Sukkur IBA

Fahim Akhtar Rajputt

Object Oriented Programming (JAVA)

Lecture 34 and 35

Introduction to GUI

- Graphical User Interface ("Goo-ee")
  - Pictorial interface to a program
    - Distinctive "look" and "feel"
- GUIs built from components
  - Component: object with which user interacts
  - Examples: Labels, Text fields, Buttons, Checkboxes

Awt & Swing

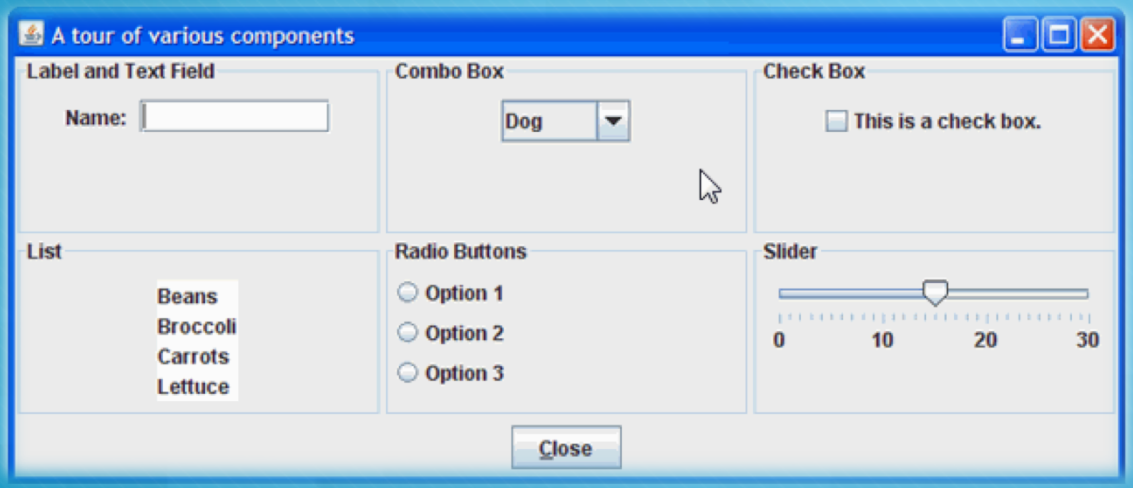
- Sun's initial idea: create a set of classes/methods that can be used to write a multi-platform GUI (Abstract Windowing Toolkit, or AWT)
  - problem: not powerful enough; limited; a bit clunky to use
- Second edition (JDK v1.2): Swing
  - a newer library written from the ground up that allows much more powerful graphics and GUI construction
- ***Drawback:*** *Both exist in Java now; easy to get them mixed up; still have to use both sometimes!*

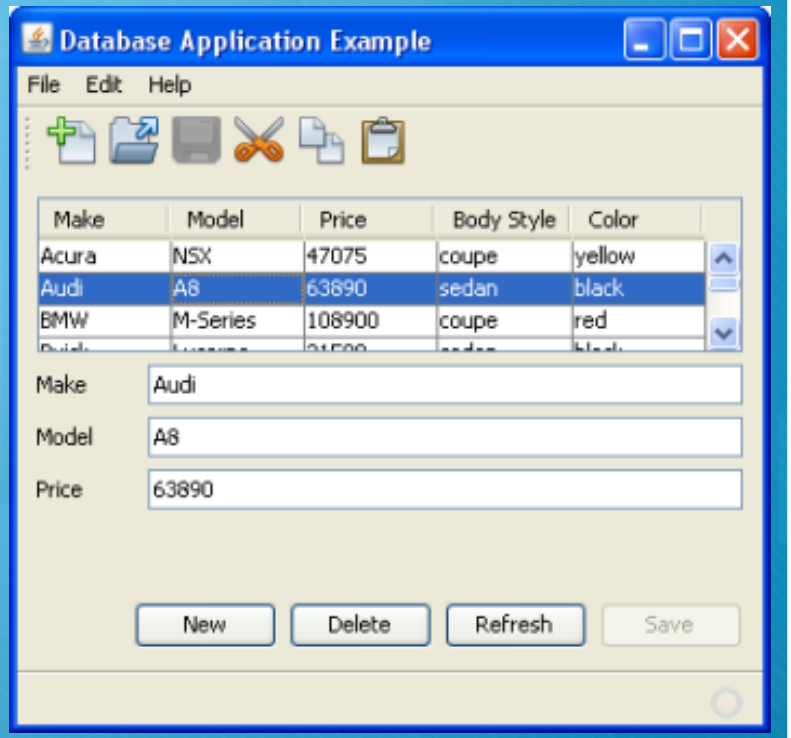
Awt & Swing

```
java.lang.Object
  +--java.awt.Component
        +--java.awt.Container
              |
              +--javax.swing.JComponent
              |     +--javax.swing.JButton
              |     +--javax.swing.JLabel
              |     +--javax.swing.JMenuBar
              |     +--javax.swing.JOptionPane
              |     +--javax.swing.JPanel
              |     +--javax.swing.JTextArea
              |     +--javax.swing.JTextField
              |
              +--java.awt.Window
                    +--java.awt.Frame
                          +--javax.swing.JFrame
```

