Unit 5.2 Event Handling [Last Topic]

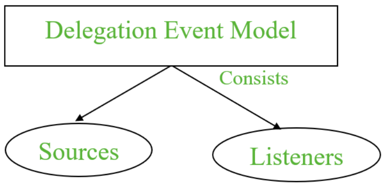
Wednesday, 25 May 2022 10:05 PM

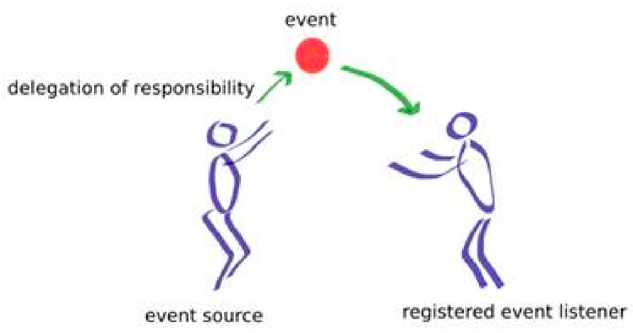
- [ ] **Event Handling:**
- [ ] Event delegation model and Classes
- [ ] Event Listener Interfaces
- [ ] Adapter classes.

Java Event Handling:

- Event is the change in the state of an Object or a Source **[unclicked ->clicked]**
- Even Handling is the mechanism that controls the event and decide what should happen if an event occurs.

Event delegation model:

- If there is a Button, that button would contain two object states
  - Unclicked stage
  - Clicked stage
- If the button gets pressed, then some event may occur
- Delegation Event Model contains two entities
  - 

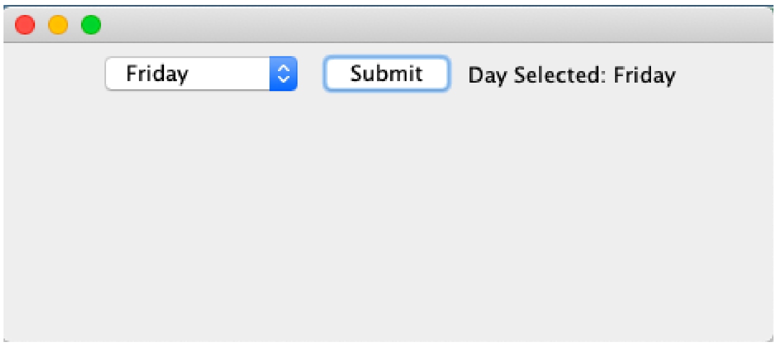
  1. **Source:**
     - It is an Object on which event occurs
     - Button is the example of Source
  2. **Listener**
     - It is known as a Even Handler
     - Responsible to generate response to the event
     - button.addActionListener(al); is the example of listener
- User interface logic is separated from event handler logic
- 

1. Action Listener Even Hander

- If a button gets pressed then perform some event by implementing **action listener** interface while providing definition of
  - public void actionPerformed(ActionEvent e) {

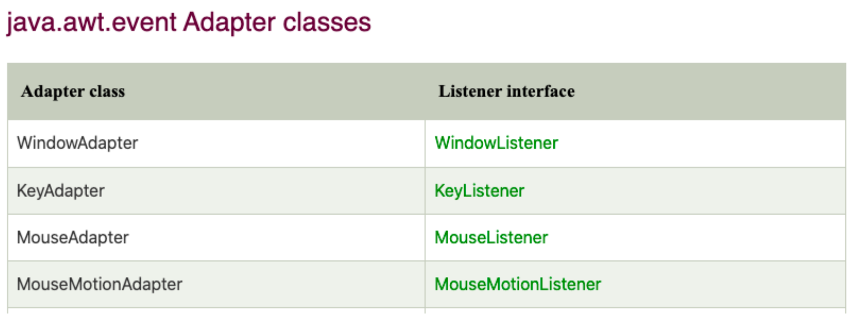
  }
- Code:

```
package Unit_04;

import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class P17_JComboBox {

    public static void main(String[] args) {

        JComboBoxClass obj = new
        JComboBoxClass();

    }

}

class JComboBoxClass extends JFrame{

    JComboBox jbox;

    JButton b;

    JLabel label1;

    JComboBoxClass(){

        //String array to store weekdays
      String week[]= {
"Monday","Tuesday","Wednesday",

"Thursday","Friday","Saturday","Sunday"};


      jbox = new JComboBox<>(week);

      b = new JButton("Submit");
    label1 = new JLabel("Choose a day from the
    list");

    add(jbox);
    add(b);
    add(label1);

    //Event Handler
    ActionListener al = new ActionListener() {

        //Event Handling
        @Override
        public void actionPerformed(ActionEvent e) {

            String data = "";
          if (jbox.getSelectedIndex() != -1)
{
             data = "Day Selected: " +
jbox.getSelectedItem();
             label1.setText(data);
          }

    }
    };

    //Button gets Registered with an action listener
    [Event Handler]
    b.addActionListener(al);

    setLayout(new FlowLayout());
    setVisible(true);
    setSize(400, 400);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
    E);
    }
}
```



2. Adapter classes:

- It simplifies the process of event handling
- Provides empty implementation of all the methods in an event listener interface
- Defines a new class to act as an event listener by extending one of the adapter classes and implement only those methods that you want to use in your code.

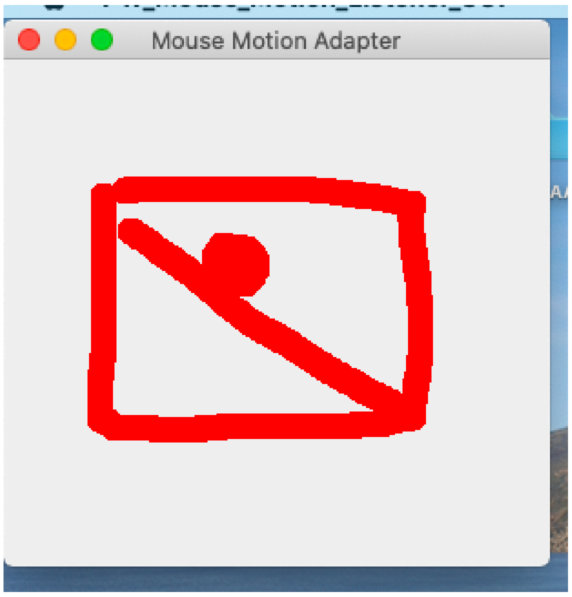


java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener

- We will use Mouse Motion Listener Class for our Work

```
package Unit_04;

import java.awt.*;
import java.awt.event.*;

public class P17_Mouse_Motion_Listener_GUI
extends MouseMotionAdapter {
    Frame f;

    P17_Mouse_Motion_Listener_GUI() {
        f = new Frame("Mouse Motion Adapter");
        f.addMouseMotionListener(this);

    f.setSize(300, 300);
    f.setLayout(null);
    f.setVisible(true);
    f.addWindowListener (new WindowAdapter() {
        public void windowClosing (WindowEvent e)
{
            f.dispose();
        }
     });
    }

    public void mouseDragged(MouseEvent e) {
        Graphics g = f.getGraphics();
        g.setColor(Color.RED);
        g.fillOval(e.getX(), e.getY(), 15, 15);
    }

    public static void main(String[] args) {
        new P17_Mouse_Motion_Listener_GUI();
    }
}
```


Mouse Motion Adapter