Helwan University

HCIlab Human Computer Interaction Laboratory

Faculty of Computer Science

Interaction Group

Programming Language 2

Object oriented design using JAVA

Dr. Ayman Ezzat

Email: ayman@fcih.net

Web: www.fcih.net/ayman

Unit 6: Graphical User Interface

Outline

1. Overview of the Swing library
2. Containers and layouts
3. GUI elements
4. `JOptionPane` Class

1. Overview of the `Swing` library

- There are a number of aspects to programming an interface:
  1. The GUI components that you use (Unit 6)
  2. The way that these components are laid out on a screen (Unit 6)
  3. The way that the program that uses the interface responds to events that occur with an interface, such as a mouse being clicked or text being entered into a text field.
     - This is called **event-driven programming** (**Unit 7**).

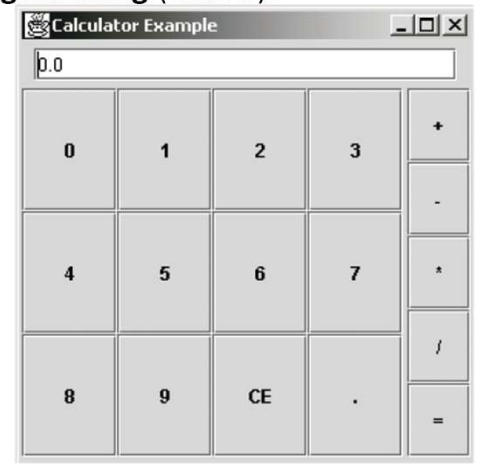


Figure 2 Calculator interface showing number and function buttons with a display for output

1. Overview of the `Swing` libraryQ

Java `Swing` library

- **`javax.swing`** contains pre-built classes of graphical interface elements.
- The Swing library is an **extension of Java's Abstract Windowing Toolkit (AWT)**.
  - The AWT is the part of the Java run-time system that is responsible for user interaction with window objects.
- The Swing library components **look the same on every platform** and so you do not have to worry about any inconsistencies between platforms.
  - **Unlike the AWT components**, Swing components are not closely tied to the underlying operating system.

- Swing can be broken down into groups of related classes:

  (1) Containers. A container is a component that can hold other components . For example:
  - **JFrame**, which provides the basic attributes of a window, and contains all of the visual components, including other containers such as JPanel.
  - **JPanel,** which group elements in an area of a window and is then added to a JFrame component.

  (2) Layout managers. These are used **with containers** to arrange embedded components into a particular layout style. There are various layout managers include **BorderLayout**, **FlowLayout, GridLayout, GridBagLayout** and **BoxLayout.**

  (3) Visual components. This group of components provides the means by which users will usually interact with your applications, e.g. buttons.