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

GUI Applications





Steps for GUI Creation

1. Import required packages, eg: swing or awt
2. Setup the top level container

```
JFrame myFrame = new JFrame();
```

3. Get component area of the top level container

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

   - System Area
   - Component Area

4. Apply Layout to that Area

```
c.setLayout(new FlowLayout());
```

Steps for GUI Creation

5. Create and add components:

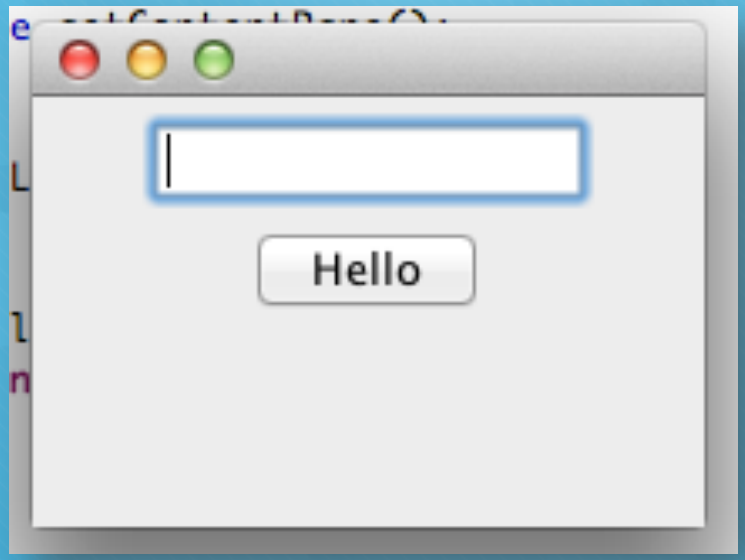
```
JButton b1 = new Jbutton("Hello");

c.add(b1);
```

6. Set size of Frame and make it visible

```
myFrame.setSize(200,200);

myFrame.setVisible(true);
```

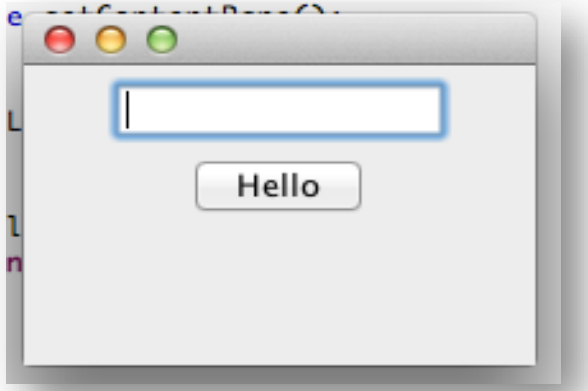
NOTE: When you are using FlowLayout you can't set the size of components

