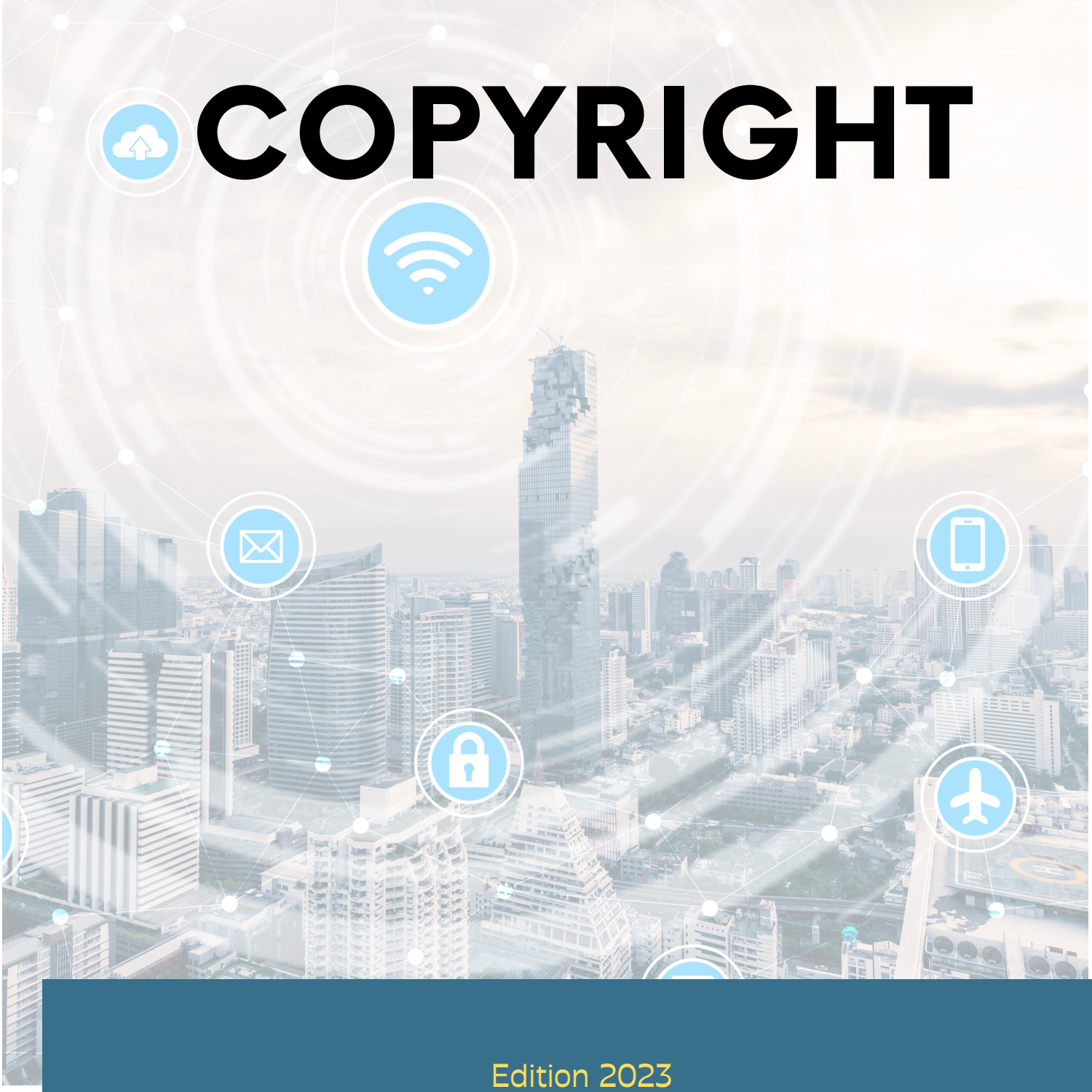
QUICK GUIDE

startup
java program

Written By

NORHAYATI SA'ADAH CHE ABD RAZAK
SUZIWATI YUSOF

COPYRIGHT



Edition 2023
Published By:
Politeknik Sultan Mizan Zainal Abidin.
Km 08, Jalan Paka,
23000 Dungun, Terengganu Darul Iman.
Tel: 09-8400800
Fax: 09-8458781
www.psmza.edu.my

QUICK GUIDE

startup java program

Written By

NORHAYATI SA'ADAH CHE ABD RAZAK
SUZIWATI YUSOF

JABATAN TEKNOLOGI MAKLUMAT DAN KOMUNIKASI
POLITEKNIK SULTAN MIZAN ZAINAL ABIDIN
KM 8, JALAN PAKA,
23000 DUNGUN
09-8400800

PREFACE

QUICK GUIDE

startup
java program

NORHAYATI SA'ADAH CHE ABD RAZAK

In the Name of Allah, the Most Gracious, the Most Merciful
First of all the writer's deepest thank To Allah SWT, the lord of the universe an to our prophet Muhammad SAW, may peace and blessing be upon him, his family and his followers.

I would like to express my gratitude to my beloved my family eternal pray, love, patience an all supports. Thank you for always giving inspiration an motivating me to finish this book, may Allah blesses them all.

SUZIWATI BINTI YUSOF

With all praise and thanks to the presence of the Almighty God, who has showered His love and grace so that the eBook titled "Quick Guide Startup Java Program" can be successfully completed.

On this occasion, I would like to say a thousand thanks to all parties who have helped us in preparing this eBook for their support and motivation and also to friends who have contributed their ideas in developing this eBook perfectly. Finally, hopefully the production of this eBook can provide the best benefits for students.

SYNOPSIS

QUICK GUIDE

startup
java program

Written By
NORHAYATI SA'ADAH CHE ABD RAZAK
SUZIWATI YUSOF

Provides students with the knowledge and skills needed to develop applications in Java Programming. course introduces students to the knowledge of GUI programming in Java. This course addresses the creation of interactive GUIs through standalone front-end applications.
This course primarily focuses on the Swing library, Abstract Window Toolkit (AWT) and also equips students with knowledge in the development of database applications solutions

TABLE OF CONTENT

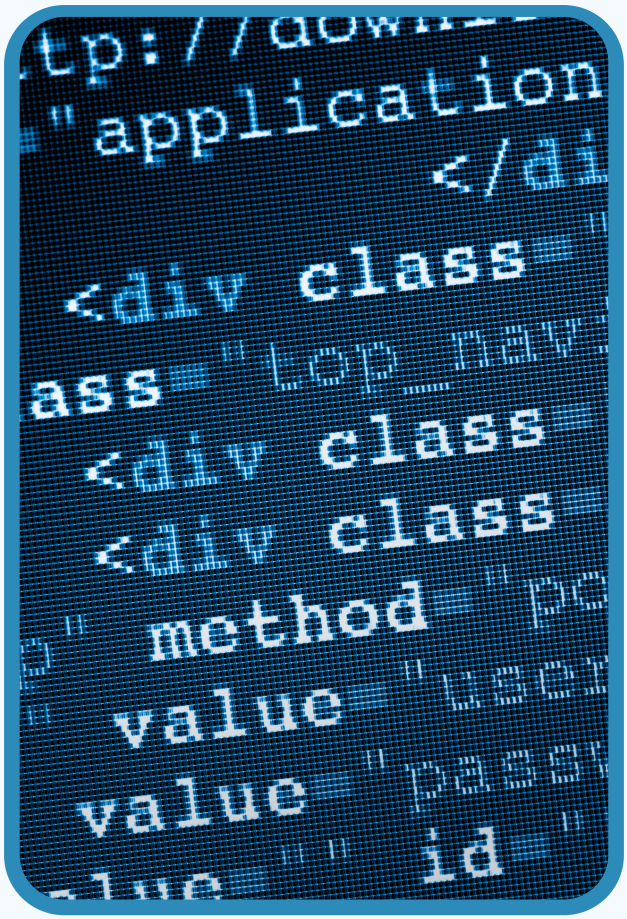
JAVA Notes

Chapter 1

ABSTRACT WINDOW TOOLKIT

ABSTRACT WINDOW TOOLKIT

A graphical user interface is built of graphical elements called components. Typical components include such items as buttons, scrollbars, and text fields.



Components allow the user to interact with the program and provide the user with visual feedback about the state of the program. In the AWT, all user interface components are instances of class Component or one of its subtypes

Components do not stand alone, but rather are found within containers. Containers contain and control the layout of components.

CHAPTER 1
ABSTRACT WINDOW TOOLKIT

Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

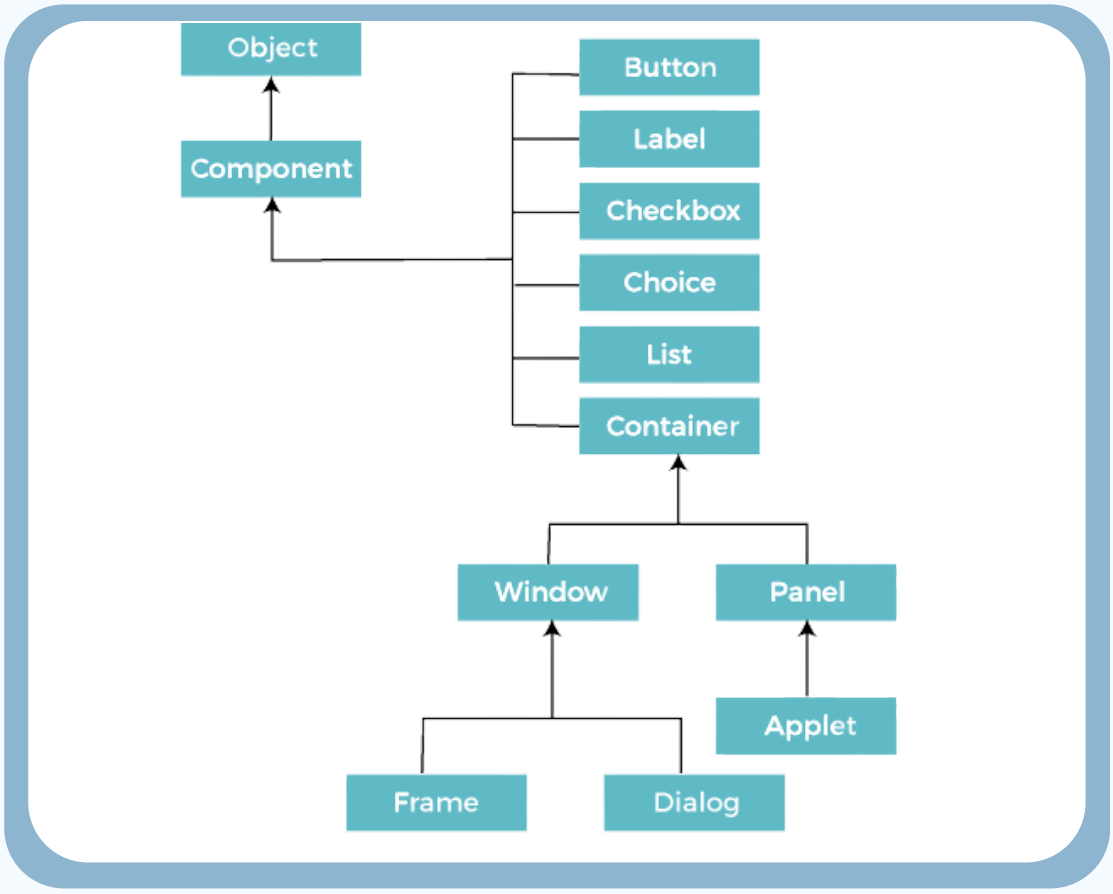


Figure 1.1 : AWT Hierarchy
Source : https://www.javatpoint.com/

ABSTRACT WINDOW TOOLKIT
ASPECT

UI ELEMENTS

These refers to the core visual elements which are visible to the user and used for interacting with the application. AWT in Java provides a comprehensive list of widely used and common elements.