1. Overview of the `Swing` library

Java `Swing` library

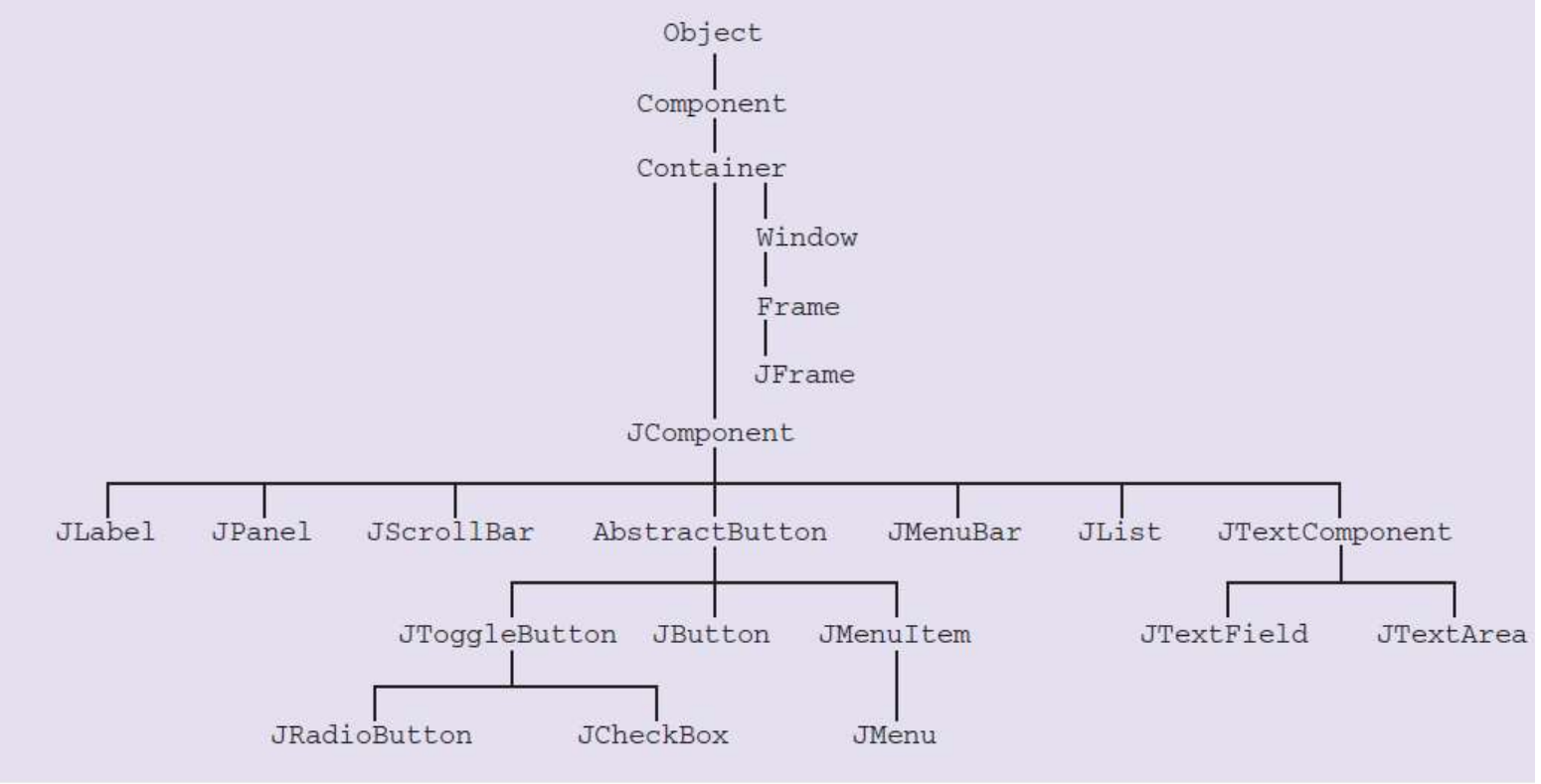


Figure 1 Part of the Swing hierarchy showing the inheritance relationships between the various Swing components and, ultimately, `Object`

2. Containers and layouts

(1) Containers

- We will start by looking at **how visual components are placed** in a component known as a **container**. In this course, we shall study:
  - **JFrame**. This is a window used in Java applications.
    - A JFrame is able to nest (contain) other components.
  - **JPanel**. This is a container that can be used in both **applications** and **applets**.
    - Panels**, like all containers, can contain both visual components and other containers**.
    - A panel is a container that can contain other containers and also GUI visual components such as buttons. Thus panels allow you to modularize the layout of the GUI.
  - **JApplet**. This is a container that can be embedded in a web page (unit 9).

2. Containers and layouts

(1) Containers (Cont'd)

- **JFrame**.
  - The actual elements within a JFrame are held in the **contentPane**. The content pane is the usable area of a frame in which other components can be placed.
  - The interface for an application is usually created by inheriting from JFrame.
  - The most commonly used JFrame methods are listed in Table1.
  - As you will see later, many of these methods are also used with JPanel

Table 1 Methods of class `JFrame`

Method signature	Description
`setLayout(LayoutManager)`	sets a particular style of layout
`add(Component)`	includes a component
`remove(Component)`	removes a component
`setVisible(boolean)`	makes the window visible or not
`setTitle(String)`	gives the window a title
`getContentPane()`	gets access to the content pane