First Java GUI Application



```
//Step1:
import java.awt.*;
import javax.swing.*;

class guiTest{
        JFrame myFrame;  // Step 2
        JTextField myTextField;
        JButton myButton1;

        void initGUI(){
                myFrame = new JFrame();
        //Step3:
                Container c = myFrame.getContentPane();
        //Step4:
                c.setLayout(new FlowLayout());
        //Setp5:
                JTextField myTextField = new JTextField(10);
                JButton myButton1 = new JButton("Hello");
                c.add(myTextField);
                c.add(myButton1);
        //Step6:
                myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                myFrame.setSize(200,150);
                myFrame.setVisible(true);
        }
}
```

```
public class GUI{

        public static void main(String[]
args) {
                guiTest gt = new guiTest();
                gt.initGUI();

        }
}
```



2 different ways you may find

Composition

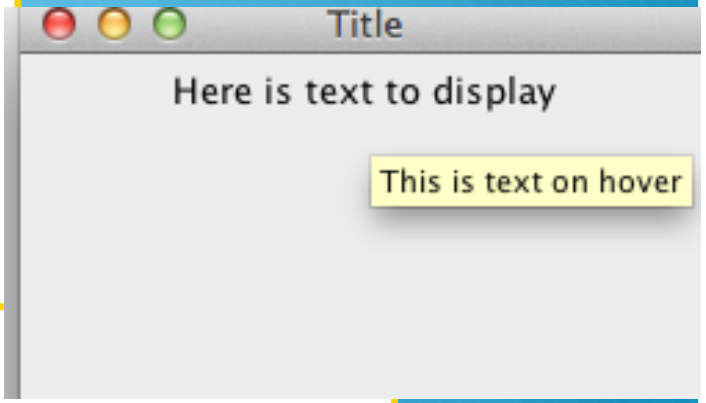
```
class GUITest{
  JFrame frame;
  Container c;
    public GUITest(){
      frame = new JFrame();
      c = frame.getContentPane();
      ...
      frame.setVisible(true);
    }
        ...
}
```

Inheritance

```
class GUITest extends JFrame{
  Container c;
    public GUITest(){
      c = getContentPane();
      ...
      setVisible(true);
    }
        ...
}
```

Example 1

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;

        public class GUI extends JFrame{
                private JLabel item1;
                GUI(){
                        super("Title");
                        setLayout(new FlowLayout());
                        item1 = new JLabel("Here is text to display");
                        item1.setToolTipText("This is text on hover");
                        add(item1);
                }
}
```



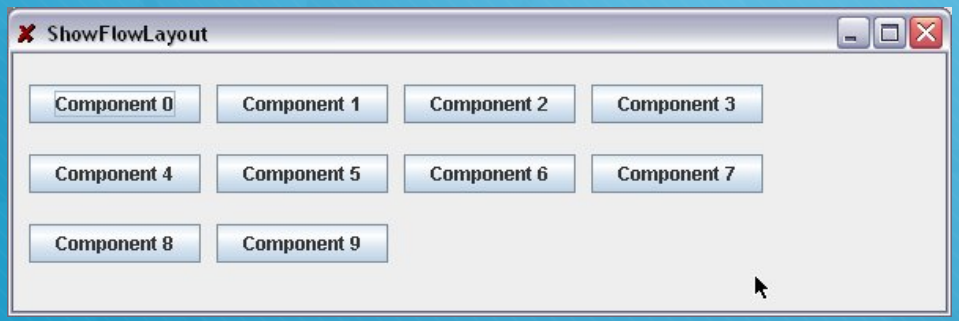

```
import javax.swing.JFrame;
public class GUITest {
        public static void main(String args[]){

