fun in these listeners.

- MouseListener
  ↓
  mouseClicked
  " Pressed
  " Released
  " Entered
  " Exited

  MouseMotionListener
  ↓
  mouseMoved
  " Dragged.

- The handler must define all these fun, even if we do not need them.
- This takes extra effort.
- To reduce this effort Adapter Classes r used.
- Adapter Class p/v empty implementation of these fun.
  So, when the handler class extends the adapter class, the handler class need not define all the fun.

- Only those fun that r required can be defined.
- Name of adapter classes :-

MouseMotionListener	MouseListener
↓	↓
MouseMotionAdapter (class)	MouseAdapter (class)

- Now 2 separate handler class can be created & only one fun can be defined in each handler class.

Code - See - 34c

Anonymous Inner Class

- To make the code even shorter, anonymous inner class can be used.
- We need not create separate handler classes.
- When creating the listener, the anonymous inner class (for handler) is created & the fun is defined inside the class.
- eg Normally.

```
addMouseListener(new myhandler(this));
                 ^^^^^^^^^^^^^^^^^^^
                 handler obj
- - - - - - - -
class ho myhandler {
        =
        =
        =
}
- - - - - - - - - -
```

- With anonymous inner class.

```
addMouseListener(new MouseAdapter(){
    public void mouseClicked(MouseEvent me){
        x = 10;
        y = 100;
        msg = "m clicked";
    }//handler.
    repaint();
}
```

anonymous inner class.

```
}
});
```

Complete Code - see 34d.