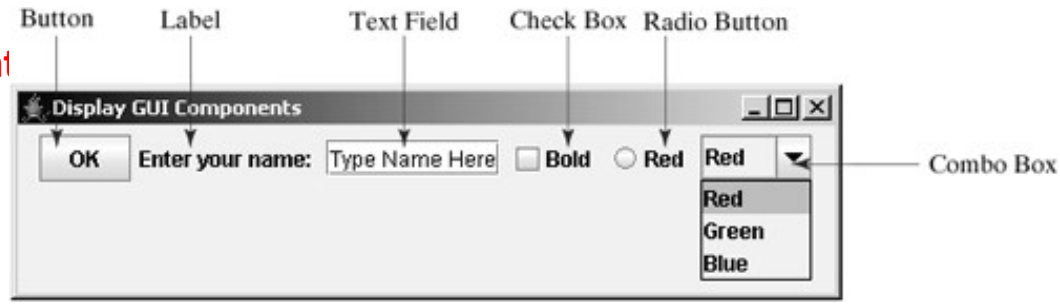
2. Containers and layouts

(2) Layout Managers

- The way in which visual components are laid out inside a container depends on the way you specify its layout.
- **Each container has a method called `setLayout`**, which allows you to specify the pattern of layout for the visual components within the container.
- Each type of container has a default pattern.
  - **JFrame** has a default pattern of **BorderLayout** .
  - **JPanel** has a default of **FlowLayout**.

2. Containers and layouts

(3) Visual component




```
// Create a button with text OK
JButton btnOK = new JButton("OK");

// Create a label with text "Enter your name: "
JLabel lblName = new JLabel("Enter your name: ");

// Create a text field with text "Type Name Here"
JTextField txtName = new JTextField("Type Name Here");

// Create a check box with text bold
JCheckBox chkBold = new JCheckBox("Bold");

// Create a radio button with text red
JRadioButton radRed = new JRadioButton("Red");

// Create a combo box with choices red, green, and blue
JComboBox cboColor = new JComboBox(new String[]{"Red", "Green", "Blue"});
```

2. Containers and layouts

(a) Using containers

- The following code is a very simple application that displays a frame and sets the title of the window:

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

public class FrameDemo extends JFrame
{
    public FrameDemo (String title)
    {
        setSize(200, 200);
        setTitle(title);
    }
}
```

To run the code we use the following class:

```
public class FrameDemoTest
{
    public static void main (String[] args)
    {
        FrameDemo fd = new FrameDemo("We Love Java");
        fd.setVisible(true);
    }
}
```

A simple frame showing only a title, the standard icon and the window buttons

2. Containers and layouts

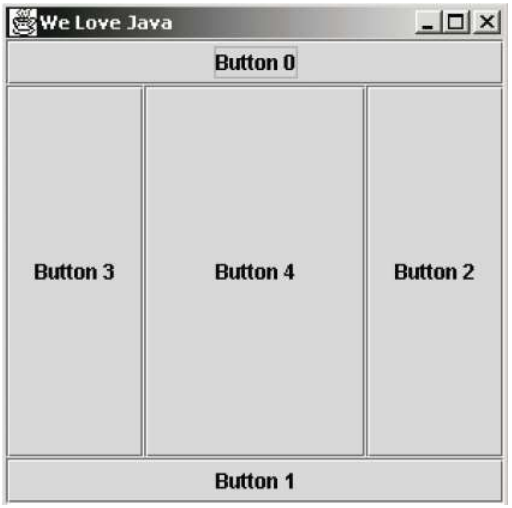
(b) Placing elements using `LayoutManager` (`BorderLayout`)

Define 'button' reference variables

Construct button objects (inside the constructor)

Add buttons to the JFrame using the BorderLayout (inside the constructor)

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

public class FrameDemo extends JFrame
{
    private final int numOfBut = 5;
    private JButton[] buttons = new JButton[numOfBut];

    public FrameDemo (String title)
    {
        setSize(300, 300);
        setTitle(title);

        /* Create and name some buttons. */
        for (int i = 0; i < numOfBut; i++)
        {
            buttons[i] = new JButton("Button " + i);
        }

        /* Place some buttons. */
        Container cp = getContentPane();
        cp.add(buttons[0], BorderLayout.NORTH);
        cp.add(buttons[1], BorderLayout.SOUTH);
        cp.add(buttons[2], BorderLayout.EAST);
        cp.add(buttons[3], BorderLayout.WEST);
        cp.add(buttons[4], BorderLayout.CENTER);
    }
```



A **Border Layout** with 5buttons

`BorderLayout`

If you don't set the layout, then the default layout is used, which is `BorderLayout` for `Jframe`.