                GUI gt = new GUI();
                gt.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                gt.setSize(250,150);
                gt.setVisible(true);
        }
}
```

Layout Managers

- Java provides many layout managers, common layout managers are:
  - FlowLayout
  - GridLayout
  - BorderLayout

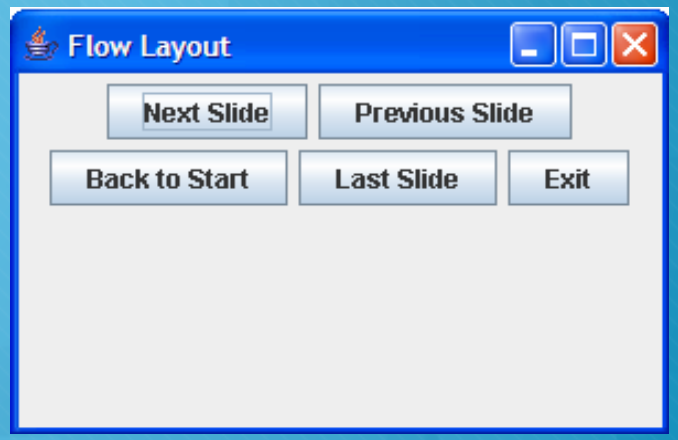
FlowLayout

- With **flow layout**, the components arrange themselves from left to right in the order they were added
- You can't set the size of components here, size is automatic, for eg. Lable of a button

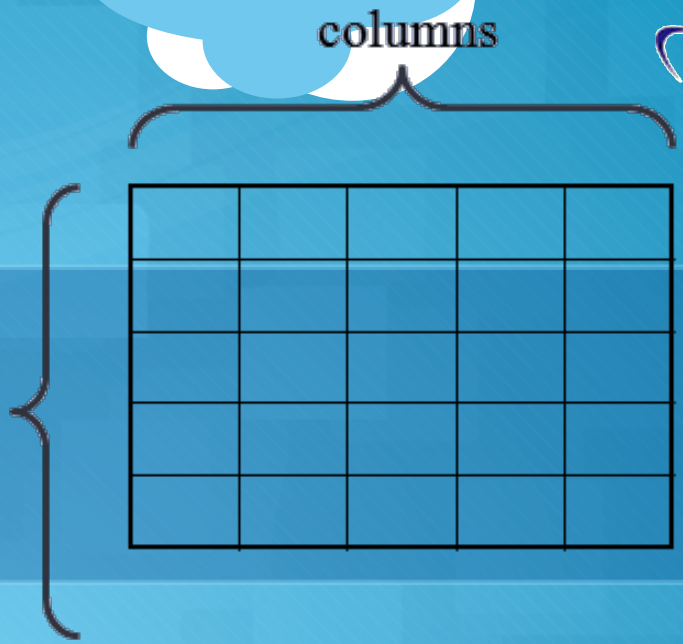


Demo of FlowLayout

```
c.setLayout( new FlowLayout( ) );

b1 = new JButton("Next Slide");
b2 = new JButton("Previous Slide");
b3 =  new JButton("Back to Start");
b4 = new JButton("Last Slide");
b5 = new JButton("Exit");

c.add(b1);
c.add(b2);
c.add(b3);
c.add(b4);
c.add(b5);
```



GridLayout



- With **grid layout**, the components arrange themselves in a matrix formation (rows, columns)
- First it fills the rows and then columns
- Forces the component to occupy whole size of cell

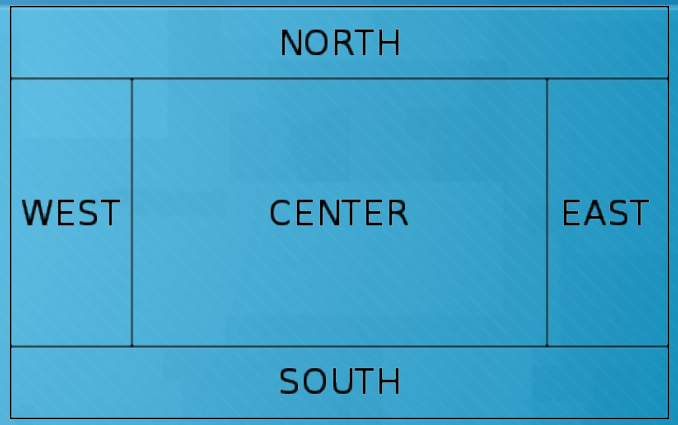
Demo of GridLayout

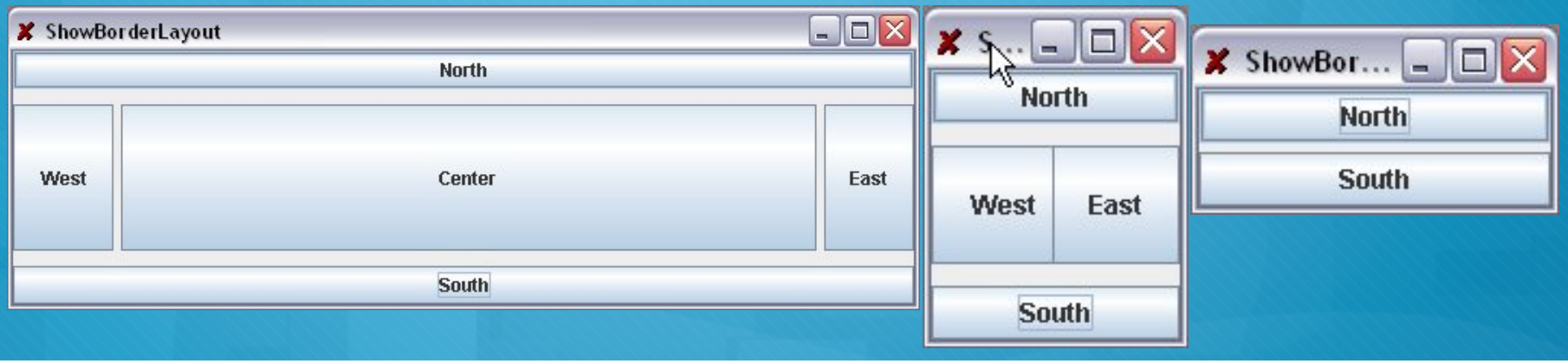
```
c.setLayout( new GridLayout( 3 , 2 ) );
b1 = new JButton("Next Slide");
b2 = new JButton("Previous Slide");
b3 = new JButton("Back to Start");
b4 = new JButton("Last Slide");
b5 = new JButton("Exit");

