UNIT 2. GUI- Components- 1

- ## Introduction:

INTRODUCTION TO GUI PROGRAMMING

GUI programming involves the use of a number of predefined components such as buttons, checkboxes, text fields, windows, menus, etc that are part of the class hierarchy. A GUI is a collection of instances of these components.

In Java, there are two sets of GUI components. Prior to Java version 1.2, Java GUIs were build with components from the Abstract Windows Toolkit (AWT). The AWT components are contained within the java.awt package. Java applications using AWT display differently on each platform.

With Java 1.2, a more flexible Swing API was introduced. Swing GUI components allow the designer to specify a uniform look and feel for a GUI application across all platforms, or to use each platforms custom look and feel. Although we will focus on Swing components, Swing and AWT are closely related, because Swing relies on features of the AWT. The principles for using both components are similar.

Using a Graphical User Interface, when you perform an action, an event is generated. In event-driven programming, the program responds to the events.

The GUI programs that we will create in Java will contain three types of software. These are:

1. **Graphical components** that make up the GUI
2. **Listener classes/methods** that receive the events and respond to them, and
3. **Application methods** that do the work for the user.

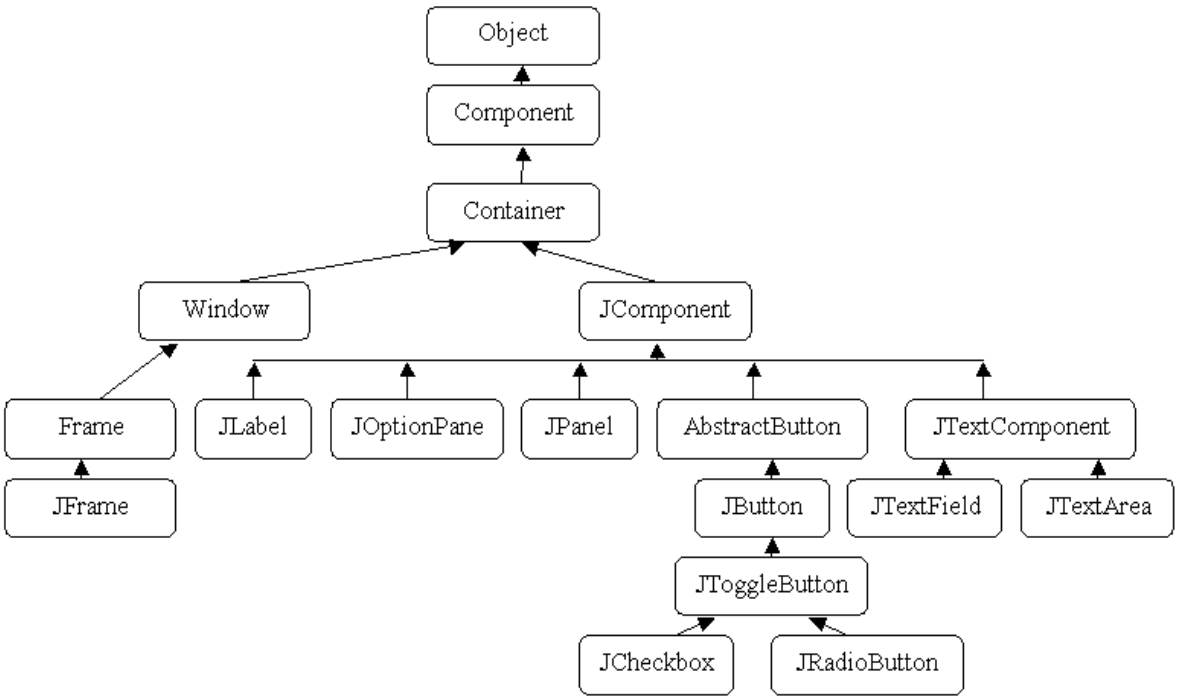
The graphical components are those like the ones listed above. You can use them as they are, or extend them into your own classes. Listener methods are the methods in your application that respond to different events. Listener methods invoke application methods. An application method is a standard Java method to perform useful tasks. In Java applications, generally these three types of software should be kept separate (reduce coupling).

Graphical Components

A GUI program consists of a collection of graphical components that are placed inside one or more windows/applets/panes. Most components are “contained” within a window. Therefore, the window acts as a container to hold various GUI components. A container is an area on the screen that contains smaller areas. I.e. the window is a container, which contains components such as buttons, menus, scroll bars etc.

Remember however, that all components belong to the Java class hierarchy. The GUI hierarchy is shown below.

Part of the AWT and Swing Hierarchy



As you can see, the class **Component** is a subclass of the class Object, which declares many attributes and behaviours common to the GUI components in package java.awt and javax.swing.

The class **Component** implements some useful public methods, which are inherited by its subclasses, such as:

- public void setSize(int width, int height) – used to set the size of a component
- public void setBackground(Color c) - used to set the background color of a component
- public void setVisible(boolean b) - used to set the visibility of a component (i.e. to make a component appear/display component on the screen)

The class **Container** is a subclass of **Component**. Components are attached to Containers (such as windows) in order for components to be organized and displayed on the screen. Any object that is a Container can be used to organize other components in a GUI. Because a Container is-a Component, you can attach Containers to other Containers. In other words, a Container is a component that can hold other

UNIT 2. GUI- Components- 1

Containers.

There are two types of **Containers**, panels and windows. Windows are the top level element displayed by an operating system. Panels are containers that cluster related components within a larger object, such as an applet, or application window.

The Container class is an ***abstract class*** which implements some useful methods, such as:

- public Component add(Component) – used to add components to a container
- public setLayout(LayoutManager mgr) – used to set the Layout of a container.

