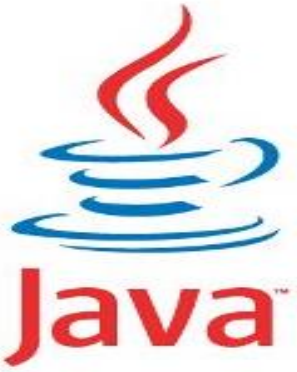

JAVA SE

(CORE JAVA)

LECTURE-32

Today's Agenda

- Event Handling in Java
- Event, Event Source and Event Listeners
- Listener interfaces and classes
- Adapter classes
- Inner classes
- Anonymous classes

Event Handling

- Every GUI application allows its users to interact with it through various devices like mouse, keyboard, scanner, joystick etc.

- When the user performs any action using any of these devices then that action is called an event. For example, clicking a button on screen, typing through keyboard etc.

- As soon as an event occurs the application should respond accordingly. Any such response by an application is programmed by a programmer and is known as event handling. Example, when an user clicks on play button in the media player the song starts playing, the user drags volume control and accordingly volume changes.

Event Handling

- To understand event handling in java, we need to understand 3 important terminologies,

1. Event:- Any action performed by the user on the application is an event. Example, clicking a button, dragging mouse, typing through a keyboard etc.

2. Event Source:- Source in the application program using which the user interacts with it. Example, a button, check box, radio buttons etc.

Event Handling

3. Event Listener:-

- As soon as an event occurs, java automatically invokes a specific method. So, the programmer can define or to be specific override these methods in his own class.
- Such classes and their objects whose methods are invoked by java on any action performed by the user are called Event Listeners.
- For example, whenever the user clicks a button, java automatically calls a method name actionPerformed(). This method is available in an interface called ActionListener. So, the **class which inherits this interface** and overrides the method actionPerformed() will be called as **event listener**.
- A method which is invoked by java automatically is known as **callback**.

Event Handling Example

- Design an applet which contains a button titled ***click me***. Whenever the user clicks the button the background color of the applet should be converted to red.
- **Solution**:-

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class MyButtonApplet extends Applet implements
  ActionListener
{
private Button b;
```

Name of the Event interface

Event Handling Example

```
public void init()
{
b=new Button("Click Me");
add(b); // Belongs to class Container. It adds button on applet
b.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{
setBackground(Color.red);
}
}
```

Event source

Event Listener
Object reference of applet screen

Handling multiple event sources

- Many times while handling action event we might have more than one event source registered for the same event listener. Example, 2 buttons and clicking on any one of them the same actionPerformed() method.

- In such cases we would like to determine which event source has been currently clicked by the user and for this java provides us a method called **getSource()**, belonging to the class ActionEvent whose object is passed as an argument to actionPerformed().

- The method getSource() returns us the memory address of the button(event source) which has been clicked by the user.

Example


```
public class MyTwoButtonApplet extends Applet implements ActionListener
{
 private Button b1,b2;
 public void init()
 {
  b1=new Button("RED");
  b2=new Button("GREEN");
  add(b1);
  add(b2);
  b1.addActionListener(this);
  b2.addActionListener(this);
 }
```

Example


```
public void actionPerformed(ActionEvent e)
 {
   if(e.getSource()==b1)
          setBackground(Color.red);
   else
          setBackground(Color.green);
 }
}
```

Alternate way of detecting Event Source:-

String str=e.getActionCommand();

if(str.eqauls("RED");

getActionCommand() returns caption of the button in the form of string.

Handling Mouse Events

- Java provides two interfaces to handle mouse events, both available in package java.awt.event ,these are –

- MouseMotionListener – This interface consists of 2 methods

1. public void mouseMoved(MouseEvent)
2. public void mouseDragged(MouseEvent)

- MouseListener – This interface consists of 5 methods

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

- MouseEvent class provides important methods to get important information for every mouse event –

1. public int getClickCount() – It is called for every click.
2. public int getX()
3. public int getY()
4. public int getButton() – returns 0 for left, 1 for middle and 2 for right clicks respectively.

Example


```
public class MyMouseApplet extends Applet implements
    MouseListener
{
    public void init()
    {
            this.addMouseListener(this);
    }
    public void mouseClicked(MouseEvent e)
    {
            int x=e.getX();
            int y=e.getY();
            showStatus("X= "+x+" Y= "+y);
    }
```

Example


```
    public void mouseEntered(MouseEvent e)
    {
            setBackground(Color.red);
    }
    public void mouseExited(MouseEvent e)
    {
            setBackground(Color.white);
    }
    public void mousePressed(MouseEvent e)
    {

    }
    public void mouseReleased(MouseEvent e)
    {

    }
}
```

Modified Example

- Modify the previous example so that whenever a click is made on the applet, your code should display the coordinates at the point where the click occurred.
- Solution:-

```
public class MyMouseApplet extends Applet implements MouseListener
{
    int x,y;
    public void init()
    {
            this.addMouseListener(this);
    }
    public void mouseClicked(MouseEvent e)
    {
            x=e.getX();
            y=e.getY();
            repaint();
    }
```

Modified Example