c.add(b1);
c.add(b2);
c.add(b3);
c.add(b4);
c.add(b5);
```

```
//Space Horizontal and Vertical
c.setLayout( new GridLayout( 3 , 2 , 10 , 20) );
```

BorderLayout

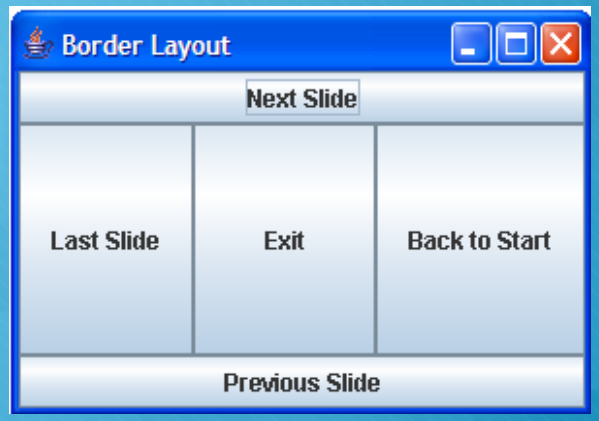
- Divide the area into 5 regions
- Add the component to specific region
- Forces the size of each component to occupy whole the region





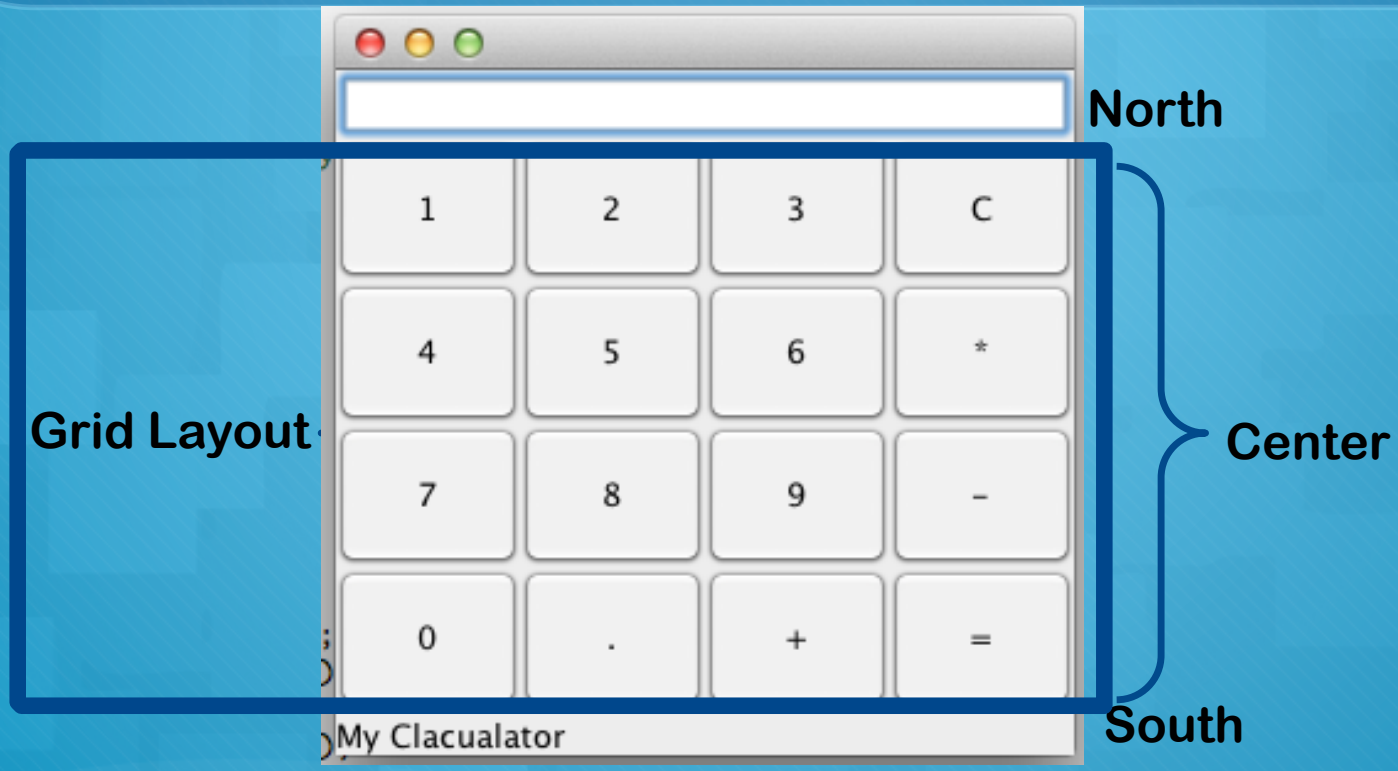
Demo of BorderLayout