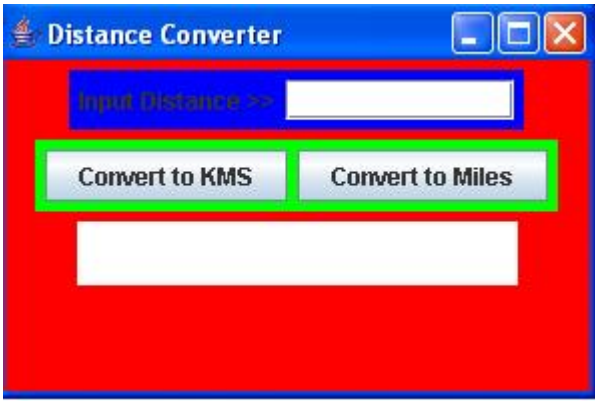
The class JComponent is a subclass of Container. JComponent is the superclass of all lightweight Swing components and declares their common attributes and behaviours. Because a JComponent is a Container, all Swing components are also Containers.

Creating a GUI

Generally a GUI **component** is an object that represents a screen element that is used to display information or to allow the user to interact with the program in a certain way. Java components include labels, buttons, text fields, scroll bars, menus etc.

A **container** is a special type of component that is used to hold and organize other components. Frames and Panels are examples of Java containers.

When building a GUI, each GUI component must be attached to a container, such as a window created with a JFrame. You also must decide where to position each GUI component within the window. This is called specifying the layout of the GUI components.



Generally, the GUI applications we create will appear inside a window. What we may refer to as a window, in Java is referred to as a frame. A frame is a window with borders, standard buttons, and other built-in features.
Above is an example of a Java frame (JFrame). The GUI components are added to the

JFrame window, and positioned inside the window using the layout manager. The area below the menu bar in the window is called the content pane. It is generally to the content pane that we add the components.

Examples: Gui.java, GuiApp.java, MessageWindow.java, MessageWindowApp.java LabelFrame.java, LabelFrameApp.java

- ## AWT and Swing (Lightweight Vs Heavyweight):

Difference between AWT and Swing with Comparison

Java programmers are often confused with respect to the difference between AWT and Swing components, the features of AWT and Swing, and the functionalities of both. While Swing is a collection of program components that possess the ability of developing **graphical user interface (GUI)** objects independent of the platform being used, AWT components are platform-dependent and perform differently on varying platforms.

An important interview question as well, the difference between AWT and Swing provides programmers with the ability of using and implementing GUI in better ways. As this article progresses, readers would understand the definition of Swing and AWT as well as the key differences between the two in the form of a comparison chart and points alike. **Read on for a closer look** at the nature, characteristics, features, functionality and differences between swing and AWT in java.

AWT vs Swing Comparison Chart

The main differences between Swing and AWT can be best understood with the help of the following comparison chart.

Basis of Differentiation	Java AWT	Java Swing
Platform dependency	AWT components are dependent on the nature of the platform.	Java swing components are not dependent on the nature of the platform. They are purely scripted in Java.
Weightiness	Heavyweight in character.	Mostly lightweight in character, swing components are designed to sit atop AWT components and perform their tasks. These components are generally lightweight as they do not need any native OS object for implementing their functionality. JFrame and Jdialog are considered to be heavyweight as they

Basis of Differentiation	Java AWT	Java Swing
		possess a peer. Therefore, components such as Jbutton, JtextArea, etc. are considered to be lightweight as they do not possess any OS peer.
Pluggable feel and looks	AWT components are non-supportive of the pluggable feel and look.	Java swing components offer support to the pluggable feel and look. Java swing API contains classes such as Jbutton, JtextField, JradioButton, Jcheckbox, JtextArea, Jmenu, JcolorChooser etc. that add further functionality to the objects being drawn.
Number of components	AWT components are less in number in comparison to Swing components.	Java Swing components are very powerful and much more in number. They are represented by lists, scroll panes, tables, tabbed panes, colour choosers, etc.
MVC (Model View Controller)	The **MVC**(Model View Controller) model is not followed by AWT components wherein the data is represented by the model, presentation represents the view and the controller serves as the bridging interface between the view and model.	Swing components in Java are known to follow the **MVC** (Model View Controller) model.
Stands for	The AWT full form is Abstract windows toolkit.	Swing components in Java are also referred to as JFC's (Java Foundation classes).
Package required	AWT components need the	Swing components in Java need the javax.swing package.

Basis of Differentiation	Java AWT	Java Swing
	java.awt package.	
Advanced features	The advanced features depicted by Swing components are not present in AWT components.	Swing components depict several advanced features such as Jtabel, Jtabbed pane, etc. These advanced features are specific only to Swing components.
Presence of peers	There are 21 **"peers"** in AWT components. There exists one peer for each control and there is one peer for the dialog itself. Peers are in the form of widgets offered by the operating system. They may be a button object or a specific entry field object.	Only one peer is present in Swing in the form of the operating system's window object. The entry fields, buttons, etc. are developed by the Swing package directly onto the drawing surface offered by the window object. Swing possesses more codes because of this very reason. The button as well as the other controls have to be drawn and their behaviour has to be implemented separately. The host operating system cannot be relied upon for the purpose of performing these functions.
Functionality	AWT components serve as a thin layer of coding that lies atop the OS.	Swing components are much larger in size and offer higher levels of functionality.
Implementation	Many features/functions of AWT components have to be implemented by the coder.	Swing components provide in-built functions for their performance.

Basis of Differentiation	Java AWT	Java Swing
Memory space	AWT components require and occupy larger memory space.	Swing components do not occupy as much memory space as awt components.
Speed of execution	AWT components are slower than swing components in terms of performance.	Swing in Java is faster than AWT in Java.

What is AWT in JAVA

Abstract Window Toolkit refers to a collection of **application program interfaces (API s)** that are utilized by Java programmers for the creation of **graphical user interface (GUI)** objects. These objects are in the form of scroll bars, windows, buttons, etc. This toolkit is an important component of **Java Foundation Classes (JFC)** belonging to Sun Microsystems, which is the company responsible for the origin of Java.