```
public void update(Graphics g)
{
        paint(g);
}
public void paint(Graphics g)
{
        g.drawString("X="+x+",Y="+y,x,y);
}
// Same as previous example.
}
```

*Here we use repaint() to call update() method, which performs three things –

1. Clears background and sets it default white color.
2. Also paints foreground with default color.
3. Calls paint().

Handling Keyboard Events

- Interface for handling keyboard event is KeyListener.

- Methods of KeyListener –

1. public void keyPressed(KeyEvent) – Called when any key is pressed.

2. public void keyTyped(KeyEvent) – Called only for keys with a ASCII or Unicode like, letters and numbers. Not for keys like shift, alt, ctrl etc. For keys with a Unicode both keyPressed() as well as keyTyped() are called.

3. public void keyReleased(KeyEvent) – Called for all keys when a press key is released.

Handling Key Events

- Methods of KeyEvent –

1. public char getKeyChar() – returns characters with Unicode.
2. public int getKeyCode() – returns an integer value which is allocated by java as a key constant to every key.

- Key Constant values –

i. VK_SHIFT
ii. VK_ALT
iii. VK_ENTER
iv. VK_F1 to F12
v. VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT

Example

- Design an applet which should behave as follows –

1. If an ASCII based key is typed then it should be displayed in the canvas.
2. If shift, control or alt are pressed, then name of the key should appear in the status bar.

- Solution –

```
public class MyKeyApplet extends Applet implements KeyListener
{
    StringBuffer str=new StringBuffer(" ");
    public void init()
    {
            this.addKeyListener(this);
    }
    public void paint(Graphics g)
    {
            g.drawString(str.toString(),50,100);
    }
```

Example

```
public void keyTyped(KeyEvent k)
{
        char c=k.getKeyChar();
        str=str.append(c);
        repaint();
}
public void update(Graphics g)
{
        paint(g);
}
public void keyReleased(KeyEvent k)
{

}
```

```
public void keyPressed(KeyEvent e)
{
        int x=e.getKeyCode();
        switch(x)
        {
        case KeyEvent.VK_SHIFT:
        showStatus("SHIFT");
        break;

        case KeyEvent.VK_CONTROL:
        showStatus("CTRL");
        break;
        }
}

}
```

Adapter Classes

- A problem in event handling using Listener interfaces is, that we have to override all the methods even though their body was left blank.
- To reduce this overhead of programmers java provides us with a bunch of classes each implementing a particular interface and have overridden all methods with empty body.
- In this way we can directly inherit these classes and prevent the overhead of overriding all the methods.
- Such classes are called Adapter classes.

Adapter Classes

- Following is the list of Adapter classes –

1. MouseAdapter
2. MouseMotionAdapter
3. FocusAdapter
4. KeyAdapter
5. ComponentAdapter
6. ContainerAdapter
7. WindowAdapter

Adapter Classes Example

```
public class MyApplet extends Applet
{
    public void init()
    {
            MyMouseAdapter obj=new MyMouseAdapter(this);
            this.addMouseListener(obj);
    }
}
class MyMouseAdapter extends MouseAdapter
{
    MyApplet m;
    public MyMouseAdapter(MyApplet a)
    {
            m=a;
    }
    public void mouseClicked(MouseEvent e)
    {
            m.showStatus("Mouse Clicked at "+e.getX()+","+e.getY());
    }
}
```

Inner classes

- Just like nested loop and if statements we can have nested or inner classes.
- All inner classes can access the methods and data members of the outer class.
- Example –

```
class A
{
int x=10;
class B
{
int a=20;
public void show( )
{
System.out.println(x);
System.out.println(a);
}
}
}
```

Event Handling using Inner class

```
public class MyApplet extends Applet
{
    public void init()
    {
            MyMouseAdapter obj=new MyMouseAdapter();
            this.addMouseListener(obj);
    }

    class MyMouseAdapter extends MouseAdapter
    {
            public void mouseClicked(MouseEvent e)
            {
            showStatus("Mouse Clicked at "+e.getX()+","+e.getY());
            setBackground(Color.red);
            }
    }
}
```

Anonymous Classes

- Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class at the same time.
- They are like local classes except that they do not have a name.
- Use them if you need to use a local class only once.
- Example –

Anonymous class

```
class A
{
 public void show( )
 {
 ---
 ---
 }
}
```

```
class B
{
 public void display(A obj)
 {
 ---
 ---
 }
}
```

***When we call display, we have to create an object of class A and then pass it as an argument. Since, used only once it uses unnecessary memory. To avoid this we can write the above program in much succinct manner.**


```
class B
{
 int a=10;
 public void display(new A()
 {
 System.out.println(a);
 });
}
```

1. **The anonymous class will be inner class to the class in which it is declared and defined.**
2. **So, the .class file generated after compilation will be named after the outer class. In this example, it will be B$1.class.**
3. **The anonymous class will also be derived class of the class whose object is to be passed as an argument. In this example, anonymous class will be derived class of class A.**

Anonymous class in Event Handling

- From the previous example –

```
public class MyApplet extends Applet
{
    public void init()
    {
            this.addMouseListener(new MouseAdapter()
            {
              public void mouseClicked(MouseEvent e)
              {
              showStatus("Mouse Clicked at "+e.getX()+","+e.getY());
              setBackground(Color.red);
              }
            });
    }
}
```

**** 2 .class files will be generated named, MyApplet.class and MyApplet$1.class.***


Java

End Of Lecture 32

Thank You

For any queries mail us @: scalive4u@gmail.com
Call us @ : 0755-4271659, 7879165533

Agenda for Next Lecture:

1. **Developing GUI Applications in Java.**
2. **Using "swing" package.**