```
c.setLayout( new BorderLayout( );
b1 = new JButton("Next Slide");
b2 = new JButton("Previous Slide");
b3 = new JButton("Back to Start");
b4 = new JButton("Last Slide");
b5 = new JButton("Exit");

c.add( b1 , BorderLayout.NORTH );
c.add( b2 , BorderLayout.SOUTH );
c.add( b3 , BorderLayout.EAST );
c.add( b4 , BorderLayout.WEST );
c.add( b5 , BorderLayout.CENTER);
```



- Container default layout is BorderLayout
- JPanel default layout is FlowLayout
  - Speakers in car example

GUI Calculator



```
// File CalculatorGUI.java
import java.awt.*;
import javax.swing.*;
public class CalculatorGUI {
        JFrame fCalc;
        JButton b1, b2, b3, b4, b5, b6, b7, b8, b9, b0;

        JButton bPlus, bMinus, bMul, bPoint, bEqual, bClear;
        JPanel pButtons;
        JTextField tfAnswer;
        JLabel lMyCalc;

        //method used for setting layout of GUI public
        void initGUI ( ) {
                fCalc = new JFrame();
                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");
                b8 = new JButton("8");  b9 = new JButton("9");
                bPlus = new JButton("+");  bMinus = new JButton("-");
                bMul = new JButton("*");  bPoint = new JButton(".");
                bEqual = new JButton("=");  bClear = new
JButton("C");

                tfAnswer = new JTextField();

                lMyCalc = new JLabel("My Clacualator");
```

Continue

```
        //creating panel object and setting its layout
        pButtons = new JPanel (new GridLayout(4,4));
        //adding components (buttons) to panel
        pButtons.add(b1);  pButtons.add(b2);
        pButtons.add(b3);  pButtons.add(bClear);
        pButtons.add(b4);  pButtons.add(b5);
        pButtons.add(b6);  pButtons.add(bMul);
        pButtons.add(b7);  pButtons.add(b8);
        pButtons.add(b9);  pButtons.add(bMinus);
        pButtons.add(b0);  pButtons.add(bPoint);
        pButtons.add(bPlus);  pButtons.add(bEqual);
        // getting componenet area of JFrame
        Container con = fCalc.getContentPane();
        con.setLayout(new BorderLayout());