Simple Java AWT Example

```
Copy Code//Swaping two Number using Java AWT
package swap;
import java.awt.*;
import java.awt.event.*;

public class swap
{
   public static void main(String args[])
   {
      Frame f = new Frame("My Frame");

      Label l1 = new Label ("First");
      Label l2 = new Label ("Second");

      TextField t1 = new TextField(20);
      TextField t2 = new TextField(20);

      Button b = new Button("OK");
```

```
        f.add(l1);
        f.add(t1);
        f.add(l2);
        f.add(t2);
        f.add(b);

        b.addActionListener(new ActionListener () {
           public void actionPerformed(ActionEvent ae)
           {
              String temp = t1.getText();
              t1.setText(t2.getText());
              t2.setText(temp);
           }
        });

        f.layout(new FlowLayout());
        f.setSize(300,300);
        f.setVisible(true);
     }
}
```

What is Swing in JAVA

Swing in Java refers to graphical user interface (GUI) in the form of a lightweight widget toolkit; the toolkit is packaged with widgets with rich functionality. Swing forms an important part of the **Java Foundation Classes (JFC)** and provides coders with several useful packages for the development of rich desktop-friendly applications in Java. The built-in controls in Swing comprise of image buttons, trees, tabbed panes, colour choosers, sliders, toolbars, tables, etc. Swing also enables the creation of text areas for displaying rich text format **(RTF) or HTTP objects**. Swing components are written purely in java and are therefore independent of the platform they work on.

Simple Java Swing Example

```
//SWAPing

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

public class swap implements ActionListener{
   JTextField tf1,tf2;
   JButton b1;
   swap(){
      JFrame f= new JFrame();
      tf1=new JTextField();
      tf1.setBounds(50,50,150,20);
```

```
    tf2=new JTextField();
    tf2.setBounds(50,100,150,20);
    b1=new JButton("Ok");
    b1.setBounds(50,200,50,50);
    b1.addActionListener(this);
    f.add(tf1);f.add(tf2);f.add(b1);
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
  }
   public void actionPerformed(ActionEvent e) {
    String temp = t1.getText();
    t1.setText(t2.getText());
    t2.setText(temp);
  }
  public static void main(String[] args) {
    new swap();
  } }
```

Key difference between Swing and AWT

An understanding of the key differences between Java AWT and Swing go a long way in helping coders use the most appropriate components as and when needed.

1. Swing presents a more-or-less Java GUI. Swing components utilize AWT for creating an operating system window. Thereafter, Swing draws buttons, labels, checkboxes, text boxes, etc. directly on to the window. These Swing objects are designed to respond to key entries and mouse-clicks alike. Here, users can decide upon what they would like to do rather than allow the OS to handle the features and functionalities to be put to use. On the other hand, AWT being a cross-platform interface, uses the OS or native GUI toolkit in order to enable its functionality.

2. An important difference between AWT and Swing is that **AWT is heavyweight** while **Swing** components are**lightweight**.

3. In case of **AWT components**, native codes are utilized for the generation of view component as provided by the OS. This is the reason behind the look and feel of AWT components changing from one platform to another. In comparison, in case of Swing components in Java, the JVM is responsible for generating the view for components.

- **Overview of Swing Components**:

UNIT 2. GUI- Components- 1

Java Swing Components and Containers

A component is an independent visual control. Swing Framework contains a large set of components which provide rich functionalities and allow high level of customization. They all are derived from JComponent class. All these components are lightweight components. This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.

A container holds a group of components. It provides a space where a component can be managed and displayed. Containers are of two types:

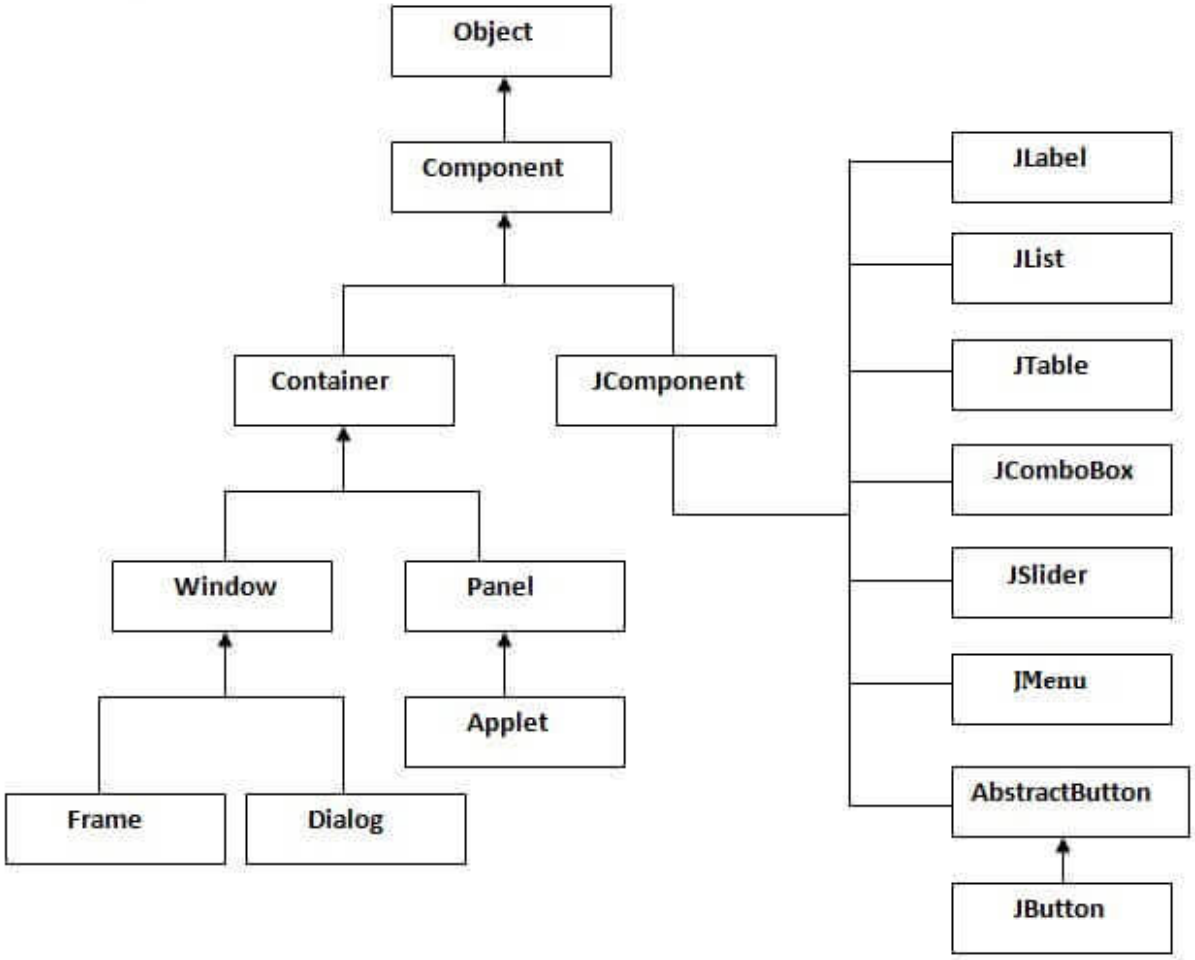
1. **Top level Containers**
    - It inherits Component and Container of AWT.
    - It cannot be contained within other containers.
    - Heavyweight.
    - Example: JFrame, JDialog, JApplet
2. **Lightweight Containers**
    - It inherits JComponent class.
    - It is a general purpose container.
    - It can be used to organize related components together.
    - Example: JPanel