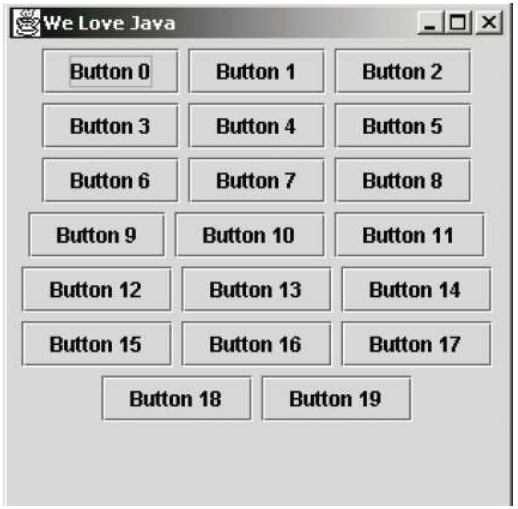
When using the `BorderLayout` each element is placed in the frame using a static constant `NORTH`, `EAST`, `SOUTH`, `WEST` and `CENTER`.

2. Containers and layouts

(c) Placing elements using `LayoutManager (FlowLayout)`

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

public class FrameDemo extends JFrame
{
    private final int numOfBut = 20;
    private JButton[] buttons = new JButton[numOfBut];

    public FrameDemo (String title)
    {
        setSize(300, 300);
        setTitle(title);

        /* Create some buttons. */
        for (int i = 0; i < numOfBut; i++)
        {
            buttons[i] = new JButton("Button " + i);
        }

        /* Place some buttons. */
        Container cp = getContentPane();
        cp.setLayout(new FlowLayout());
        for (int i = 0; i < numOfBut; i++)
        {
            cp.add(buttons[i]);
        }
    }
}
```

Add buttons to the JFrame using the FlowLayout (you don't need to set location)



A **FlowLayout** with 20 buttons

`FlowLayout`

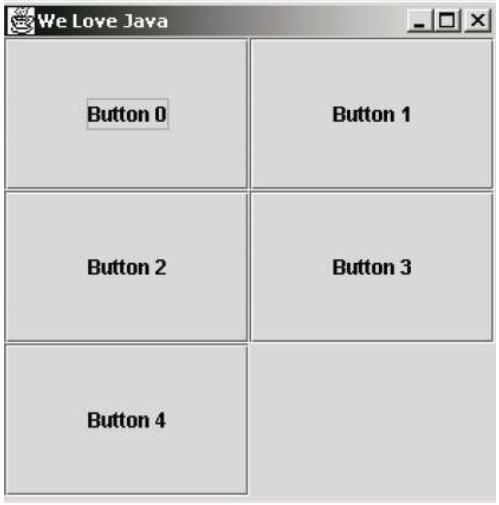
This layout places the items in the order in which they are added. If there isn't enough room on a line to include all of the components then they will flow onto the next line.

2. Containers and layouts

(c) Placing elements using `LayoutManager (GridLayout)`

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

public class FrameDemo extends JFrame
{
    private final int numOfBut = 5;
    private JButton[] buttons = new JButton[numOfBut];

    public FrameDemo (String title)
    {
        setSize(300, 300);
        setTitle(title);

        /* Create some buttons. */
        for (int i = 0; i < numOfBut; i++)
        {
            buttons[i] = new JButton("Button " + i);
        }

        /* Place some buttons. */
        Container cp = getContentPane();
        // set size of grid - rows then columns
        cp.setLayout(new GridLayout(3, 2));
        for (int i = 0; i < numOfBut; i++)
        {
            cp.add(buttons[i]);
        }
    }
}
```

Add buttons to the JFrame using the GridLayout



A **GridLayout** with 5 buttons

`GridLayout`

This layout places the elements according to a grid of rows and columns.
When you define a GridLayout object it has two arguments: the first is the number of rows and the second is the number of columns.

