Lecture-10

GUI - Graphical user interface

Support of GUI :-

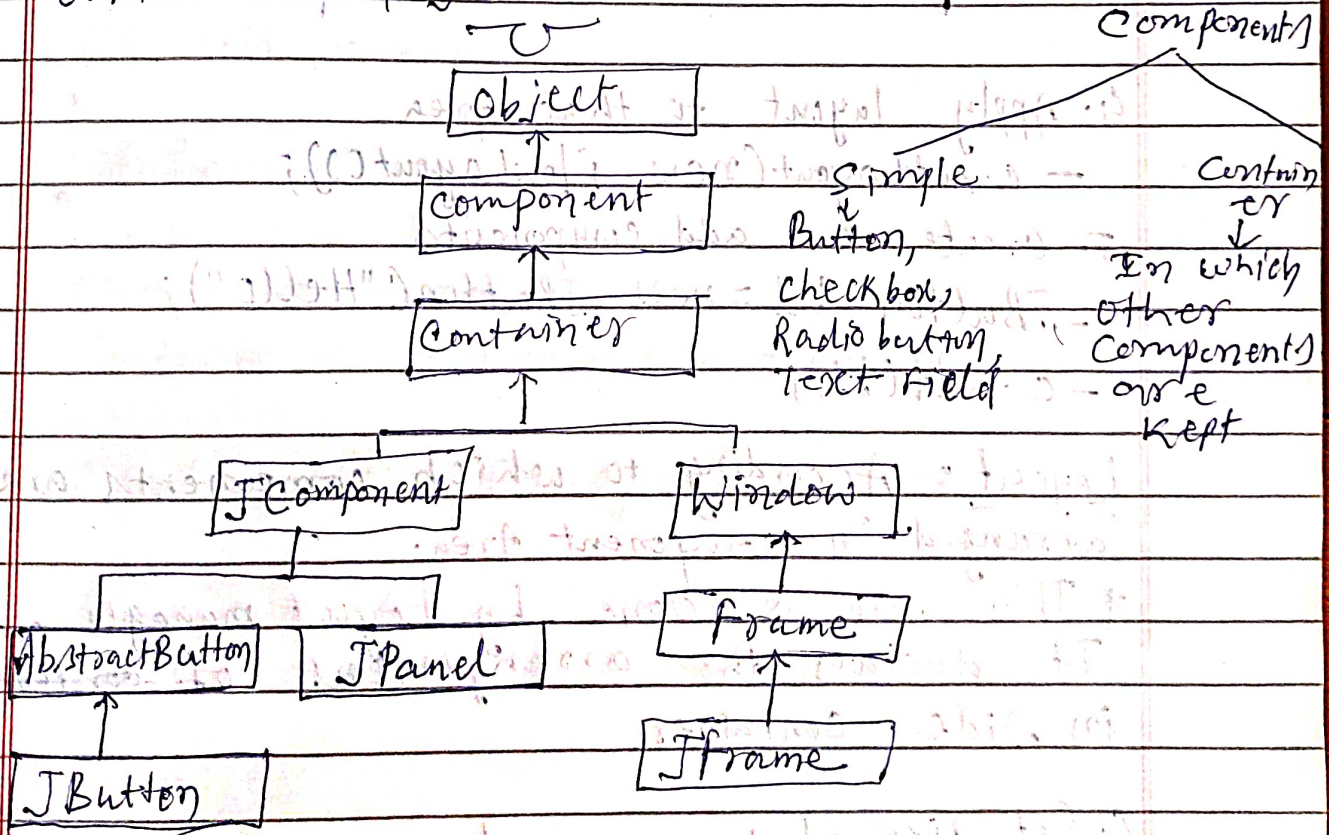
* Abstract Window Toolkit (AWT) & Swing packages
- Provide rich set of user interface components.
- java.awt & javax.swing
- Old (AWT) Vs New (Swing)

* Components in awt & swing :-
(start with J)
- Frame, JFrame
- Menu, JMenu
- Button, JButton
- TextField, JTextField
- Label, JLabel
- and many more

* Use Java API Documentation well.