LAYOUTS

These define how UI elements will be organized on the screen and provide the final look and feel to the GUI.

BEHAVIOR

These define the events which should occur when a user interacts with UI elements.

COMPONENTS & CONTAINER

COMPONENTS

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

CONTAINER

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

COMPONENTS & CONTAINER

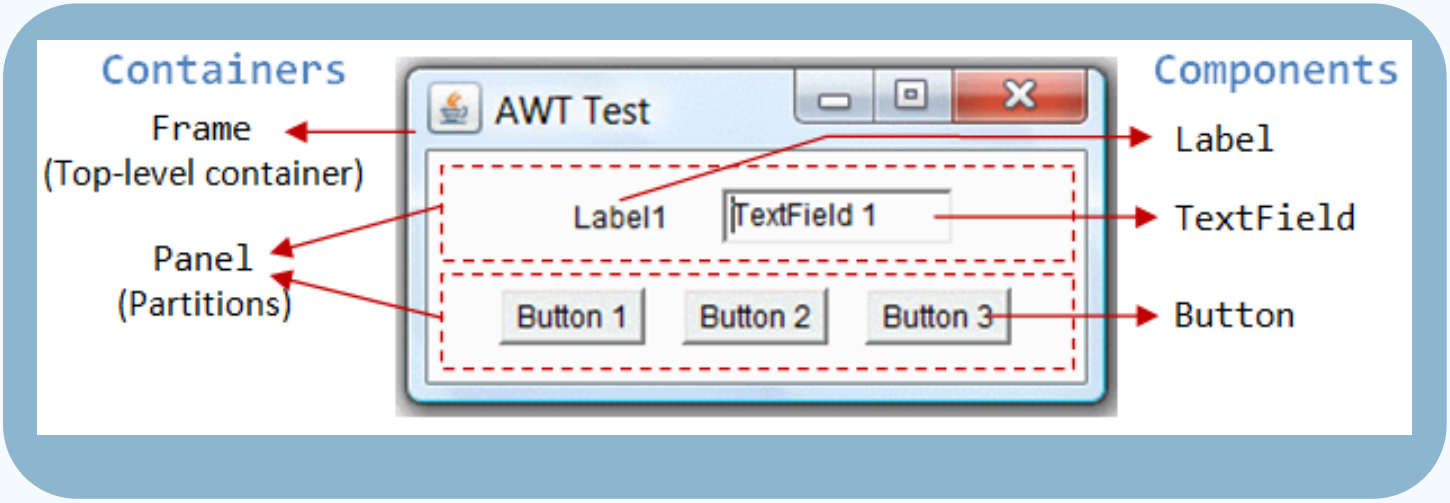


Figure 1.2 : AWT Components & Container
Source : https://www.javatpoint.com/

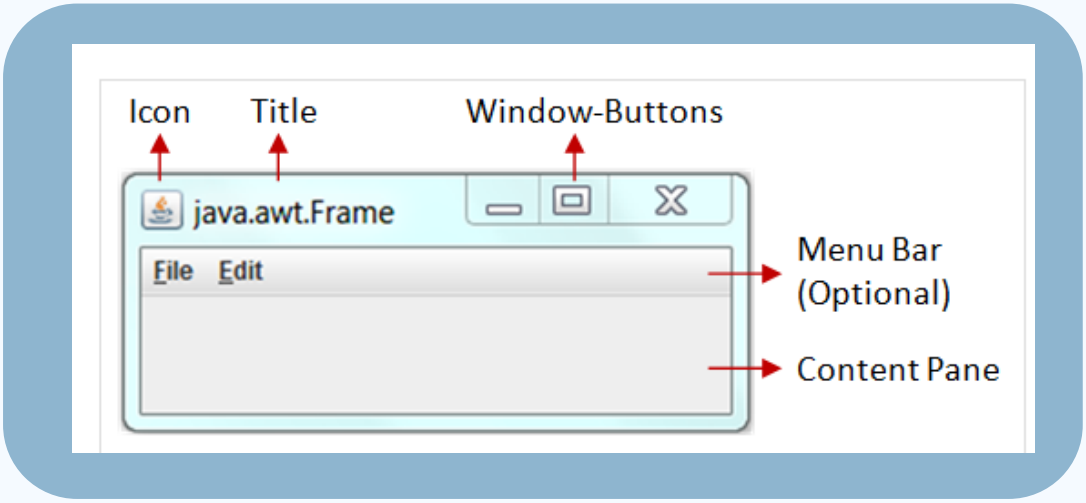


Figure 1. 2 : AWT Components & Container
Source : https://www.javatpoint.com/

CONTAINER

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

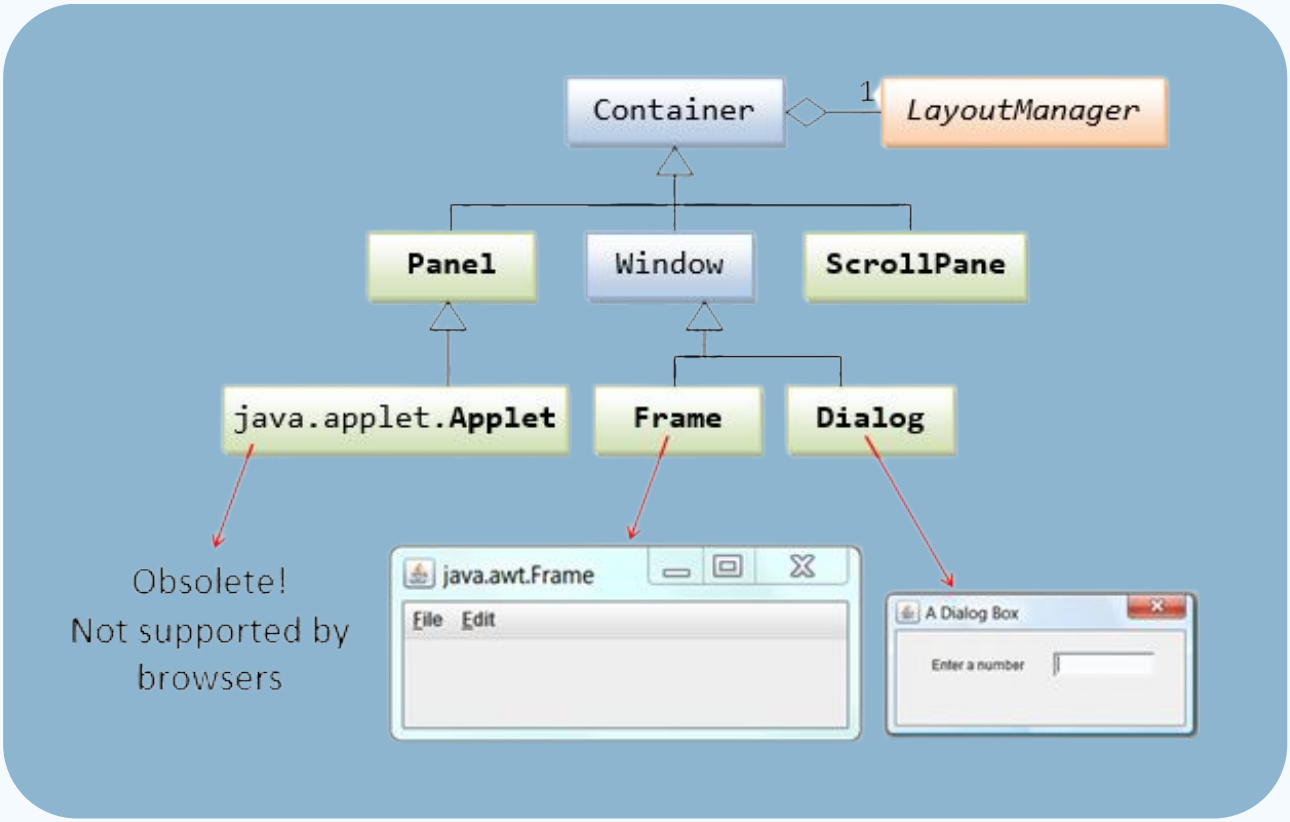


Figure 1.2 : Awt Container Hierarchy
source : https://www3.ntu.edu.sg

CONTAINER

WINDOW

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

PANEL

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

FRAME

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

FRAME

Frame is a top-level container which is used to hold components in it. Components such as a button, checkbox, radio button, menu, list, table etc.

CODE

```
import java.awt.*;
class FrameEx{
Frame frame;
Label label;
Button button;
FrameEx()
{
   frame = new Frame("Frame");
   label = new Label("Hello");
   frame.setSize(210,250);                                    f
   frame.setVisible(true);
}
   public static void main(String... ar)
   {
     new FrameEx();
   }
 }
```

PANEL

The panel provides a free space where components can be placed.

CODE

```
import java.awt.*;
public class PanelDemo1{
  PanelDemo1()
  {
    Frame panel_f= new Frame(" Panel Example");
    Panel panel11=new Panel();
    panel11.setBounds(40,80,200,200);
    panel11.setBackground(Color.red);
    Button box1=new Button("On");
    box1.setBounds(50,100,80,30);
    box1.setBackground(Color.gray);
    Button box2=new Button("Off");
    box2.setBounds(100,100,80,30);
    box2.setBackground(Color.gray);
    panel11.add(box1);
    panel11.add(box2);
    panel_f.add(panel11);
    panel_f.setSize(400,400);
    panel_f.setLayout(null);
    panel_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new PanelDemo1();
  }
}
```