        //adding components to container
        con.add(tfAnswer, BorderLayout.NORTH);
        con.add(lMyCalc, BorderLayout.SOUTH);
        con.add(pButtons, BorderLayout.CENTER);
        fCalc.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fCalc.setSize(300, 300);
        fCalc.setVisible(true);
} // end of GUI Method

public CalculatorGUI () { // default constructor
        initGUI ();
}
public static void main (String args[ ]){
        CalculatorGUI calGUI = new CalculatorGUI (); }
} // end of class
```

Events

- GUI Generates events when the user interact with interface; clicking button, moving the mouse, closing window etc..
- In java, events are represented by objects
  - ActionEvent (When you click on a button)
  - WindowEvent (When you do something with window like; close, minimize or maximize a window)
  - KeyEvent, MouseEvent, InputEvent…
- So there are several events for every action we perform on a GUI interface

Events

- Both awt and swing provides events
  - java.awt.event.*;
  - java.swing.event.*;

```
java.lang.Object
  +--java.util.EventObject
        +--java.awt.awt.event
              +--java.awt.event.ActionEvent
              +--java.awt.event.TextEvent
              +--java.awt.event.ComponentEvent
                    +--java.awt.event.FocusEvent
                    +--java.awt.event.WindowEvent
                    +--java.awt.event.InputEvent
                          +--java.awt.event.KeyEvent
                          +--java.awt.event.MouseEvent
```

Event Handling

- **Event Delegation Model**
  - Processing of an Event is delegated to a particular object (Handler) in program
  - Publish-subscribe model
  - Separate GUI from logic source code

Steps in Event Handling

1. Create a component which can generate events
2. Build class that can handle the events (Event Handler)
3. Register the handler with generators

Step 1

- We have learned about buttons, text fields, window etc..

```
JButton b1 = new JButton (“Hello”);
```

- Now, b1 can generate events

Step 2

- Implement Listener interfaces
  - If a class needs to handle an event it needs to implement corresponding listener interface
    - ActionListener for ActionEvent, MouseListener for MouseEvent, KeyListener for KeyEvent…

```
public interface ActionListener {
        public void actionPerformed(ActionEvent e);
}
```

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

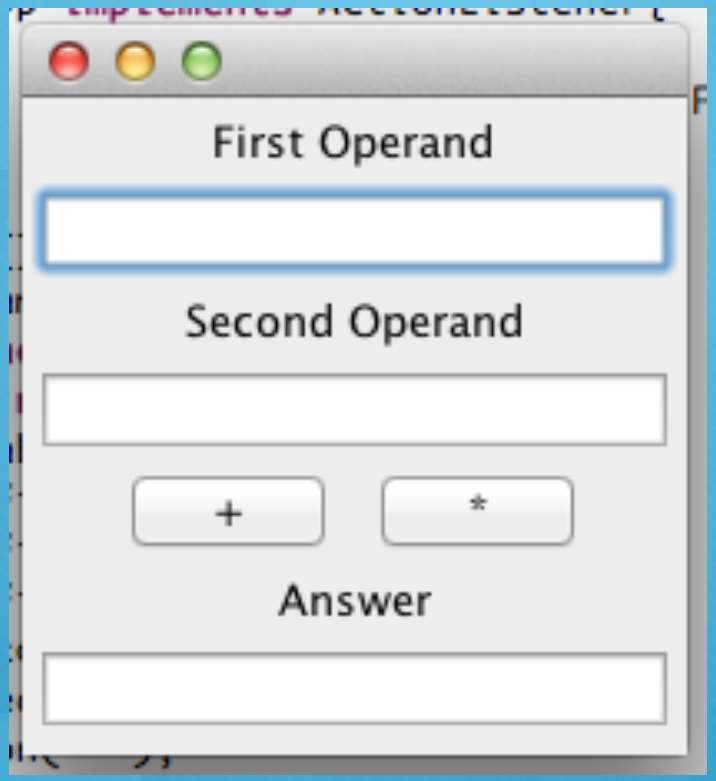
CONTRACT:
By implementing an interface a class agrees to implement all of the methods of interface

Step 3

- The event generator is told about the object which can handle its events
- Event generator have a method