* AWT is dependent on the underlying O.S. on which it is working.
* When we are creating a component then it may lead to some dependency with the underlying O.S. So when we run the code in some other O.S. it may show some problem.
* By considering these problems AWT is said as heavy weight components because it is creating a process in underlying O.S., and synchronising with the button created using AWT.

Swing:- These are considered as light weight components. These components are purely written in java.



* In java first we create container then within this we can place components.

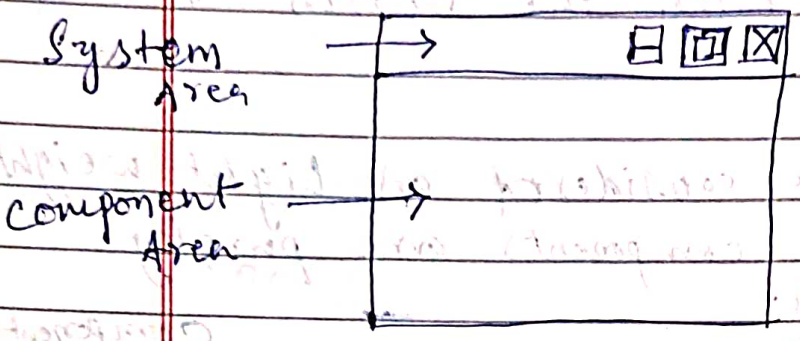
GUI creation

Steps :-

1. Import required packages
   - e.g. :- swing, awt
2. Setup the top level container
   e.g. :- JFrame myframe = new JFrame();

Window = frame in java

3. Get the component area of the top level container.
   ~~Container c = new Container~~
   Container c = myframe.getContentPane();



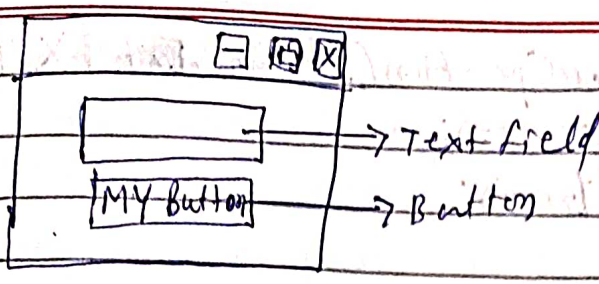
returns the area where we can add the components.

4. Apply layout to that Area
   - c.setLayout(new FlowLayout());
5. Create & add components
   - JButton b1 = new JButton("Hello");
   - c.add(b1);

Layout :- According to which components are arranged in component Area.

* This work is done by Layout manager. It decides the arrangement of components in side container.

6. Set size of frame and make it visible.
   - myframe.setSize(200,200);
   - myframe.setVisible(true);




```
import java.awt.*;
import javax.swing.*;

public class GUITest
{
JFrame myframe;
JTextField tf;
JButton b1;


public void initGui()
{
myframe = new JFrame();

Container c = new Container();
Container c = myFrame.getContentPane();

c.setLayout(new FlowLayout);

JTextField tf = new JTextField(10);
JButton b1 = new JButton("MY Button");

c.add(tf);
c.add(b1);

myframe.setSize(
```

```
myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
myframe.setSize(200,150);
myFrame.setVisible(true);
}


public GUITest()
 {
  initGUI();
 }

public static void main(String args[])
 {
 GUITest gui=new GUITest();

 }
}
```

Different Layout Managers :-

* Layout Manager :-

- Java supplies many layout managers, Five commonly used are :-
  - FlowLayout
  - GridLayout
  - BorderLayout
  - BoxLayout
  - GridBagLayout

Flowlayout :- The way in which you add components in that way it shows the components. If you stretch the size of container then the component which is present in second row will come to the 1st row. It fills the rows up to which its end will not come. When end will come it goes to next line.