COMPONENTS

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

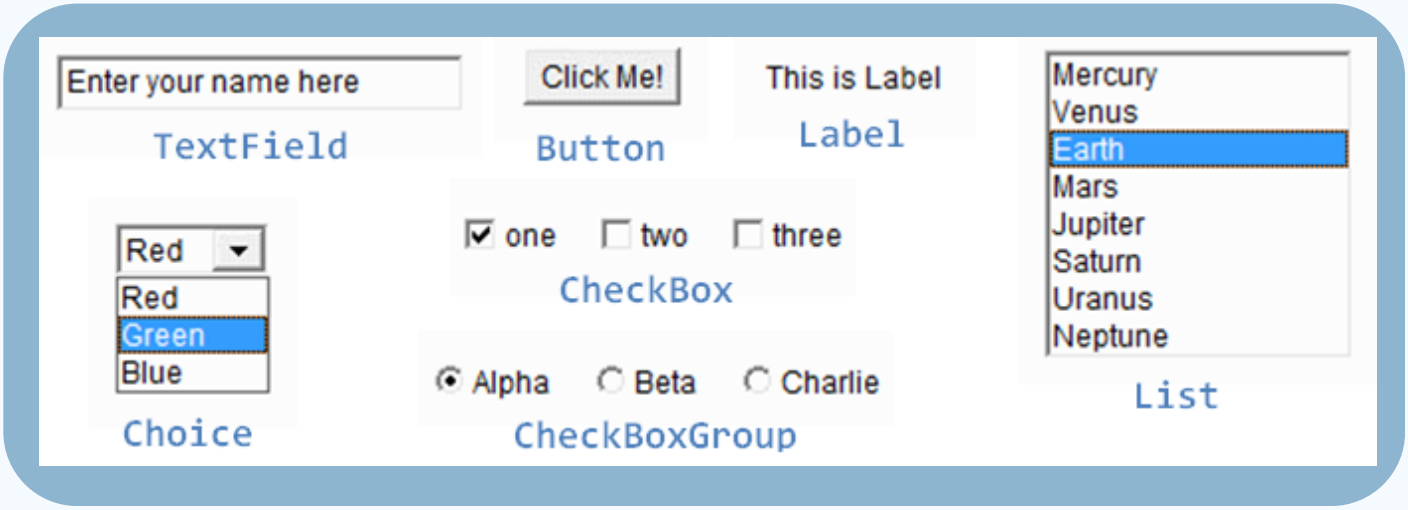
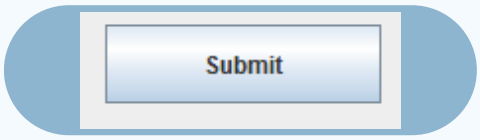


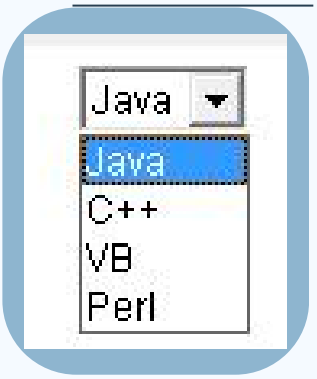
Figure 1.3 : AWT Components
source : https://https://www.edureka.com

COMPONETS

BUTTON

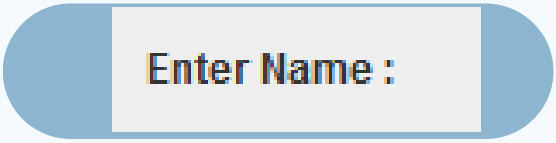


CHOICE

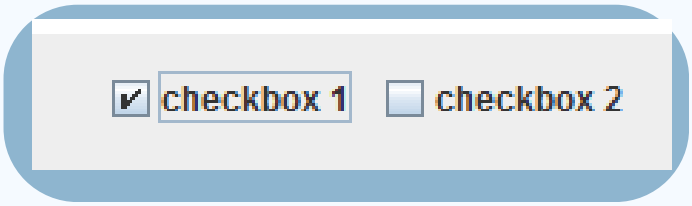


TEXTFIELD

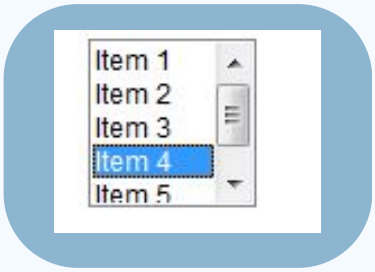
LABEL



CHECKBOX



LIST



RADIO BUTTON

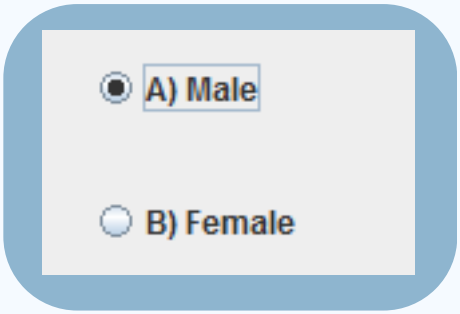


Figure 1.4 : AWT Components
source : https://javapoint.com

EXAMPLE JAVA CODE

AWT COMPONENTS

BUTTON

GUI component that triggers a certain programmed action upon clicking it

```
1  //Construct a Button with the given label
2  public Button(String btnLabel);
3
4  //Construct a Button with empty label
5  public Button();
```

CODE

```
import java.awt.*;
public class ButtonDemo1
{
public static void main(String[] args)
{
    Frame f1=new Frame("studytonight ==> Button Demo");
    Button b1=new Button("Press Here");
    b1.setBounds(80,200,80,50);
    f1.add(b1);
    f1.setSize(500,500);
    f1.setLayout(null);
    f1.setVisible(true);
}
}
```

LABEL

Provides a descriptive text string that is visible on GUI.
An AWT Label object is a component for placing text in a container

```
// Construct a Label with the given text String, of the text alignment
public Label(String strLabel, int alignment);

//Construct a Label with the given text String
public Label(String strLabel);

//Construct an initially empty Label
public Label();
```

CODE

```
import java.awt.*;
class LabelDemo1
{
  public static void main(String args[])
  {
    Frame l_Frame= new Frame("studytonight ==> Label Demo");
    Label lab1,lab2;
    lab1=new Label("Welcome to studytonight.com");
    lab1.setBounds(50,50,200,30);
    lab2=new Label("This Tutorial is of Java");
    lab2.setBounds(50,100,200,30);
l_Frame.add(lab1);
l_Frame.add(lab2);
l_Frame.setSize(500,500);
l_Frame.setLayout(null);
l_Frame.setVisible(true);
}
```

TEXTFIELD

TextField class creates a single-line text box for users to enter texts.

```java
//Construct a TextField instance with the given initial text string
public TextField(String initialText, int columns);

//Construct a TextField instance with the given initial text string.
public TextField(String initialText);

//Construct a TextField instance with the number of columns.
public TextField(int columns);
```

CODE

```
import java.awt.*;
class TextFieldDemo1{
public static void main(String args[]){
    Frame TextF_f= new Frame("studytonight ==>TextField");
TextField text1,text2;
    text1=new TextField("Welcome to studytonight");
    text1.setBounds(60,100, 230,40);
    text2=new TextField("This tutorial is of Java");
    text2.setBounds(60,150, 230,40);
TextF_f.add(text1);
TextF_f.add(text2);
TextF_f.setSize(500,500);
TextF_f.setLayout(null);
TextF_f.setVisible(true);
}
}
```

TEXT AREA

TextArea class is a multiline region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

The text area allows us to type as much text as we want. When the text in the text area becomes larger than the viewable area, the scroll bar appears automatically which helps us to scroll the text up and down, or right and left.

CODE

```
   import java.awt.*;
public class TextAreaDemo1
{
 TextAreaDemo1()
 {
 Frame textArea_f= new Frame();
 TextArea area=new TextArea("Welcome to studytonight.com");
 area.setBounds(30,40, 200,200);
 textArea_f.add(area);
 textArea_f.setSize(300,300);
 textArea_f.setLayout(null);
 textArea_f.setVisible(true);
 }
 public static void main(String args[])
 {
 new TextAreaDemo1
```

CHECKBOX

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

CODE

```
 import java.awt.*;
public class CheckboxDemo1
{
  CheckboxDemo1(){
    Frame checkB_f= new Frame("studytonight ==>Checkbox
Example");
    Checkbox ckbox1 = new Checkbox("Yes", true);
    ckbox1.setBounds(100,100, 60,60);
    Checkbox ckbox2 = new Checkbox("No");
    ckbox2.setBounds(100,150, 60,60);
    checkB_f.add(ckbox1);
    checkB_f.add(ckbox2);
    checkB_f.setSize(400,400);
    checkB_f.setLayout(null);
    checkB_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new CheckboxDemo1();
  }
}
```

CHOICE

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

CODE

```
import java.awt.*;
public class ChoiceDemo
{
  ChoiceDemo()
  {
    Frame choice_f= new Frame();
    Choice obj=new Choice();
    obj.setBounds(80,80, 100,100);
    obj.add("Red");
    obj.add("Blue");
    obj.add("Black");
    obj.add("Pink");
    obj.add("White");
    obj.add("Green");
    choice_f.add(obj);
    choice_f.setSize(400,400);
    choice_f.setLayout(null);
    choice_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new ChoiceDemo();
  }
}
```

LIST

The object of List class represents a list of text items. With the help of the List class, user can choose either one item or multiple items. It inherits the Component class.

CODE

```
import java.awt.*;
public class ListDemo
{
  ListDemo()
  {
    Frame list_f= new Frame();
    List obj=new List(6);
    obj.setBounds(80,80, 100,100);
    obj.add("Red");
    obj.add("Blue");
    obj.add("Black");
    obj.add("Pink");
    obj.add("White");
    obj.add("Green");
    list_f.add(obj);
    list_f.setSize(400,400);
    list_f.setLayout(null);
    list_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new ListDemo();
  }
}
```

LIST

The object of List class represents a list of text items. With the help of the List class, user can choose either one item or multiple items. It inherits the Component class.

CODE

```
import java.awt.*;
public class ListDemo
{
  ListDemo()
  {
    Frame list_f= new Frame();
    List obj=new List(6);
    obj.setBounds(80,80, 100,100);
    obj.add("Red");
    obj.add("Blue");
    obj.add("Black");
    obj.add("Pink");
    obj.add("White");
    obj.add("Green");
    list_f.add(obj);
    list_f.setSize(400,400);
    list_f.setLayout(null);
    list_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new ListDemo();
  }
}
```

MENU & MENUBAR

As we know that every top-level window has a menu bar associated with it. This menu bar consist of various menu choices available to the end user. Further each choice contains list of options which is called drop down menus. Menu and MenuItem controls are subclass of MenuComponent class.

MENU COMPONENT

It is the top level class for all menu related controls.

MENU BAR

The MenuBar object is associated with the top-level window.

MENU ITEM

The items in the menu must belong to the MenuItem or any of its subclass.

MENU

The Menu object is a pull-down menu component which is displayed from the menu bar.

CHECKBOX MENU ITEM

CheckboxMenuItem is subclass of MenuItem.

POPUP MENU

PopupMenu can be dynamically popped up at a specified position within a component.

CODE

```
import java.awt.*;
class MenuDemo1
{
  MenuDemo1()
  {
    Frame menu_f= new Frame("studytonight ==> Menu and
MenuItem Demo");
    MenuBarmenu_bar=new MenuBar();
    Menu menu11=new Menu("Menu");
    Menu sub_menu1=new Menu("Sub Menu =>");
    MenuItem a1=new MenuItem("Red");
    MenuItem a2=new MenuItem("Light Red");
    MenuItem a3=new MenuItem("Drak Red");
    MenuItem b1=new MenuItem("Green");
    MenuItem b2=new MenuItem("Light Green");
    MenuItem b3=new MenuItem("Dark Green");
    menu11.add(a1);
    sub_menu1.add(a2);
    sub_menu1.add(a3);
    menu11.add(b1);
    sub_menu1.add(b2);
    sub_menu1.add(b3);
    menu11.add(sub_menu1);
    menu_bar.add(menu11);
    menu_f.setMenuBar(menu_bar);
    menu_f.setSize(400,400);
    menu_f.setLayout(null);
    menu_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new MenuDemo1();
  }
}
```

END OF CHAPTER 1

Chapter 2

SWING

SWING

A component is an independent visual control and Java Swing Framework contains a large set of these 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.