What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.

public void setVisible(boolean b)	sets the visibility of the component. It is by d

Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
1. import javax.swing.*;
2. public class FirstSwingExample {
3. public static void main(String[] args) {
4. JFrame f=new JFrame();//creating instance of JFrame
5.
6. JButton b=new JButton("click");//creating instance of JButton
7. b.setBounds(130,100,100, 40);//x axis, y axis, width, height
8.
9. f.add(b);//adding button in JFrame
10.
11. f.setSize(400,500);//400 width and 500 height
12. f.setLayout(null);//using no layout managers
13. f.setVisible(true);//making the frame visible
14. }
15. }
```

- ## Simple GUI-based Input /Output with JOptionPane:

Simple GUI-Based Input/Output with JOptionPane

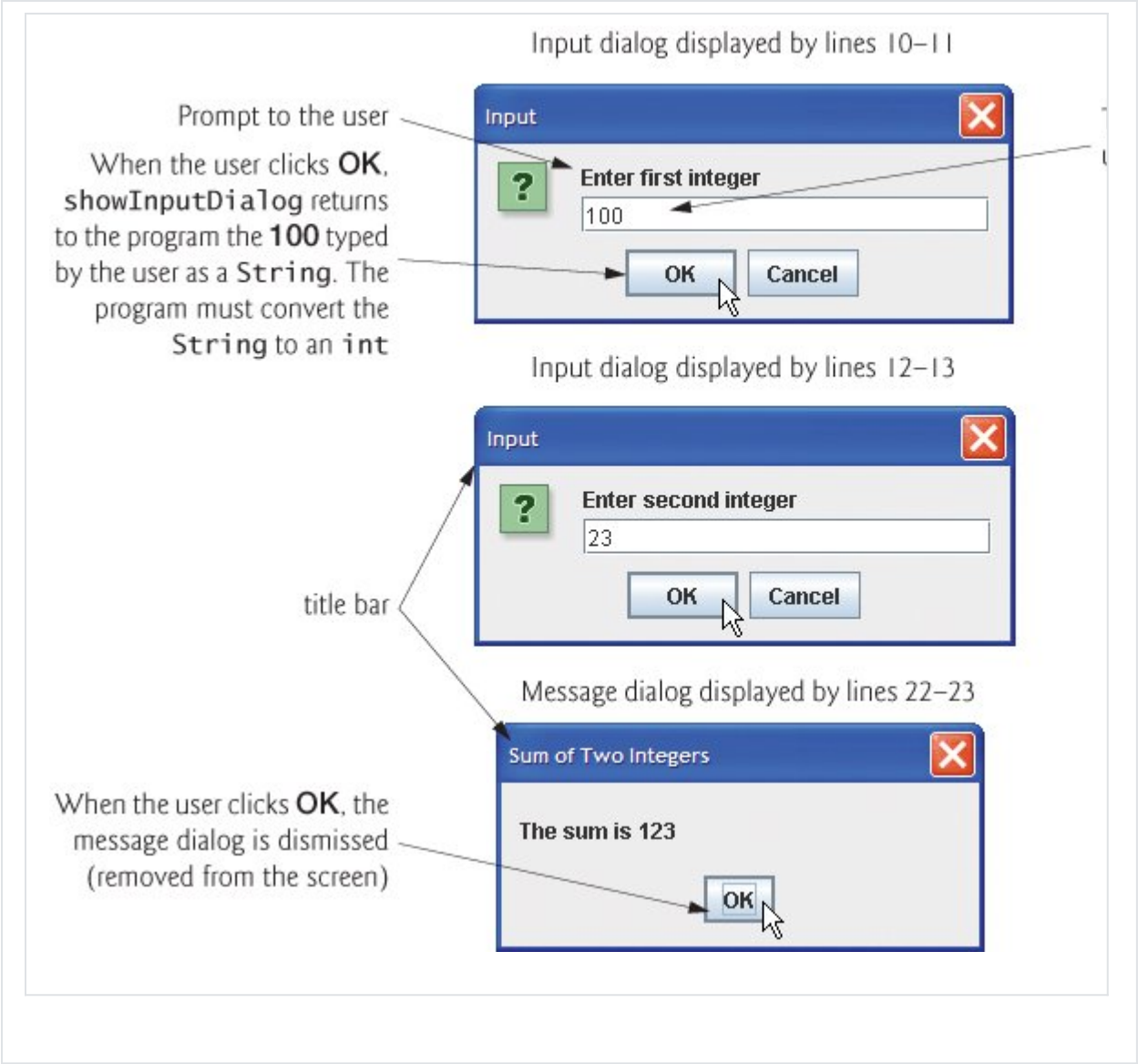
The applications in display text at the command window and obtain input from the command window. Most applications you use on a daily basis use windows or **dialog boxes** (also called dialogs) to interact with the user. For example, e-mail programs allow you to type and read messages in a window provided by the e-mail program. Typically, dialog boxes are windows in which programs display important messages to the user or

obtain information from the user. Java's **JOptionPane** class (package javax.swing) provides prepackaged dialog boxes for both input and output. These dialogs are displayed by invoking static JOptionPane methods. Figure 11.2 presents a simple addition application that uses two **input dialogs** to obtain integers from the user and a **message dialog** to display the sum of the integers the user enters.

Figure 11.2. JOptionPane input and message dialogs used to input values from the user and display results.

(This item is displayed on page 513 in the print version)