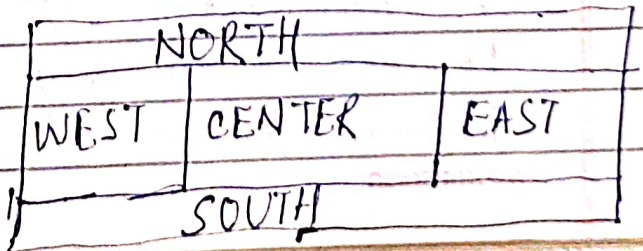
* we can not fix the size of components. According to the button name the length of button will be fixed. According to ~~length of text~~ no. of characters the length of textfield will be fixed.
* component size can be fixed/set by using a method - SetPreferedSize()

Grid Layout :- We are adding components in row and column manner like adding elements into matrix.

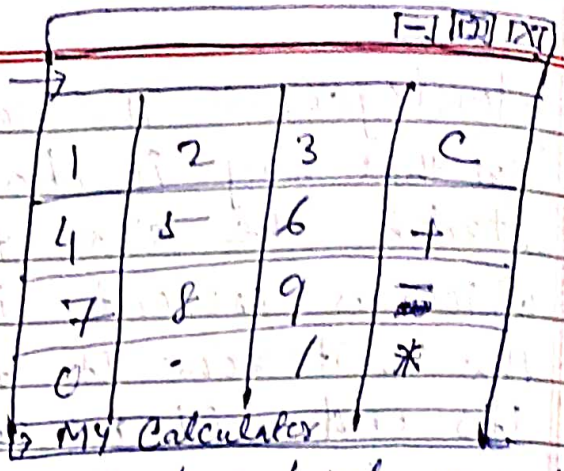
* Splits the panel into a grid with given no. of rows and columns.
* places components in to the grid cells.
* forces the size of each component to occupy the whole cell.
* Allows additional spacing between cells.

Borderlayout :-

* Divides the area in to five regions.
* Adds a component to the specified region.
* Forces the size of each component to occupy the whole region.



import java.awt.*;
import javax.swing.*;
public class GUICalc
{



JPanel :- It is a component which can keep a no. of components with in it.

```
import java.awt.*;
import javax.swing.*;
public class GUICalc
{
   JFrame myFrame;
   JButton b0,b1,b2,b3,b4,b5,b6,b7,b8,b9;
   JTextField tf;
   JLabel l6;
   JPanel P;
public GUICalc()
  {
   initGUI();
  }
public void initGUI()
 {
  b0=new JButton("0");
  b1=new JButton("1");
  b2=new JButton("2");
  b3=new JButton("3");
  b4=new JButton("4");
  b5=new JButton("5");
  b6=new JButton("6");
  b7=new JButton("7");
```

```
myframe = new JFrame();
Container c = myframe.getContentPane();
c.setLayout(new BorderLayout());
tf = new JTextField();
lb = new JLabel();
p = new JPanel(new GridLayout(4,4));
p.add(b0);
p.add(b1);
p.add(b2);
p.add(b3);
p.add(b4);
p.add(
```

Lecture - 11 Event handling

* A class who implements interface, it has to provide definition of the methods which are present inside the interface. If it will not provide then we can not create object of that class.