In Java Swing, there are several components like a scroll bar, button, text field, text area, checkbox, radio button, etc. These components jointly form a GUI that offers a rich set of functionalities and allows high-level customization.

CHARATERISTIC

The swing component's data is represented using Model.

The controller component of the MVC architecture reads input from the user on the view and then these changes are passed to the component data.

In each Swing component, the view and controller are clubbed together while the model is a separate one. This gives swing a pluggable look and feel feature.

It is visually represented using a view.

FEATURES OF THE SWING API

platform-independent

extensible

light-weight

rich in functionality

PAGE

FEATURES OF THE SWING API

highly customizable

look-and-feel

simply change

SWING CONTAINER

JSwing has a big set of components that we can include in our programs and avail the rich functionalities using which we can develop highly customized and efficient GUI applications.

A **component** can be defined as a control that can be represented visually and is usually independent. It has got a specific functionality and is represented as an individual class in Swing API. For example, class JButton in swing API is a button component and provides the functionality of a button.

One or more components form a group and this group can be placed in a "Container". A **container** provides a space in which we can display components and also manage their spacing, layout, etc.

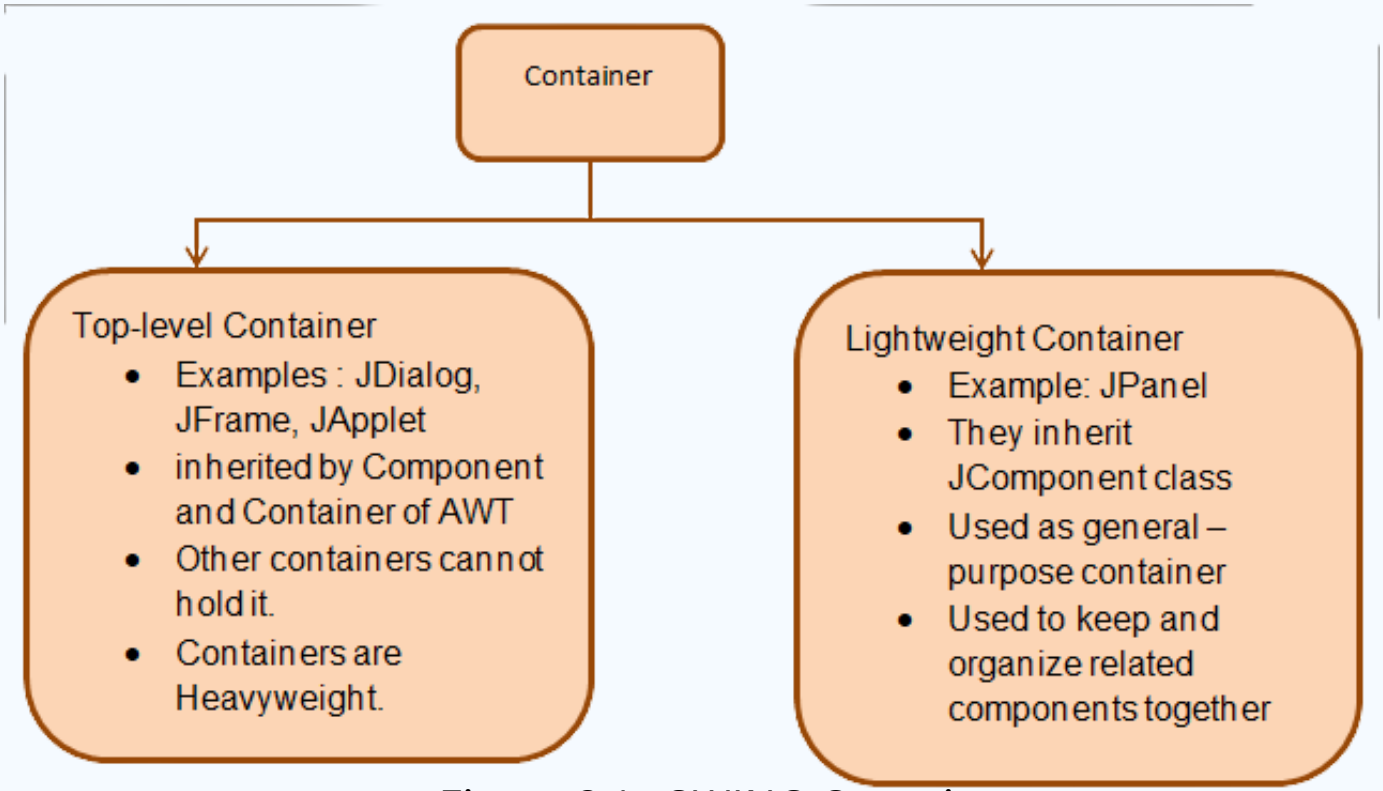


Figure 2.1 : SWING Container
Source : https://www.softwaretestinghelp.com//

SWING CLASS

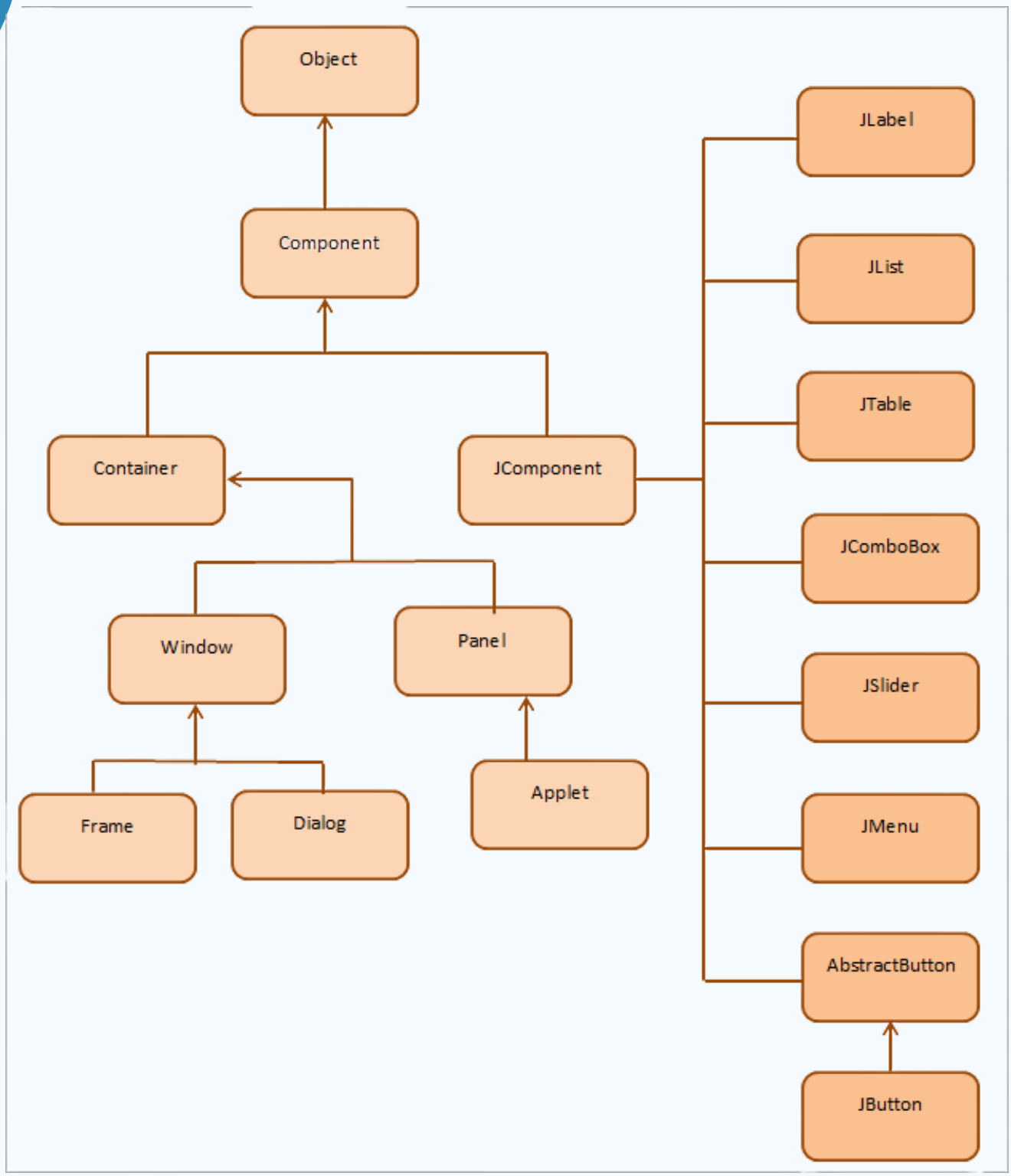


Figure 2.2 : SWING Container
Source : https://www.softwaretestinghelp.com//

IMPORTANT CLASSES OF SWING

JWINDOW

The JWindow class of Swing inherits the Window class directly. The JWindow class uses ‘BorderLayout’ as the default layout.

JPANEL

JPanel is a descendent of JComponent class and is on similar lines to AWT class Panel and has ‘FlowLayout’ as the default layout.

JFRAME

JFrame descends from the Frame class. The components added to the Frame are called contents of the Frame.

IMPORTANT CLASSES OF SWING

JLABEL

JLabel class is a subclass of the JComponent. It is used to create text labels in the application.

JBUTTON

The push-button functionality in Swing is provided by JButton. We can associate a string, an icon, or both with the JButton object.

JTEXTFIELD

JJTextField class provides a text field in which we can edit a single line of text.