2. Containers and layouts

Do we always have to use Layout Managers?

- It is also possible not to use any layout manager and to place components manually. This is known as **absolute positioning**. It is done in **3 STEPS**:
  1. specifying the **layout manager** to be **null**,
  2. **adding the component** to the holder in the normal way
  3. using the `setBounds` method to place and size the component in the holder.

For example:

```
...
holder.setLayout(null);
JButton b1 = new JButton("Java!");
holder.add(b1);
b1.setBounds(10, 10, 20, 20);
...
```

Advantage: you get greater control over placement.

Disadvantage: every element has to be placed manually, rather than relying on a layout manager. This can have major ramifications when you have to change the user interface; for example, inserting a new element will mean that many of the existing elements will need to be moved.

Outline

1. Overview of the Swing library
2. Containers and layouts
3. GUI elements
4. `JOptionPane` Class

3. GUI elements

Buttons

- A button is used so that the user can ask for action from a program.
- **How to create:**

```
JButton offButton = new JButton(); // without label
JButton onButton = new JButton("Cancel"); //with label
```

- **Setting:** To set the label of a button →setText

```
offButton.setText("Press Me");
```

- **Getting:** To get the label of a button → getText

```
String offButtonLabel = offButton.getText();
```

Labels

- A label is just text that can be placed within a container.
- **How to create:**

```
JLabel lb = new JLabel("Hello there");
```

- **Setting:** To set the text on the label →setText

```
lb.setText("Good Bye");
```

- **Getting:** To get the text from a label → getText

```
String currentLabelStr = lb.getText();
```

3. GUI elements

Check boxes

- Check boxes are used for the input of boolean data.
- **How to create:**

```
JCheckBox noviceUserType = new JCheckBox("Novice");
JCheckBox expeUserType = new JCheckBox("Experienced");
```

- **Getting:** The state of a check box – whether or not it is on – can be discovered by means of the method isSelected, which returns a boolean result.

```
boolean b = expeUserType.isSelected();
```

3. GUI elements

Radio buttons

- Radio buttons allow the user to choose one of a list of choices.
  - Radio buttons are grouped together and have the property that only one of the buttons can be selected at a time.
- **How to create:**

```
// (1) Create the buttons
JRadioButton fr = new JRadioButton("French", true); //initially selected
JRadioButton en = new JRadioButton("English", false);
// (2) Group the buttons
ButtonGroup languageGroup = new ButtonGroup();
languageGroup.add(fr);
languageGroup.add(en);
```

Note:

when you add elements to the JFrame (or JPanel), you must add the RadioButton objects, not the ButtonGroup object. The aim of using a ButtonGroup object is to tell Java that a set of buttons are are grouped together and have the property that only one of the buttons can be selected at a time.

3. GUI elements

Combo boxes (drop-down lists)

- A combo box is a drop-down list that allows the programmer to specify a number of **strings**, one of which can be selected.
- **How to create:**

```
// (1) Create the ComboBox
JComboBox computerChoice = new JComboBox();
// (2) Add items to the ComboBox
computerChoice.addItem("VAX");
computerChoice.addItem("PC");
computerChoice.addItem("Mac");
computerChoice.setSelectedItem("VAX"); //initially selected
```

- **Getting:** using a method to return the string of the currently selected item.

```
String s = computerChoice.getSelectedItem();
```

This results in the string **s** being set to the string "VAX".

3. GUI elements

Lists

- A list of items where the user can select a **single** string or a **number** of strings.
- **How to create:** the simplest way to create a list is…

```
// (1) Create an array containing the choices available
String []data = {"brendan", "anton", "barbara", "martin"};
//(2) then to pass this array to the JList constructor
JList nameList = new JList(data);
```

- **Getting:** The most important methods are…

`String[] getSelectedValues()`

returns a string array that contains the names of the items that have been selected.

- Example: `String [] names = nameList.getSelectedValues();` places the strings in the data array that have been selected by the user in the string array `names`.

`int[] getSelectedIndices()`

returns an array of all the selected indices in increasing order.