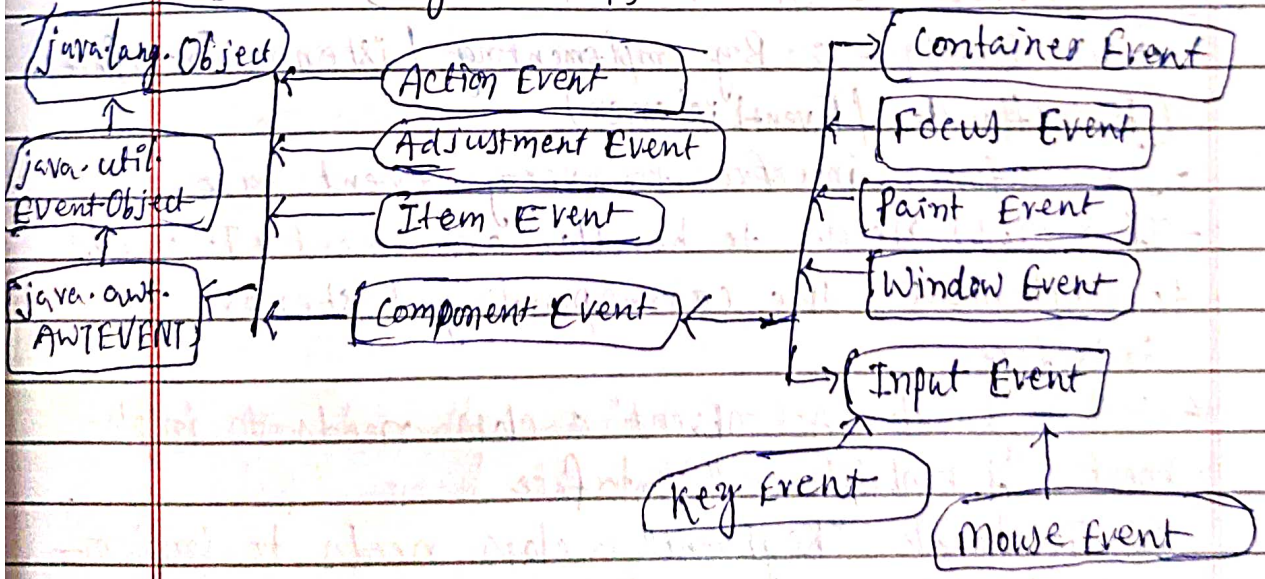
Events :- GUIs generate events when the user interacts with GUI.

Ex:- - clicking a button
- moving the mouse
- closing window.

* In java, events are represented by objects.
- These objects tells us about event and its source

Example:-

* ActionEvent (clicking a button)
* WindowEvent (Doing something with window e.g. closing, minimizing etc)
* Both AWT and swing components (not all) generate events.
- java.awt.event.*;
- javax.swing.event.*;



Event delegation model :- Event handling responsibility is given to some class.
- Processing of an event is delegated to a particular object (handelers) in the program.
- Publish - Subscribe
- Seperate UI code from program logic.

Event handling steps :-
Step-1 :- Create components which can generate events. (Button)
Step-2 :- Build component (objects) that can handle events (Event-Handlers)
Step-3 :- Register handlers with generators.

① Event generators :-
- Button, Mouse, Key, Window etc.
JButton b1 = new JButton("Hello");
- Now b1 can generate events.

② Event handler :- (How to handle event)
- First Technique :- By implementing Listener Interfaces. (EventHandlers / EventListener)
- Java defines interfaces for every event type.
- If a class needs to handle an event. It needs to implement the corresponding Listener interface.
Ex :- ① To handle "ActionEvent" a class needs to implement "ActionListener" interface
② To handle "KeyEvent" a class needs to implement "KeyListener" interface.

Event Listener interfaces of package "java.awt.event"

java.util.EventListener
- ActionListener
- AdjustmentListener
- ComponentListener
- ContainerListener
- FocusListener
- ItemListener
- KeyListener
- MouseListener
- MouseMotionListener
- TextListener
- WindowListener

Event Handling Process[3]

Every Handlers

* By implementing an interface the class agrees to implement all the methods that are present in that interface.
* Implementing an interface is like signing a contract.
* Inside the method the class can do what ever it wants to do with that event.
* Event Generator and Event handler can be same or different classes.