JAVA COMPONENTS

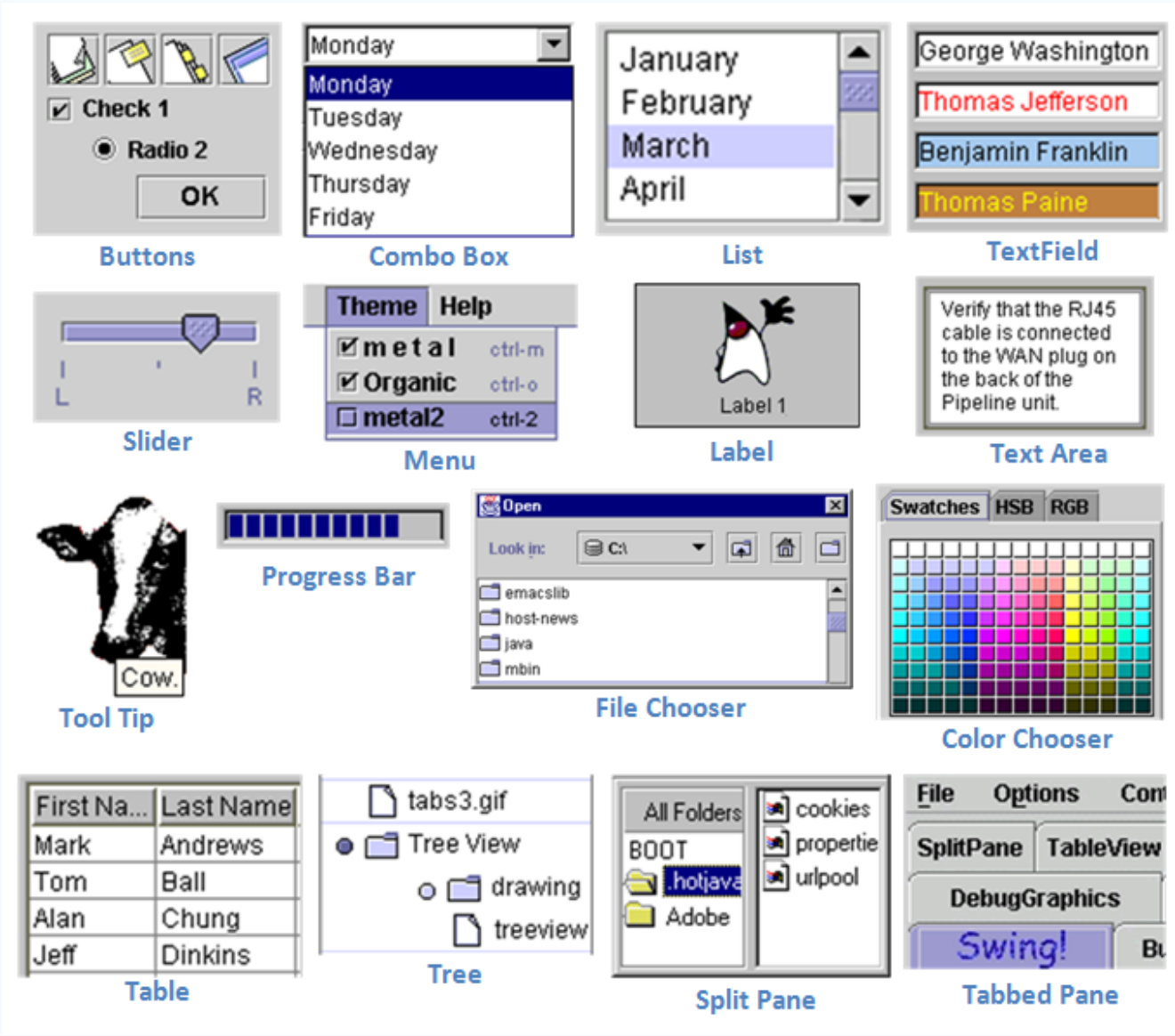


Figure 2.3 : SWING Components
Source : https://www3.ntu.edu.sg/

EXAMPLE JAVA CODE

SWING COMPONENTS

JFRAME

Frame, in general, is a container that can contain other components such as buttons, labels, text fields, etc. A Frame window can contain a title, a border, and also menus, text fields, buttons, and other components. An application should contain a frame so that we can add components inside it.

The Frame in Java Swing is defined in class javax.swing.JFrame. JFrame class inherits the java.awt.Frame class. JFrame is like the main window of the GUI application using swing.

By Extending The JFrame Class

CODE

```
import javax.swing.*;
class FrameInherited extends JFrame{    //inherit from JFrame class
    JFrame f;
    FrameInherited(){
        JButton b=new JButton("JFrame_Button");//create button object
        b.setBounds(100,50,150, 40);

        add(b);//add button on frame
        setSize(300,200);
        setLayout(null);
        setVisible(true);
    }
}
public class Main {
    public static void main(String[] args) {
      new FrameInherited(); //create an object of FrameInherited class
    }
}
```

JFRAME

By Instantiating The JFrame Class

CODE

```
import javax.swing.*;
class FrameInherited extends JFrame{    //inherit from JFrame class
    JFrame f;
    FrameInherited(){
        JButton b=new JButton("JFrame_Button");//create button object
        b.setBounds(100,50,150, 40);

        add(b);//add button on frame
        setSize(300,200);
        setLayout(null);
        setVisible(true);
    }
}
public class Main {
    public static void main(String[] args) {
      new FrameInherited(); //create an object of FrameInherited class
    }
}
```

JPANEL

CODE

```
import javax.swing.*;
class JPanelExample {
    JPanelExample(){
        JFrame frame = new JFrame("Panel Example"); //create a frame
        JPanel panel = new JPanel(); //Create JPanel Object
        panel.setBounds(40,70,100,100); //set dimensions for Panel
        JButton b = new JButton("ButtonInPanel"); //create JButton object
        b.setBounds(60,50,80,40); //set dimensions for button
        panel.add(b);   //add button to the panel
        frame.add(panel);   //add panel to frame
        frame.setSize(400,400);
        frame.setLayout(null);
        frame.setVisible(true);
    }

}
public class Main {
    public static void main(String[] args) {
      new JPanelExample(); //create an object of FrameInherited class
    }
}
```

JTEXTAREA

TextArea defines an editable text field. It can have multiple lines. The swing class that defines the text area is JTextArea and it inherits the JTextComponent class.

public class JTextArea extends JTextComponent
JTextArea class contains 4 constructors that allow us to create a text area with various options.

`JTextArea ( )`

Default constructor. Create an empty text area.

`JTextArea (String s)`

Creates a textarea with s as the default value.

`JTextArea (int row, int column)`

Creates a text area with a specified row x column.

`JTextArea (String s, int row, int column)`

Creates a text area with specified row x column and default value s.

JTEXTAREA

CODE

```
import javax.swing.*;
class JTextAreaExample {
    JTextAreaExample(){
        JFrame frame= new JFrame();
        JTextArea t_area=new JTextArea("JTextArea example");    //create object of JTextArea
        t_area.setBounds(10,30, 150,100);  //set its dimensions
        frame.add(t_area);       //add it to the frame
        frame.setSize(200,200);
        frame.setLayout(null);
        frame.setVisible(true);
    }

}
public class Main {
    public static void main(String[] args) {
      new JTextAreaExample(); //create an object of TextAreaExample class
    }
}
```

JBUTTON

A button is a component that is used to create a push button with a name or label on it. In swing, the class that creates a labeled button is JButton. JButton inherits the AbstractButton class. We can associate the ActionListener event to the button to make it take some action when it is pushed.

CODE

```
import javax.swing.*;

public class Main {
    public static void main(String[] args) {

        JFrame frame=new JFrame("JButton Example"); //create JFrame object
        JButton button=new JButton("Button");        //Create a JButton object
        button.setBounds(50,50,75,35); //set dimensions for button
        frame.add(button);                                    //add button to the fra
        frame.setSize(250,200);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```

JLIST

A list consists of multiple text items. Users can either select a single item or multiple items at a time. The class that implements the list in swing API is JList. JList is a descendent of the JComponent class.

Default constructor that creates an empty, read-only list.

```
JList ()
```

Create a JList which initially contains elements of array listItem.

```
JList (array[] listItem
```

Creates a list with elements from the specified model dataModel.

```
JList (ListModel<array> dataModel)
```

JLIST

CODE

```
import javax.swing.*;

public class Main {
    public static void main(String[] args) {
        JFrame frame= new JFrame("JList Example");
        //create a list model and add items to it
        DefaultListModel&lt;String&gt; colors = new DefaultListModel&lt;&gt;();
        colors.addElement("Red");
        colors.addElement("Green");
        colors.addElement("Blue");
        //create JList object and add listModel to it
        JList&lt;String&gt; colorsList = new JList&lt;&gt;(colors);
        colorsList.setBounds(100,100, 75,50);
        frame.add(colorsList);              //add list to the frame
        frame.setSize(400,400);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```

JCOMBOBOX

The JCombobox class shows a list of choices from which a user can select an option. The selected choice is at the top. JComboBox derives from the JComponent class.

`JComboBox ()`

Default constructor that creates a ComboBox with the default data model.

`JComboBox (Object[] items)`

This constructor creates a ComboBox having items as elements of the given array items.

`JComboBox (Vector<?> items)`

This constructor reads the elements of the given vector and constructs a ComboBox with these elements as its items.

JCOMBOBOX

CODE

```
import javax.swing.*;
class ComboBoxExample {
    JFrame frame;
    ComboBoxExample(){
        frame=new JFrame("ComboBox Example");
        //create a string array
        String country[]={"India","SriLanka","Singapore","Maldives","SeyChelles"};
        //create a combobox object with given string array
        JComboBox countries=new JComboBox(country);
        countries.setBounds(50, 50,90,20);
        frame.add(countries);       //add it to the frame
        frame.setLayout(null);
        frame.setSize(200,300);
        frame.setVisible(true);
    }
}
public class Main {
    public static void main(String arg[]) {
       new ComboBoxExample();
    }
}
```

JPASSWORD FIELD

In Java, Swing toolkit contains a JPasswordField Class. It is under package javax.swing.JPasswordField class. It is specifically used for password and it can be edited.

creates a new password field (0 column width, null String)	`JPasswordField( )`
creates a new password field with given number of columns	`JPasswordField(int columns)`
creates a new password field with specified text	`JPasswordField(String text)`
creates a new password field with specified text and columns	`JPasswordField(String text, int columns)`

JPASSWORD FIELD

CODE

```
import javax.swing.*;
public class SPasswordFieldDemo1
{
    public static void main(String[] args)
  {
  JFrame passWord_f=new JFrame("studytonight ==> Password Field Demo");
  JPasswordField passWord_value = new JPasswordField();
  JLabel passWord_l1=new JLabel("Password ");
        passWord_l1.setBounds(20,100, 100,30);
        passWord_value.setBounds(100,100,100,30);
        passWord_f.add(passWord_value);
  passWord_f.add(passWord_l1);
        passWord_f.setSize(300,300);
        passWord_f.setLayout(null);
        passWord_f.setVisible(true);
  }
}
```

FLOWLAYOUT

The FlowLayout arranges the components in a flow direction, one after another. This is the default layout for the containers like Panel and Applet.

The FlowLayout class in Java that represents the FlowLayout manager contains the following Fields and constructors.

- `public static final int LEADING`
- `public static final int TRAILING`
- `public static final int LEFT`
- `public static final int RIGHT`
- `public static final int CENTER`

FIELDS OF FLOWLAYOUT CLASS

CONSTRUCTORS OF FLOWLAYOUT CLASS

In Java, Swing toolkit contains a JPasswordField Class. It is under package javax.swing.JPasswordField class. It is specifically used for password and it can be edited.

`FlowLayout ()`

This is a default constructor. This constructor creates a flow layout having centrally aligned components with a default gap of 5 units in the horizontal and vertical direction.