`String getElementAt(int)`

returns the string that can be found at the specified index.

3. GUI elements

Text fields

- Text fields are used for the input of small items of textual data.
- **How to create:**

```
JTextField txfA = new JTextField(); // empty textbox
JTextField txfB = new JTextField(20); // empty, width = 20 characters
JTextField txfC = new JTextField("Type here"); // not empty
JTextField txfD = new JTextField("Type here", 20); // not empty, width=20
```

- **Setting:** `txfA.setText("Type here");`
- **Getting:** `String t = txfA.getText();`

3. GUI elements

Text areas

- A text area is used for entering comparatively large amounts of textual data.
  - containing a number of lines of text, rather than the single line found in a text field.
  - Both the class JTextArea and the class JTextField inherit from a class known as JTextComponent.
- **How to create:**

```
JTextArea ta = new JTextArea (4, 20); // empty, 4 rows, 20 columns
JTextArea ta = new JTextArea ("line1\nline2", 4, 20);//not empty, 4 r, 20 c
```

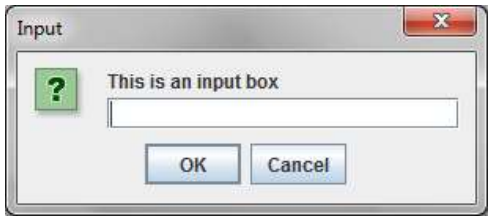
- **Setting & Getting**
  - same as textbox (setText(), getText())

Outline

1. Overview of the Swing library
2. Containers and layouts
3. GUI elements
4. `JOptionPane` Class

4. JOptionPane Class

- **JOptionPane** makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something.
- The JOptionPane class is part of the **javax.swing** library.
- Three of the dialog boxes are:
  - **Message Dialog** - Displays a message (left figure).
  - **Input Dialog** - Prompt for some input. (middle figure).
  - **Confirm Dialog** – Asks a confirming question, like yes/no/cancel (right figure)



Some JOptionPane methods to use the above dialogues are:

```
showMessageDialog(Component parentComponent, Object message)
showInputDialog(Object message)
showConfirmDialog(Component parentComponent, Object message)
showConfirmDialog(Component parentComponent, Object message, String title, int optionType)
```

4. `JOptionPane` Class

The parameters to the **`JOptionPane`**'s methods follow consistent patterns:

- **`parentComponent`**: Defines the Component that is to be the parent of this dialog box. It is used in two ways:
  1. the Frame that contains it is used as the Frame parent for the dialog box, and its screen coordinates are used in the placement of the dialog box.
  2. This parameter may be **`null`**, in which case a default Frame is used as the parent, and the dialog will be centered on the screen.
- **`Message`**: A descriptive message to be placed in the dialog box. In the most common usage, message is just a String.
- **`title`**: The title for the dialog box.
- **`Option type`**, one of DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION or OK_CANCEL_OPTION.

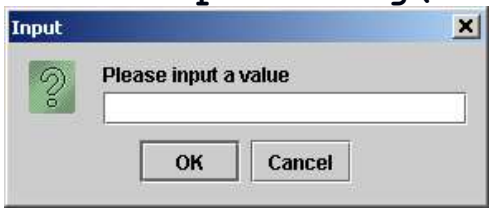
4. JOptionPane Class

Examples

```
JOptionPane.showMessageDialog(null, "Hello World");
```


```
JOptionPane.showMessageDialog(null, "alert", "alert",
                              JOptionPane.ERROR_MESSAGE);
```


```
String s = JOptionPane.showInputDialog("Please input a value");
```