Ex: Action Event handling :-

To handle events generated by Button. A class need to implement "ActionListener" interface.

```
class TEST implements ActionListener
{
public void actionPerformed(ActionEvent ae)
{
   //do something
}
}
```

Event handling process [4]

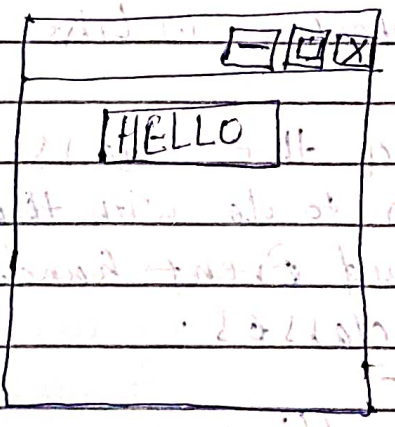
Registering Handler with Generator :-

* The event generator is told about the object which can handle its events.
* Event Generator have a method
  - add_____Listener(_____)

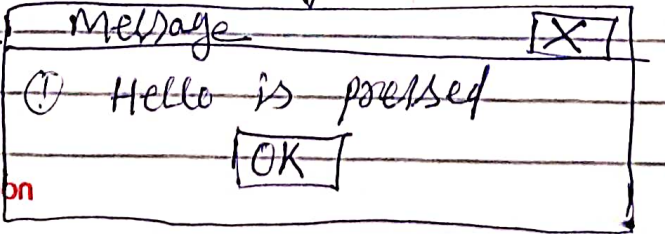
Ex:

```
b1.addActionListener(objectOfTestClass)
```

Ex:



when HELLO button is pressed, the Dialog box will display.



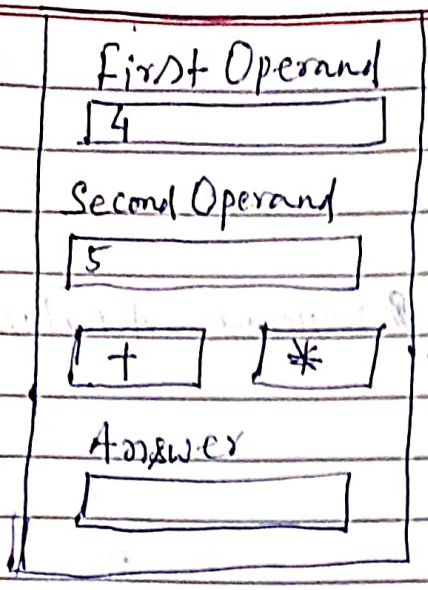
```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class ActionEventTest implements ActionListener
{ JFrame frame;
  JButton bHello;

public void initGUI()
  {
  frame = new JFrame();
  bHello=new JButton("Hello");
  ~~Container con=new Container();~~
  Container con=frame.getContentPane();
  con.add(bHello);
  frame.size(200,200);
  frame.setVisible(True);
  bHello.addActionListener(this); //Step-3 (Event Registration)
  }
Step-2 }

public void actionPerformed(ActionEvent ae)
  {
  JOptionPane.ShowMessageDialog("Hello is pressed");
  }

Public ACtionEventTest()
{ initGUI();
}
public static void main(String args[])
  {
  ActionEventTest o=new ActionEventTest();
  }
```




```
class SmallCalcApp implements ActionListener
{
  JFrame frame;
  JLabel firstOperand, secondOperand, answer;
  JTextfield op1, op2, ans;
  JButton plus, mul;

public void initGUI()
{
  frame = new JFrame();
  firstOperand = new JLabel("First Operand");
  secondOperand = new JLabel("Second Operand");
  answer = new JLabel("Answer");
  op1 = new JTextField(20);
  op2 = new JTextField(20);
  ans = new JTextField(20);
  plus = new JButton("+");
  mul = new JButton("*");
Container c = frame.getContentPane();
  c.setLayout(new FlowLayout());
  c.add(firstOperand);
  c.add(secondOperand); c.add(op1);
```

```
c.add(SecondOperand);
c.add(op2);
c.add(plus);
c.add(mul);
c.add(ans);
c.add(ans);
frame.size(200,200);
frame.setVisible(true);
fram.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

plus.addActionListener(this);
mul.addActionListener(this);
}

public void actionPerformed(ActionEvent ae)
{
  String oper, result;
  int num1, num2, res;
  if(ae.getSource() == plus)
  {
    oper = op1.getText();
    num1 = Integer.parseInt(oper);

    oper = op2.getText();
    num2 = Integer.parseInt(oper);

    res = num1 + num2;
    result = res + "";
    ans.setText(result);
  }
```

```
else if(ae.getSource() == mul)
{
    oper = op1.getText();
    num1 = Integer.parseInt(oper);

    oper = op2.getText();
    num2 = Integer.parseInt(oper);

    res = num1 * num2;
    result = res + "";
    ans.setText(result);
}
}

public SmallCalcApp()
{
    initGUI();
}

public static void main(String args[])
{
    SmallCalcApp o = new SmallCalcApp();
}
}
```