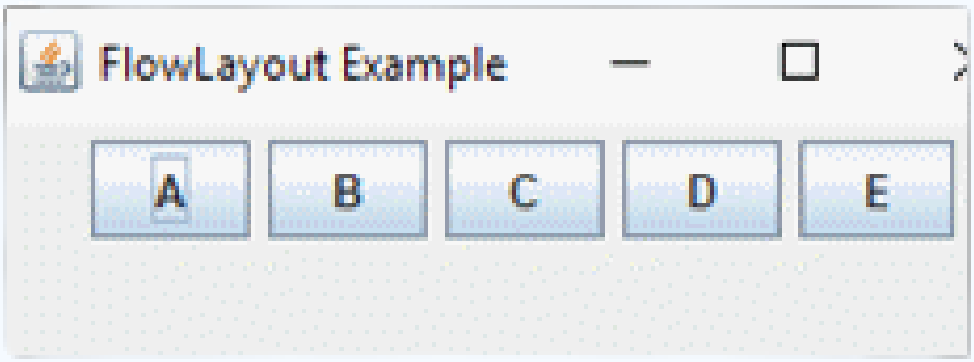
`FlowLayout (int align)`

This constructor creates a flow layout with the specified alignment value and with a horizontal and vertical gap of 5 units.

`FlowLayout (int align, int hgap, int vgap)`

Creates a flow layout with specified alignment value and horizontal and vertical gap.

FLOWLAYOUT EXAMPLE



FLOWLAYOUT EXAMPLE

CODE

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

class FlowLayoutClass {
    JFrame frame;
    FlowLayoutClass() {
        frame = new JFrame("FlowLayout Example");
        //create button components
        JButton b1 = new JButton("A");
        JButton b2 = new JButton("B");
        JButton b3 = new JButton("C");
        JButton b4 = new JButton("D");
        JButton b5 = new JButton("E");
        //add components to the frame
        frame.add(b1);
        frame.add(b2);
        frame.add(b3);
        frame.add(b4);
        frame.add(b5);
        //set layout as 'FlowLayout.CENTER'
        frame.setLayout(new FlowLayout(FlowLayout.CENTER));
        //setting flow layout of right alignment

        frame.setSize(300, 300);
        frame.setVisible(true);
    }

}
public class Main{
    public static void main(String[] args) {
        new FlowLayoutClass();
    }
}
```

GRID LAYOUT

Using GridLayout we can layout the components in a rectangular grid fashion i.e. each component is arranged in each rectangle.

`GridLayout ()`

default constructor that generates a grid layout having one column per one component in a row.

This constructor generates a grid layout with specified rows and columns. There is no gap between the components.

`GridLayout (int rows, int columns)`

`GridLayout (int rows, int columns, int hgap, int vgap)`

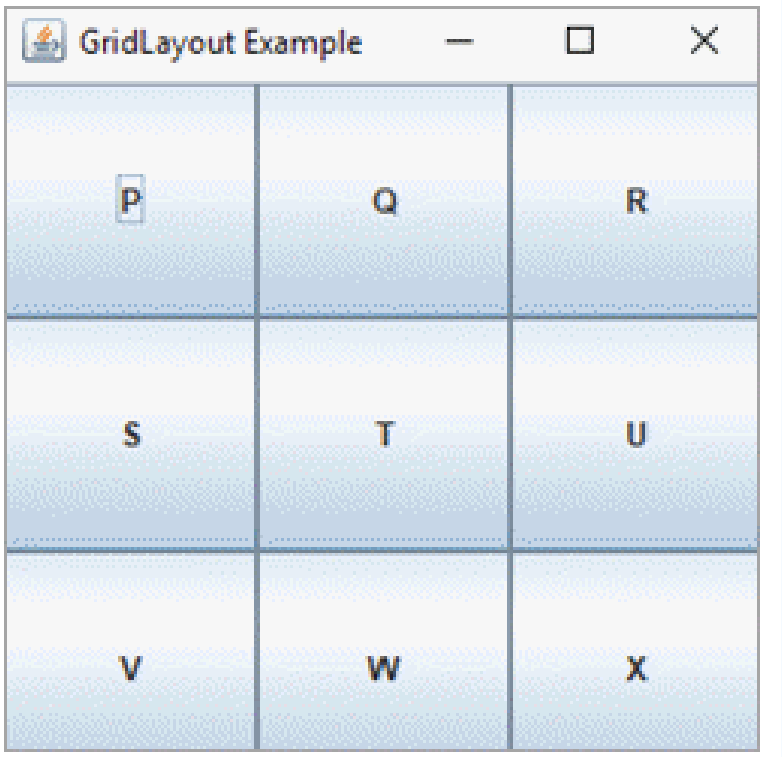
Using this constructor, we generate a grid layout with specified rows and columns and horizontal and vertical gaps.

GRID LAYOUT EXAMPLE

CODE

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

class GridLayoutClass {
    JFrame frame;

    GridLayoutClass() {
        frame=new JFrame("GridLayout Example");
        //create components to be laid out as per GridLayout
        JButton b1=new JButton("P");
        JButton b2=new JButton("Q");
        JButton b3=new JButton("R");
        JButton b4=new JButton("S");
        JButton b5=new JButton("T");
        JButton b6=new JButton("U");
        JButton b7=new JButton("V");
        JButton b8=new JButton("W");
        JButton b9=new JButton("X");
        //add components to the frame
        frame.add(b1);frame.add(b2);frame.add(b3);frame.add(b4);frame.add(b5);
        frame.add(b6);frame.add(b7);frame.add(b8);frame.add(b9);
        //set frame layout to GridLayout of 3 rows and 3 columns
        frame.setLayout(new GridLayout(3,3));

        frame.setSize(300,300);
        frame.setVisible(true);
    }

}
public class Main{
    public static void main(String[] args) {
        new GridLayoutClass();
    }
}
```

GRID LAYOUT EXAMPLE



BORDER LAYOUT

This layout will display the components along the border of the container. This layout contains five locations where the component can be displayed. Locations are North, South, East, west, and Center. The default region is the center. The above regions are the predefined static constants belonging to the BorderLayout class. Whenever other regions' spaces are not in use automatically container selects as a center region default and the component occupies the surrounding region's spaces of the window and which damages the look and feel of the user interface.

`BorderLayout()`

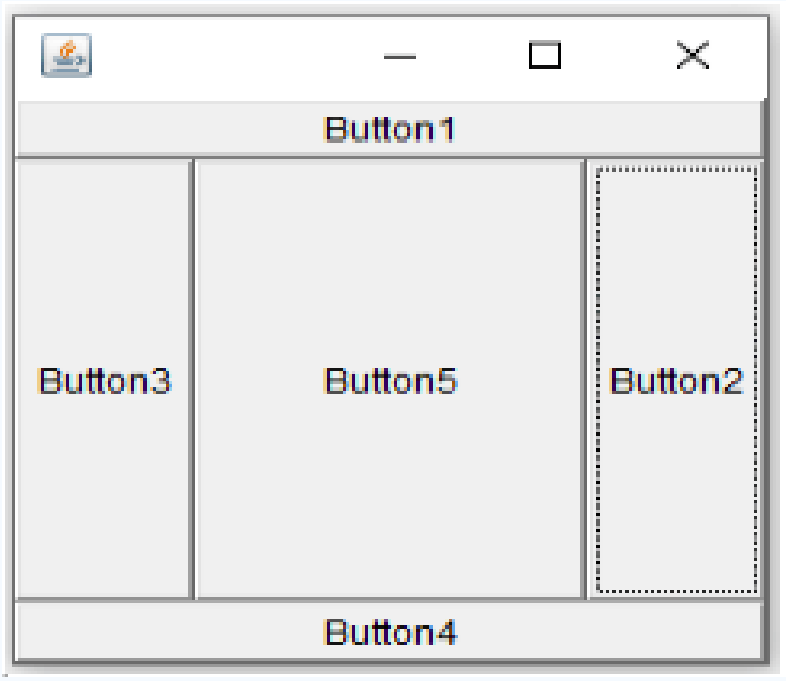
It will construct a new borderlayout with no gaps between the components.

`BorderLayout (int hgap, int vgap)`

It will constructs a border layout with the specified gaps between the components.

BORDER LAYOUT

```
import java.awt.*;
public class BorderLayoutDemo
{
    public static void main (String[]args)
    {
        Frame f1 = new Frame ();
        f1.setSize (250, 250);
        Button b1 = new Button ("Button1");
        Button b2 = new Button ("Button2");
        Button b3 = new Button ("Button3");
        Button b4 = new Button ("Button4");
        Button b5 = new Button ("Button5");
        f1.add (b1, BorderLayout.NORTH);
        f1.add (b2, BorderLayout.EAST);
        f1.add (b3, BorderLayout.WEST);
        f1.add (b4, BorderLayout.SOUTH);
        f1.add (b5);
        f1.setVisible (true);
    }
}
```



BOX LAYOUT

This manager is like a vertical version of the FlowLayout. Like the FlowLayout, each component is placed in the order in which it was added, and each component gets its own size.

But the BoxLayout can arrange the components vertically by inserting Y_AXIS (thus causing a vertical stacking).

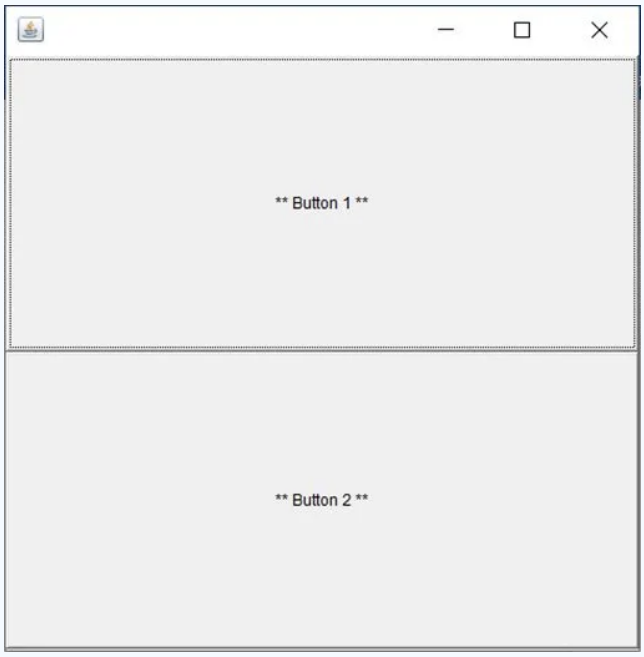
Box Layout is used, when we want to arrange the components vertically or horizontally.