```
1  // Fig. 11.2: Addition.java
2  // Addition program that uses JOptionPane for input and output.
3  import javax.swing.JOptionPane; // program uses JOptionPane
4
5  public class Addition
6  {
7     public static void main( String args[] )
8     {
9        // obtain user input from JOptionPane input dialogs
10       String firstNumber =
11          JOptionPane.showInputDialog( "Enter first integer" );
12       String secondNumber =
13          JOptionPane.showInputDialog( "Enter second integer" );
14
15       // convert String inputs to int values for use in a calculation
16       int number1 = Integer.parseInt( firstNumber );
17       int number2 = Integer.parseInt( secondNumber );
18
19       int sum = number1 + number2; // add numbers
20
21       // display result in a JOptionPane message dialog
22       JOptionPane.showMessageDialog( null, "The sum is " + sum,
23          "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE );
24    } // end method main
25 } // end class Addition
```



Line 3 imports class JOptionPane for use in this application. Lines 10,11 declare the local String variable firstNumber and assign it the result of the call JOptionPane static method **showInputDialog**. This method displays an input dialog (see the first screen capture in Fig. 11.2), using the method's String argument ("Enter first integer") as a prompt to the user.

The user types characters in the text field, then clicks the OK button or presses the **Enter** key to submit the String to the program.
Clicking OK also dismisses (hides) the dialog. [**Note:** If you type in the text field and nothing appears, activate the text field by clicking it with the mouse.] Unlike Scanner, which can be used to input values of several types from the user at the keyboard, an input dialog can input only input Strings. This is typical of most GUI components. Technically, the user can type anything in the input dialog's text field. Our program assumes that the user enters a valid integer value. If the user clicks the Cancel button, showInputDialog returns null. If the user either types a noninteger

value or clicks the Cancel button in the input dialog, a runtime logic error will occur in this program and it will not operate correctly. Exception Handling, discusses how to handle such errors. Lines 12,13 display another input dialog that prompts the user to enter the second integer.

To perform the calculation in this application, we must convert the Strings that the user entered to int values. Recall from that
the Integer class's static method parseInt converts its Stringargument to an int value. Lines 16,17 assign the converted values to local variables number1 and number2. Then, line 19 sums these values and assigns the result to local variable sum.

Lines 22,23 use JOptionPane static method **showMessageDialog** to display a message dialog (the last screen capture of Fig. 11.2) containing the sum. The first argument helps the Java application determine where to position the dialog box. The value null indicates that the dialog should appear in the center of the computer screen. The first argument can also be used to specify that the dialog should appear centered over a particular window, which we will demonstrate later in . The second argument is the message to displayin this case, the result of concatenating the String "The sum is "and the value of sum. The third argument"Sum of Two Integers"represents the string that should appear in the dialog's title bar at the top of the dialog. The fourth argumentJOptionPane.PLAIN_MESSAGEis the type of message dialog to display. A PLAIN_MESSAGE dialog does not display an icon to the left of the message. Class JOptionPane provides several overloaded versions of methods showInputDialog and showMessageDialog, as well as methods that display other dialog types.

- ## Displaying Text and Images in a window:

JLabel | Java Swing

JLabel is a class of java Swing . JLabel is used to display a short string or an image icon. JLabel can display text, image or both . JLabel is only a display of text or image and it cannot get focus . JLabel is inactive to input events such a mouse focus or keyboard focus. By default labels are vertically centered but the user can change the alignment of

label.

Constructor of the class are :

1. **JLabel() :** creates a blank label with no text or image in it.
2. **JLabel(String s) :** creates a new label with the string specified.
3. **JLabel(Icon i) :** creates a new label with a image on it.
4. **JLabel(String s, Icon i, int align) :** creates a new label with a string, an image and a specified horizontal alignment

Commonly used methods of the class are :

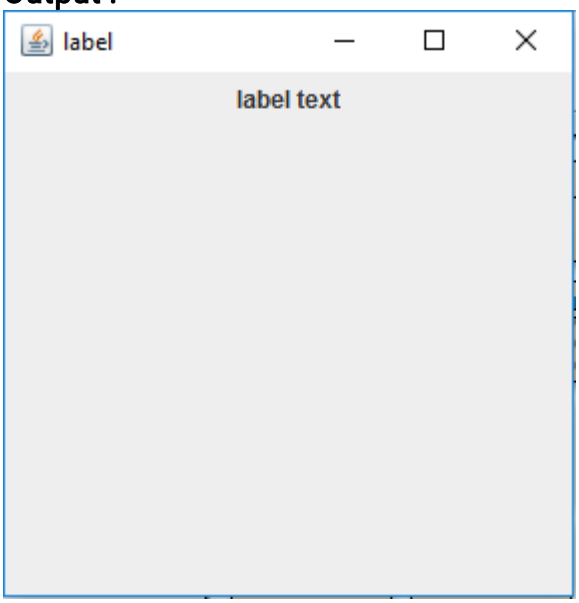
1. **getIcon() :** returns the image that that the label displays
2. **setIcon(Icon i) :** sets the icon that the label will display to image i
3. **getText() :** returns the text that the label will display
4. **setText(String s) :** sets the text that the label will display to string s

1. Program to create a blank label and add text to it.