Date __/__/__
Page _____

Eventhandling Lecture -12

* Handling Mouse Events :- Mouse events can be trapped for any GUI component that inherit from Component class.
Ex:- JPanel, JFrame & JButton etc.

* To handle mouse event two types of Listener interfaces are available
  - MouseMotionListener
  - MouseListener

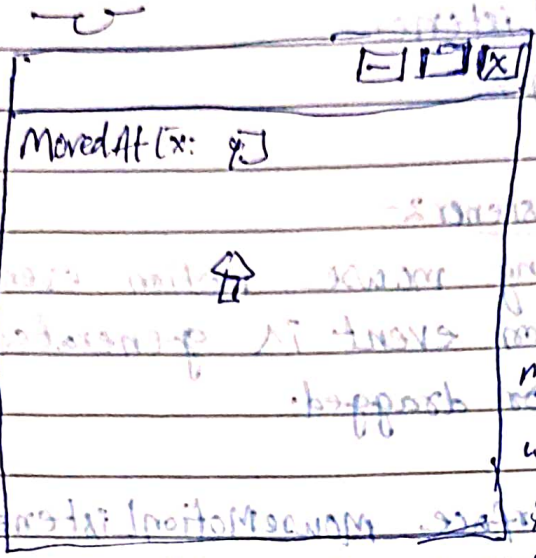
- MouseMotionListener :-
  - For processing mouse motion events
  - Mouse motion event is generated when mouse is moved or dragged.

```
public interface MouseMotionListener {
    public void mouseDragged(MouseEvent me);
    public void mouseMoved(MouseEvent me);
}
```

- MouseListener :-
  - Mouse event is generated when mouse is
    * pressed
    * Release
    * clicked (Pressed & released without moving cursor)

* Enter (Mouse cursor enter the bound of component)
* Exit (Mouse cursor leaves the bound of component)

```
public interface MouseListener {
public void mousePressed(MouseEvent me);
public void mouseClicked(  "  );
public void mouseReleased(  "  );
public void mouseEntered(  "  );
public void mouseExited(  "  );
}
```



As here we are moving the mouse with in the frame so, while registering handler with Listener we use myframe.

```
class EventEx implements MouseMotionListener {
        JFrame myframe;
        JLabel coord;

public void initGUI()
{
        JFrame myframe=new JFrame();
        coord=new JLabel("moved At");
    Container c=myframe.getContentPane();
        c.add(coord);
        c.setLayout(new FlowLayout());
        myframe.size(200,200);
        myframe.setVisible(true);
        myframe.setDefaultCloseOperation(JFrame.
                                EXIT_ON_CLOSE);
```

```
myFrame.addMouseMotionListener(this);
}

public void mouseMoved(MouseEvent me)
{
  int x=me.getX();
  int y=me.getY();
  coord.setText("Moved At ["+x+"] ["+y+"]");
}

public void mouseDragged(MouseEvent me)
{
  int x=me.getX();
  int y=me.getY();
  coord.setText("Dragged At ["+x+"] ["+y+"]");
}

public EventEx();
{
  initGUI();
}
public static void main(String args[])
    {
      EventEx e=new EventEx();
    }

}
```

WindowEvent :-

Public Interface WindowListener

{

public void windowActivated(WindowEvent we);

public void windowClosed(WindowEvent we);

" " windowClosing(");

" " windowDeactivated(");

" " windowDeiconified(");

" " windowIconified(");

" " windowOpened (");

}