Alignment of the components are horizontal from left to right.

```
BoxLayout(Container c, int axis)
```

BOX LAYOUT

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

public class BoxDemo1 extends Frame {
 Button buttonBox[];
 public BoxDemo1 ()
 {
  buttonBox = new Button [2];
  for (inti = 0;i<2;i++)
  {
   buttonBox[i] = new Button ("** Button " + (i + 1)+" **");
   add (buttonBox[i]);
  }
 setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
 setSize(500,500);
 setVisible(true);
 }

public static void main(String args[])
{
 BoxDemo1 obj=new BoxDemo1();
}
}
```



END OF CHAPTER 2

Chapter 3

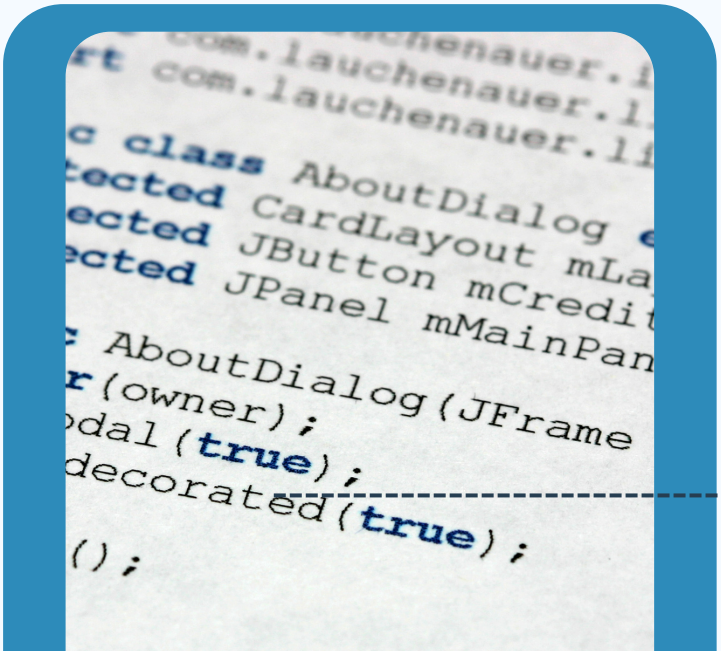
EVENT HANDLING

EVENT HANDLING

An event can be defined as a change of state of an object. From the GUI point of view, an event occurs when the end-user interacts with the GUI components. The events that get triggered in the GUI can be the click of a button, scrolling, selecting list items, changing text, etc.

Event occurring in the GUI listed above are mostly foreground events. We can also have some background events like background operation completion, timer expiration, etc.

Event handling is a mechanism through which an action is taken when an event occurs. For this, we define a method which is also called an event handler that is called when an event occurs. Java uses a standard mechanism called the “Delegation event model” to generate as well as handle events.



THE DELEGATION EVENT MODEL

SOURCE

The Source of the event is the object. The object on which an event occurs is the source and the source is responsible for sending information about the event to the event handler.

LISTENER

The listener is nothing but the event handler responsible for taking an action when an event occurs. In Java, a listener is an object that waits on an event. Once the event occurs, the listener processes the event.

COMPONENTS OF EVENT HANDLING

An event is a change in state of an object.

EVENTS

EVENTS SOURCE

Event source is an object that generates an event.

A listener is an object that listens to the event. A listener gets notified when an event occurs.

LISTENERS

TYPES OF EVENT

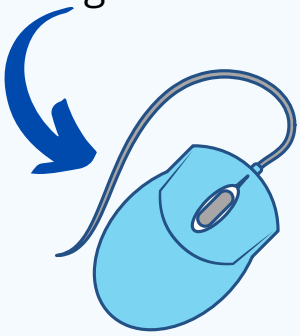
The events can be broadly classified into two categories as below:

FOREGROUND EVENTS

- Those events require the direct interaction of the user.
- They are generated as consequences of a person interacting with the graphical components in Graphical User Interface

EXAMPLE

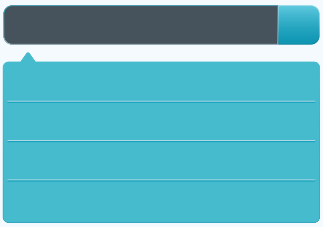
Moving the mouse

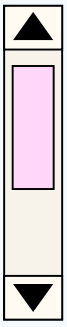


Clicking on a button

Entering a character through the keyboard

Selecting an item from the list





Scrolling the page

PAGE

TYPES OF EVENT

The events can be broadly classified into two categories as below:

BACKGROUND EVENTS

- Those events that require the interaction of the end-user are known as background events.
- Operating system interrupts, hardware or software failure, the timer expires, an operation completion is the example of background events.

FLOW OF EVENT HANDLING

User Interaction with a component is required to generate an event.

The object of the respective event class is created automatically after event generation, and it holds all information of the event source.

The newly created object is passed to the methods of the registered listener.

The method executes and returns the result.

IMPORTANT EVENT CLASSES AND INTERFACE

Event Class	Listener Interface	Description
ActionEvent	ActionListener	An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list.
AdjustmentEvent	AdjustmentListener	The adjustment event is emitted by an Adjustable object like Scrollbar.
ComponentEvent	ComponentListener	An event that indicates that a component moved, the size changed or changed its visibility.
ContainerEvent	ContainerListener	When a component is added to a container (or) removed from it, then this event is generated by a container object.
FocusEvent	FocusListener	These are focus-related events, which include focus, focusin, focusout, and blur.
ItemEvent	ItemListener	An event that indicates whether an item was selected or not.
MouseEvent	MouseListener & MouseMotionListener	The events that occur due to the user interaction with the mouse (Pointing Device).

IMPORTANT EVENT CLASSES AND INTERFACE

Event Class	Listener Interface	Description
KeyEvent	KeyListener	An event that occurs due to a sequence of keypresses on the keyboard.
MouseWheelEvent	MouseWheelListener	An event that specifies that the mouse wheel was rotated in a component
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	An event which indicates whether a window has changed its status or not.

IMPORTANT EVENT CLASSES AND INTERFACE

Different interfaces consists of different methods which are specified below.

Listener Interface	Methods
ActionListener	actionPerformed()
AdjustmentListener	adjustmentValueChanged()
ComponentListener	componentResized() componentShown() componentMoved() componentHidden()
ContainerListener	componentAdded() componentRemoved()
FocusListener	focusGained() focusLost()
ItemListener	itemStateChanged()

IMPORTANT EVENT CLASSES AND INTERFACE

Listener Interface	Methods
KeyListener	keyTyped() keyPressed() keyReleased()
MouseListener	mousePressed() mouseClicked() mouseEntered() mouseExited() mouseReleased()
MouseMotionListener	mouseMoved() mouseDragged()
MouseWheelListener	mouseWheelMoved()
TextListener	textChanged()
WindowListener	windowActivated() windowDeactivated() windowOpened() windowClosed() windowClosing() windowIconified() windowDeiconified()

JAVA ACTION LISTENER

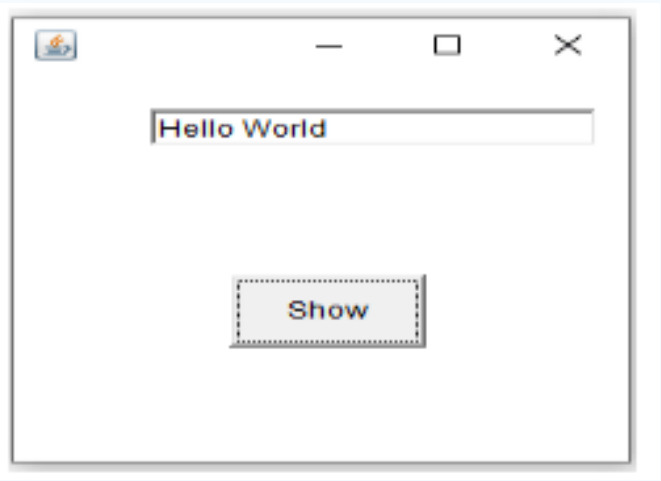
The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

EVENT HANDLING BUTTON

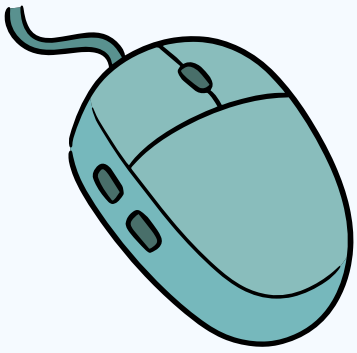
```
import java.awt.*;
import java.awt.event.*;

class EventHandling extends Frame implements ActionListener
{
    TextField textField;

    EventHandling ()
    {
        textField = new TextField ();
        textField.setBounds (60, 50, 170, 20);
        Button button = new Button ("Show");
        button.setBounds (90, 140, 75, 40);

        button.addActionListener (this);
        add (button);
        add (textField);
        setSize (250, 250);
        setLayout (null);
        setVisible (true);
    }


    public void actionPerformed (ActionEvent e)
    {
        textField.setText ("Hello World");
    }

    public static void main (String args[])
    {
        new EventHandling ();
    }
}
```



JAVA MOUSE LISTENER

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package, and it has three methods.

Method name	Description
public abstract void keyPressed (KeyEvent e);	It is invoked when a key has been pressed.
public abstract void keyReleased (KeyEvent e);	It is invoked when a key has been released.
public abstract void keyTyped (KeyEvent e);	It is invoked when a key has been typed.



EXAMPLE JAVA CODE

EVENT HANDLING

EXAMPLE MOUSE LISTENER

```
// importing awt libraries
import java.awt.*;
import java.awt.event.*;
// class which inherits Frame class and implements KeyListener
interface
public class KeyListenerExample extends Frame implements
KeyListener {
// creating object of Label class   and TextArea class
 Label l;
    TextArea area;
// class constructor
    KeyListenerExample() {
          // creating the label
        l = new Label();
// setting the location of the label in frame
        l.setBounds (20, 50, 100, 20);
// creating the text area
        area = new TextArea();
// setting the location of text area
        area.setBounds (20, 80, 300, 300);
// adding the KeyListener to the text area
        area.addKeyListener(this);
// adding the label and text area to the frame
        add(l);
add(area);
// setting the size, layout and visibility of frame
        setSize (400, 400);
        setLayout (null);
        setVisible (true);
    }
```

```
// overriding the keyPressed() method of KeyListener
interface where we set the text of the label when key
is pressed
    public void keyPressed (KeyEvent e) {
        l.setText ("Key Pressed");
    }
// overriding the keyReleased() method of KeyListener
interface where we set the text of the label when key
is released
    public void keyReleased (KeyEvent e) {
        l.setText ("Key Released");
    }
// overriding the keyTyped() method of KeyListener
interface where we set the text of the label when a key
is typed
    public void keyTyped (KeyEvent e) {
        l.setText ("Key Typed");
    }
  // main method
    public static void main(String[] args) {
        new KeyListenerExample();
    }
}
```

ITEM LISTENER

The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().

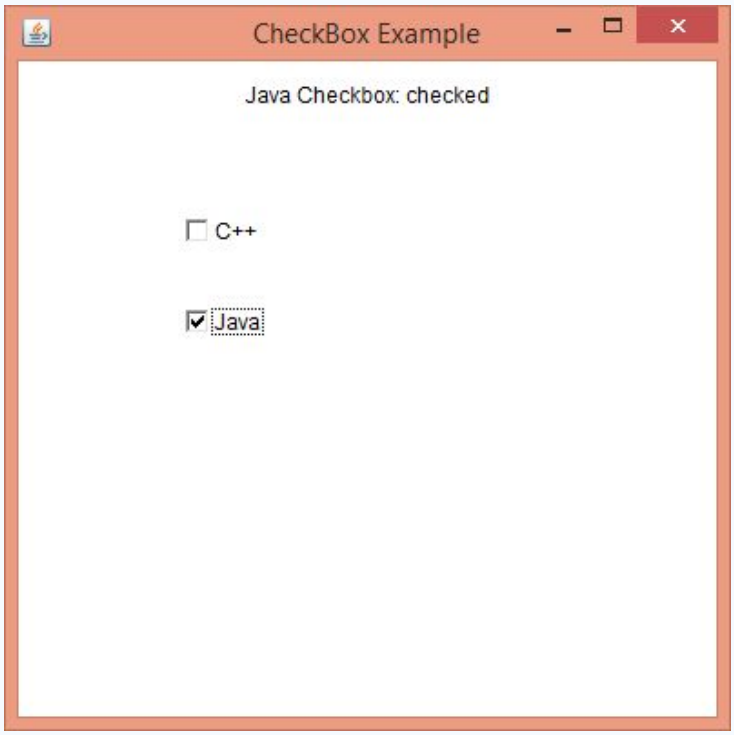
itemStateChanged() method

The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

EXAMPLE ITEM LISTENER