```
// Java Program to create a

// blank label and add text to it.

import java.awt.event.*;

import java.awt.*;

import javax.swing.*;

class text extends JFrame {


   // frame

   static JFrame f;


   // label to diaplay text

   static JLabel l;


   // default constructor

   text()

   {

   }


   // main class
```

```
    public static void main(String[] args)
    {
        // create a new frame to stor text field and button
        f = new JFrame("label");

        // create a label to display text
        l = new JLabel();

        // add text to label
        l.setText("label text");

        // create a panel
        JPanel p = new JPanel();

        // add label to panel
        p.add(l);

        // add panel to frame
        f.add(p);

        // set the size of frame
        f.setSize(300, 300);

        f.show();
    }
}
```

Output :

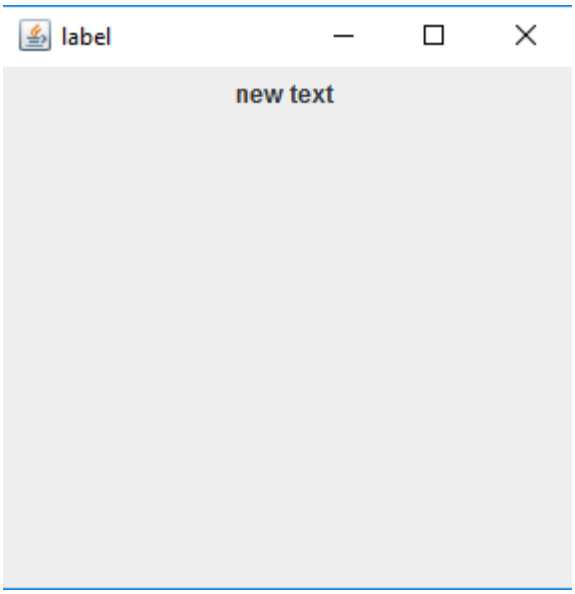


2. Program to create a new label using constructor – JLabel(String s)

```
// Java Program to create a new label
// using constructor - JLabel(String s)
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class text extends JFrame {

    // frame
    static JFrame f;

    // label to diaplay text
    static JLabel l;

    // default constructor
    text()
    {
```

```
    }

    // main class
    public static void main(String[] args)
    {
        // create a new frame to stor text field and button
        f = new JFrame("label");

        // create a label to display text
        l = new JLabel("new text ");

        // create a panel
        JPanel p = new JPanel();

        // add label to panel
        p.add(l);

        // add panel to frame
        f.add(p);

        // set the size of frame
        f.setSize(300, 300);

        f.show();
    }
}
```

Output :



3. Program to create a label and add image to it .

```
// Java Program to create  a label
// and add image to it .
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class text extends JFrame {

    // frame
    static JFrame f;

    // label to diaplay text
    static JLabel l;

    // default constructor
    text()
    {
    }
```

```
    // main class
    public static void main(String[] args)
    {
        // create a new frame to stor text field and button
        f = new JFrame("label");

        // create a new image icon
        ImageIcon i = new ImageIcon("f:/image.png");

        // create a label to display image
        l = new JLabel(i);

        // create a panel
        JPanel p = new JPanel();

        // add label to panel
        p.add(l);

        // add panel to frame
        f.add(p);

        // set the size of frame
        f.setSize(500, 500);

        f.show();
    }
}
```

Output :

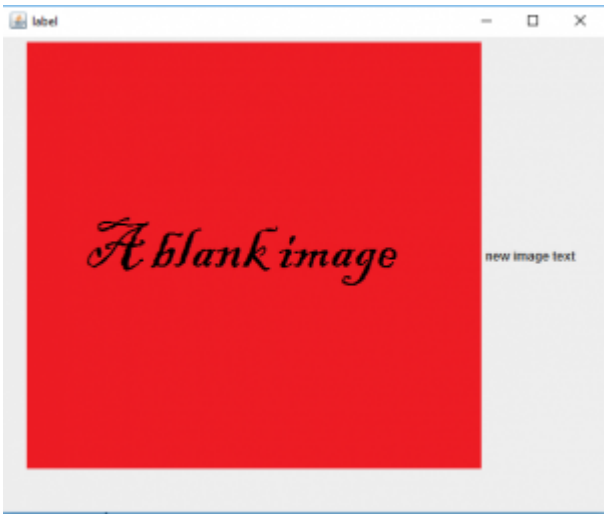
4.Program to add a image and string to a label

```
// Java Program to add a image and string
// to a label with horizontal alignment
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class text extends JFrame {

    // frame
    static JFrame f;

    // label to diaplay text
    static JLabel l;

    // default constructor
    text()
```

```
{
}

// main class
public static void main(String[] args)
{
    // create a new frame to stor text field and button
    f = new JFrame("label");

    // create a new image icon
    ImageIcon i = new ImageIcon("f:/image.png");

    // create a label to display text and image
    l = new JLabel("new image text ", i, SwingConstants.HORIZONTAL);

    // create a panel
    JPanel p = new JPanel();

    // add label to panel
    p.add(l);

    // add panel to frame
    f.add(p);

    // set the size of frame
    f.setSize(600, 500);

    f.show();
```

```
    }
}
```

Output:



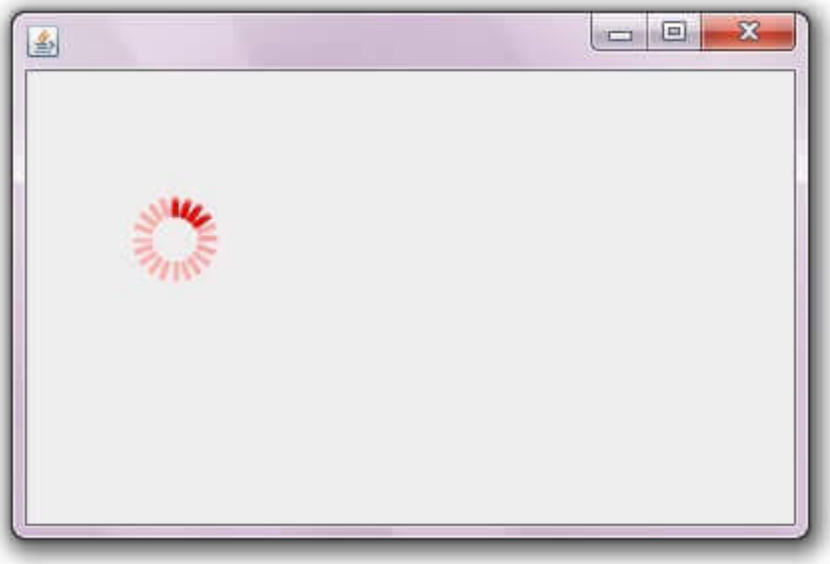
Note : This programs might not run in an online compiler please use an offline IDE.

Displaying image in swing:

For displaying image, we can use the method drawImage() of Graphics class.

Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is use image.



Example of displaying image in swing:

```
1.  import java.awt.*;
2.  import javax.swing.JFrame;
3.
4.  public class MyCanvas extends Canvas{
5.
6.     public void paint(Graphics g) {
7.
8.        Toolkit t=Toolkit.getDefaultToolkit();
9.        Image i=t.getImage("p3.gif");
10.       g.drawImage(i, 120,100,this);
11.
12.   }
13.       public static void main(String[] args) {
14.       MyCanvas m=new MyCanvas();
15.       JFrame f=new JFrame();
16.       f.add(m);
17.       f.setSize(400,400);
18.       f.setVisible(true);
19.   }
20.
21.}
```

- ## Introduction to Event Handling:

What is an Event?

Change in the state of an object is known as **Event**, i.e., event describes the change in the state of the source. Events are generated as a result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from the list, and scrolling the page are the activities that causes an event to occur.

Types of Event

The events can be broadly classified into two categories −

- **Foreground Events** − These events require direct interaction of the user. They are generated as consequences of a person interacting with the graphical components in the Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.
- **Background Events** − These events require the interaction of the end user. Operating system interrupts, hardware or software failure, timer expiration, and operation completion are some examples of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler, that is executed when an event occurs.

Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The Delegation Event Model has the following key participants.

- **Source** − The source is an object on which the event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide us with classes for the source object.
- **Listener** − It is also known as event handler. The listener is responsible for generating a response to an event. From the point of view of Java implementation, the listener is also an object. The listener waits till it receives an

event. Once the event is received, the listener processes the event and then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to a separate piece of code.

In this model, the listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listeners who want to receive them.

Steps Involved in Event Handling

Step 1 – The user clicks the button and the event is generated.

Step 2 – The object of concerned event class is created automatically and information about the source and the event get populated within the same object.

Step 3 – Event object is forwarded to the method of the registered listener class.

Step 4 – The method is gets executed and returns.

Points to Remember About the Listener

- In order to design a listener class, you have to develop some listener interfaces. These Listener interfaces forecast some public abstract callback methods, which must be implemented by the listener class.
- If you do not implement any of the predefined interfaces, then your class cannot act as a listener class for a source object.

Callback Methods

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represent an event method. In response to an event, java jre will fire callback method. All such callback methods are provided in listener interfaces.

If a component wants some listener to listen ot its events, the source must register itself to the listener.

Event Handling Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {
  private JFrame mainFrame;
  private JLabel headerLabel;
  private JLabel statusLabel;
  private JPanel controlPanel;

  public SwingControlDemo(){
    prepareGUI();
  }
  public static void main(String[] args){
    SwingControlDemo swingControlDemo = new SwingControlDemo();
    swingControlDemo.showEventDemo();
  }
  private void prepareGUI(){
    mainFrame = new JFrame("Java SWING Examples");
    mainFrame.setSize(400,400);
    mainFrame.setLayout(new GridLayout(3, 1));

    headerLabel = new JLabel("",JLabel.CENTER );
    statusLabel = new JLabel("",JLabel.CENTER);
    statusLabel.setSize(350,100);

    mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
        System.exit(0);
      }
    });
    controlPanel = new JPanel();
    controlPanel.setLayout(new FlowLayout());

    mainFrame.add(headerLabel);
    mainFrame.add(controlPanel);
    mainFrame.add(statusLabel);
    mainFrame.setVisible(true);
  }
  private void showEventDemo(){
    headerLabel.setText("Control in action: Button");

    JButton okButton = new JButton("OK");
    JButton submitButton = new JButton("Submit");
```

```
    JButton cancelButton = new JButton("Cancel");

    okButton.setActionCommand("OK");
    submitButton.setActionCommand("Submit");
    cancelButton.setActionCommand("Cancel");

    okButton.addActionListener(new ButtonClickListener());
    submitButton.addActionListener(new ButtonClickListener());
    cancelButton.addActionListener(new ButtonClickListener());

    controlPanel.add(okButton);
    controlPanel.add(submitButton);
    controlPanel.add(cancelButton);

    mainFrame.setVisible(true);
  }
  private class ButtonClickListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
      String command = e.getActionCommand();

      if( command.equals( "OK" ))  {
        statusLabel.setText("Ok Button clicked.");
      } else if( command.equals( "Submit" ) )  {
        statusLabel.setText("Submit Button clicked.");
      } else {
        statusLabel.setText("Cancel Button clicked.");
      }
    }
  }
}
```