```
add_____Listener(_____);

b1.addActionListner(ObjectOfTestClass);
```

GUI Example – Small Calculator



```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class SmallCalcApp implements ActionListener{
// There can be multiple listeners separating by comma.
        JFrame frame;
        JLabel firstOperand, secondOperand, answer; JTextField op1, op2, ans;
        JButton plus, mul;
        // setting layout
        public void initGUI() {
                frame = new JFrame();
                firstOperand = new JLabel("First Operand");
                secondOperand = new JLabel("Second Operand");
                answer = new JLabel("Answer");
                op1 = new JTextField (15);
                op2 = new JTextField (15);
                ans = new JTextField (15);
                plus = new JButton("+");
                plus.setPreferredSize(new Dimension(70,25));
                mul = new JButton("*");
                mul.setPreferredSize(new Dimension(70,25));

                Container cont = frame.getContentPane();
                cont.setLayout(new FlowLayout());
                cont.add(firstOperand); cont.add(op1);
                cont.add(secondOperand); cont.add(op2);
                cont.add(plus); cont.add(mul);
                cont.add(answer); cont.add(ans);

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

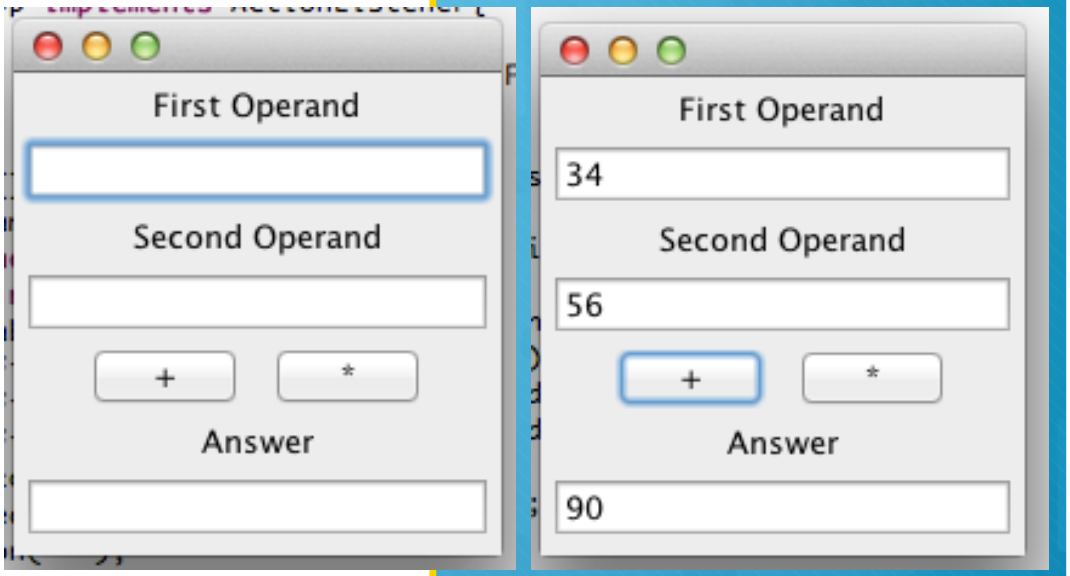
Step 1.
Create Components

Step 3.
Registration

Continue

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(200, 220);
            frame.setVisible(true);
        }
        public SmallCalcApp ( ) {   // Constructor
            initGUI();
        }
        public void actionPerformed(ActionEvent event){
            String oper, result; int num1, num2, res;
            if (event.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 (event.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 static void main(String args[]) {
            SmallCalcApp scApp = new SmallCalcApp();
        }
}// end class
```

Step 2.
methods of interfaces



Incomplete implementation of interface?

- You will find interfaces which may have more than one methods, so what to do if you don't need all of those?
- 2 ways:
  1. Keep **empty body** for all methods you don't need
  2. OR add **abstract** as prefix of class name

Questions?


Sukkur
IBA
Thanks…