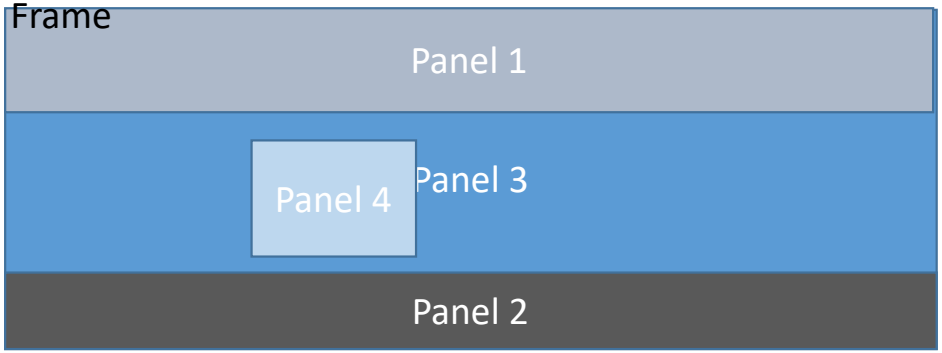
4. JOptionPane Class

Examples

```
import javax.swing.JOptionPane;
public class Test {
    public static void main(String[] args) {
        String firstName;  // The user's first name
        String lastName;   // The user's last name
        // Get the user's name
        firstName = JOptionPane.showInputDialog("What is your first name? ");
        lastName = JOptionPane.showInputDialog("What is " + "your last name? ");
        // Display a greeting
        JOptionPane.showMessageDialog(null, "Hello " + firstName + " " + lastName);

    }
}
```

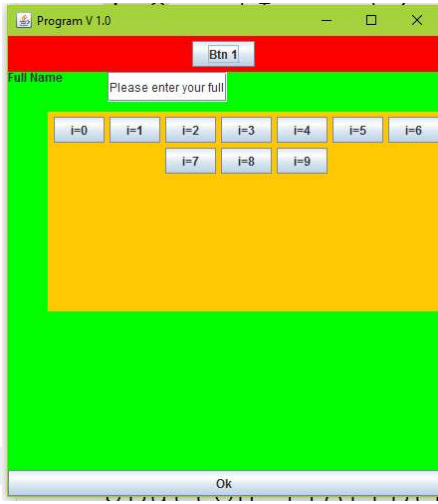
Nested Panels

- A frame could Contain Many Panels




```
JPanel jp1=new JPanel(new FlowLayout());
JPanel jp2=new JPanel (new BorderLayout());
JPanel jp3=new JPanel();
JPanel jp4=new JPanel();
jp3.setLayout(null);
jp3.setBackground(Color.GREEN);
jp4.setBackground(Color.ORANGE);
JLabel lblFullName=new JLabel("Full Name");
lblFullName.setBounds(0,0,100,10);
jp3.add(lblFullName);
JTextField txtFullName=new JTextField("Please enter
txtFullName.setBounds(lblFullName.getBounds().width,
jp3.add(txtFullName);
jp4.setBounds(40,40, 400, 200);
JButton []ArrBtn=new JButton[10];
```

```
for (int i=0;i<10;i++)
{
    ArrBtn[i]=new JButton("i="+i);
    jp4.add(ArrBtn[i]);
}
jp3.add(jp4);
jp1.setBackground(Color.red);
this.add(jp1,BorderLayout.NORTH);
this.add(jp2,BorderLayout.SOUTH);
this.add(jp3,BorderLayout.CENTER);
jp2.setBackground(Color.BLUE);
jp1.add(new JButton("Btn 1"));
jp2.add(new JButton("Ok"), BorderLayout.CENTER);
```



Add Image to Labels and resize

```
JFileChooser fc = new JFileChooser();
if(fc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
BufferedImage img = ImageIO.read(fc.getSelectedFile());//
JLabel label = new JLabel();

label.setIcon(new ImageIcon(img));
this.getContentPane().add(label);
}
```

- JLabel Label2=new JLabel();
- Label2.setIcon(new ImageIcon(new ImageIcon("d:\\MIU.png").getImage().getScaledInstance(150, 150, Image.SCALE_DEFAULT)));
- this.getContentPane().add(Label2);

Visual programming

Visual programming approach

- Many IDEs provide the facilities to create user interfaces using a 'drag and drop‘ approach from a palette of components.
  - The IDE automatically creates the code necessary to produce the interface that you have visually created. The IDE then allows you to adjust the parameters of the components and to add the code necessary to make them work.
  - However, creating layouts in this way can produce code that is unnecessarily complex and may make the application less portable.

- While this course will not make use of them, we would draw your attention to these features in order that you can explore them independently outside of the course.