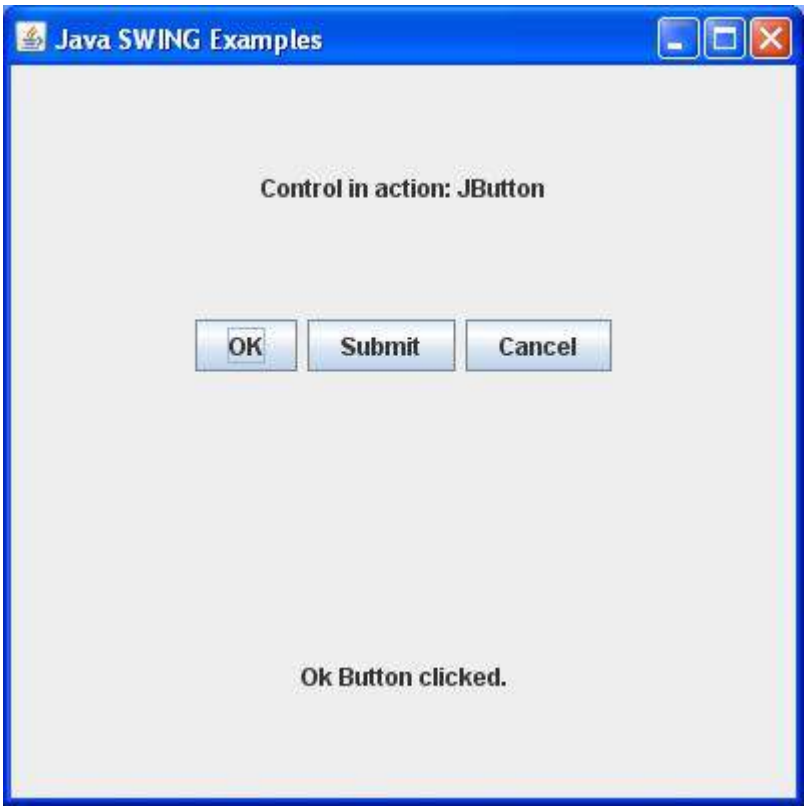
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

D:\AWT>javac com\tutorialspoint\gui\SwingControlDemo.java

If no error occurs, it means the compilation is successful. Run the program using the following command.

D:\AWT>java com.tutorialspoint.gui.SwingControlDemo

Verify the following output.



- ## Common GUI Types:

Containers and Components

There are two types of GUI elements:

1. *Component*: Components are elementary GUI entities, such as Button, Label, and TextField.
2. *Container*: Containers, such as Frame and Panel, are used to *hold components in a specific layout* (such as FlowLayout or GridLayout). A container can also hold sub-containers.

In the above figure, there are three containers: a Frame and two Panels. A Frame is the *top-level container* of an AWT program. A Frame has a title bar (containing an icon, a title, and the minimize/maximize/close buttons), an optional menu bar and the content display area. A Panel is a *rectangular area* used to group related GUI components in a certain layout. In the above figure, the top-level Frame contains two Panels. There are five components: a Label (providing description), a TextField (for users to enter text), and three Buttons (for user to trigger certain programmed actions).

In a GUI program, a component must be kept in a container. You need to identify a container to hold the components. Every container has a method called add(Component c). A container (say c) can invoke c.add(aComponent) to add aComponent into itself. For example,

```
Panel pnl = new Panel();         // Panel is a container
Button btn = new Button("Press"); // Button is a component
pnl.add(btn);                    // The Panel container adds a Button component
```

GUI components are also called *controls* (e.g., Microsoft ActiveX Control), *widgets* (e.g., Eclipse's Standard Widget Toolkit, Google Web Toolkit), which allow users to interact with (or control) the application.

- ## Listener Interfaces:

SWING - Event Listeners

Event listeners represent the interfaces responsible to handle events. Java provides various Event listener classes, however, only those which are more frequently used will be discussed. Every method of an event listener method has a single argument as an object which is the subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.

- ### EventListner Interface
- It is a marker interface which every listener interface has to extend. This class is defined in **java.util** package.

- ### Class Declaration
- Following is the declaration for **java.util.EventListener** interface −
- public interface EventListener

- ### SWING Event Listener Interfaces
- Following is the list of commonly used event listeners.

Sr.No.	Class & Description
1	ActionListener This interface is used for receiving the action events.
2	ComponentListener

	This interface is used for receiving the component events.
3	ItemListener This interface is used for receiving the item events.
4	KeyListener This interface is used for receiving the key events.
5	MouseListener This interface is used for receiving the mouse events.
6	WindowListener This interface is used for receiving the window events.
7	AdjustmentListener This interface is used for receiving the adjustment events.
8	ContainerListener This interface is used for receiving the container events.
9	MouseMotionListener This interface is used for receiving the mouse motion events.
10	FocusListener This interface is used for receiving the focus events.

Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse
java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces

ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**

- public void addActionListener(ActionListener a){}

- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

- ## How Event Handling works:

How Events Work

Event handling in Java is comprised of two key elements:

- **The event source**, which is an object that is created when an event occurs. Java provides several types of these event sources, discussed in the section **Types of Events** below.
- **The event listener**, the object that "listens" for events and processes them when they occur.

There are several types of events and listeners in Java: each type of event is tied to a corresponding listener. For this discussion, let's consider a common type of event, an *action event* represented by the Java class *ActionEvent*, which is triggered when a user clicks a button or the item of a list.
At the user's action, an *ActionEvent* object corresponding to the relevant action is created. This object contains both the event source information and the specific action taken by the user. This event object is then passed to the corresponding *ActionListener* object's method:

```
void actionPerformed(ActionEvent e)
```

This method is executed and returns the appropriate GUI response, which might be to open or close a dialog, download a file, provide a digital signature, or any other of the myriad actions available to users in an interface.

Types of Events

Here are some of the most common types of events in Java:

- *ActionEvent*: Represents a graphical element is clicked, such as a button or item in a list. Related listener: *ActionListener.*
- *ContainerEvent*: Represents an event that occurs to the GUI's container itself, for example, if a user adds or removes an object from the interface. Related listener: *ContainerListener.*
- *KeyEvent*: Represents an event in which the user presses, types or releases a key. Related listener: *KeyListener.*
- *WindowEvent*: Represents an event relating to a window, for example, when a window is closed, activated or deactivated. Related listener: *WindowListener.*
- *MouseEvent*: Represents any event related to a mouse, such as when a mouse is clicked or pressed. Related listener: *MouseListener.*

Note that multiple listeners and event sources can interact with one another. For example, multiple events can be registered by a single listener, if they are of the same type. This means that, for a similar set of components that perform the same type of action, one event listener can handle all the events. Similarly, a single event can be bound to multiple listeners, if that suits the program's design (although that is less common).