```
import java.awt.*;
import java.awt.event.*;
public class ItemListenerExample implements ItemListener{
    Checkbox checkBox1,checkBox2;
    Label label;
    ItemListenerExample(){
        Frame f= new Frame("CheckBox Example");
        label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        checkBox1 = new Checkbox("C++");
        checkBox1.setBounds(100,100, 50,50);
        checkBox2 = new Checkbox("Java");
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
        checkBox1.addItemListener(this);
        checkBox2.addItemListener(this);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
    public void itemStateChanged(ItemEvent e) {
        if(e.getSource()==checkBox1)
            label.setText("C++ Checkbox:+
(e.getStateChange()==1?"checked":"unchecked"));
        if(e.getSource()==checkBox2)
        label.setText("Java Checkbox: "
        + (e.getStateChange()==1?"checked":"unchecked"));
     }
public static void main(String args[])
{
    new ItemListenerExample();
}
}
```

EXAMPLE ITEM LISTENER



END OF CHAPTER 3

Chapter 4

JAVA DATABASE CONNECTIVITY

JAVA DATABASE CONNECTIVITY

JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction(API or Protocol) for java applications to communicate with various databases.

It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets. The enterprise data stored in a relational database(RDB) can be accessed with the help of JDBC APIs.

JAVA DATABASE CONNECTIVITY ARCHITECTURE

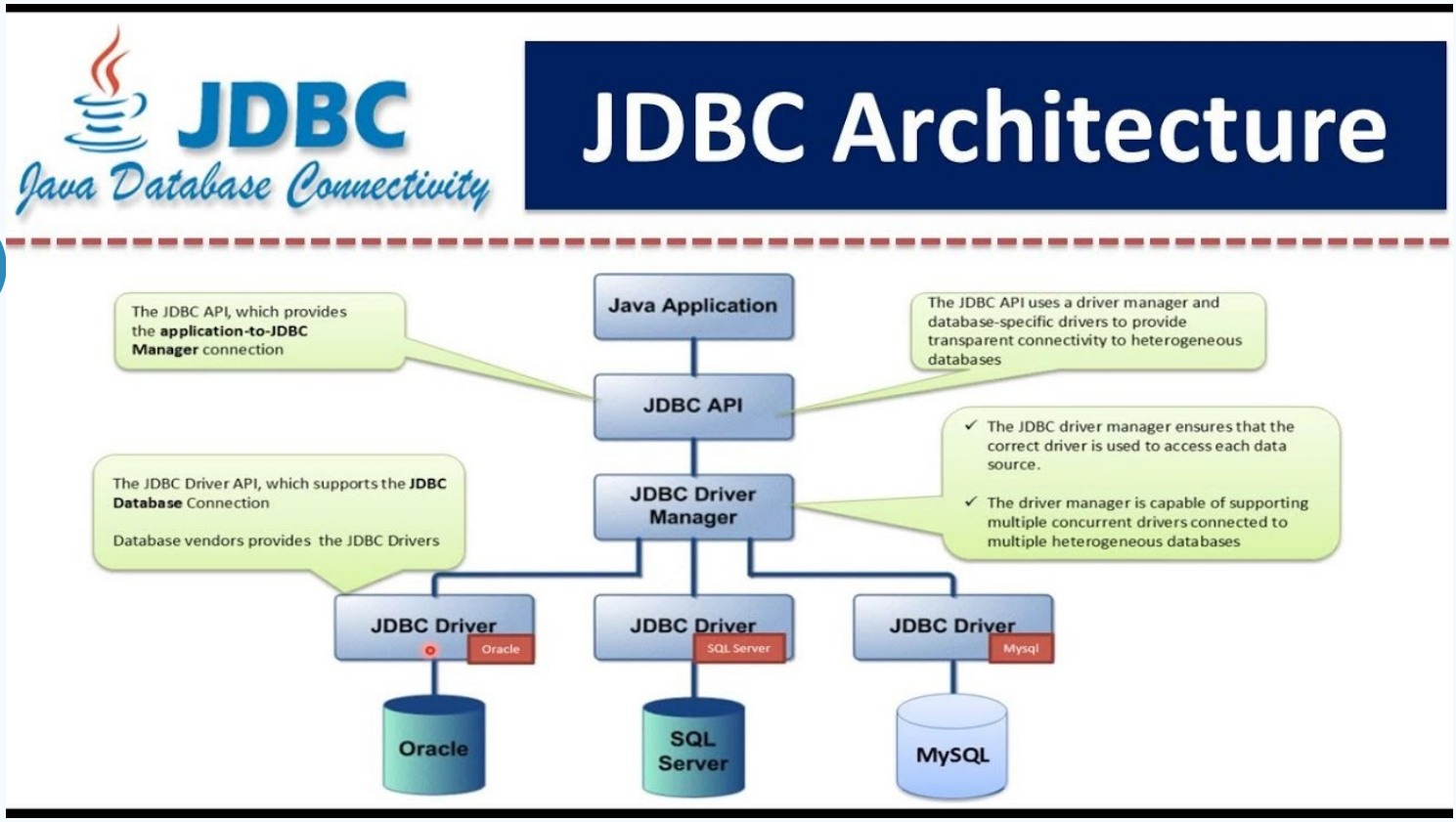


Figure 4.1 : JDBC Architecture
Source : http://ramj2ee.blogspot.com/2014/07/jdbc-architecture.htmlJDBC Architecture.JavaEE

HOW JDBC WORKS

JDBC to interact with a database from within a Java program. JDBC acts as a bridge from your code to the database

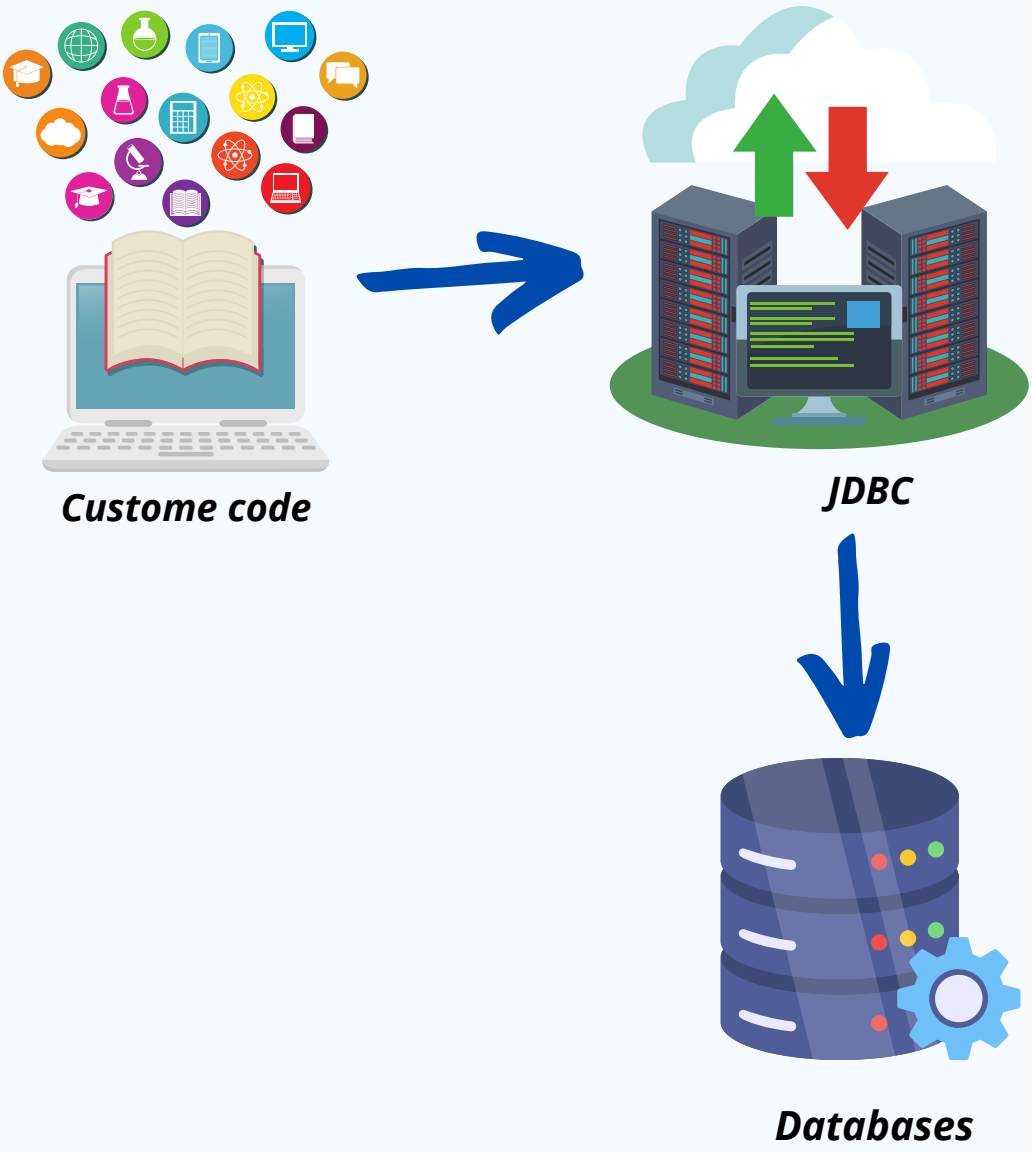


Figure 4.2 : JDBC connects Java programs to databases.

JDBC - ODBC

- Before JDBC, developers used Open Database Connectivity (ODBC)
- A language-agnostic standard approach to accessing a relational database management system, or RDBMS.
- In some ways, JDBC takes its inspiration from ODBC.
- The difference is that JDBC is Java-specific, offering a programming-level interface that handles the mechanics of Java applications communicating with a database.

JDBC'S ARCHITECTURE

The JDBC interface consists of two layers:

- The JDBC API supports communication between the Java application and the JDBC manager.
- The JDBC driver supports communication between the JDBC manager and the database driver.

JDBC DRIVERS

TYPE 1 **JDBC-ODBC BRIDGE DRIVER**

A thin Java layer that uses an ODBC driver under the hood.

NATIVE API DRIVER **TYPE 2**

Provides an interface from Java to the native database client.

TYPE 3 **MIDDLEWARE DRIVER**

A universal interface (“middleware”) between Java and the RDBMS’s vendor-specific protocol.

PURE JAVA DRIVER **TYPE 4**

A driver that implements the vendor-specific protocol directly in Java.

JDBC-ODBC BRIDGE DRIVER

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

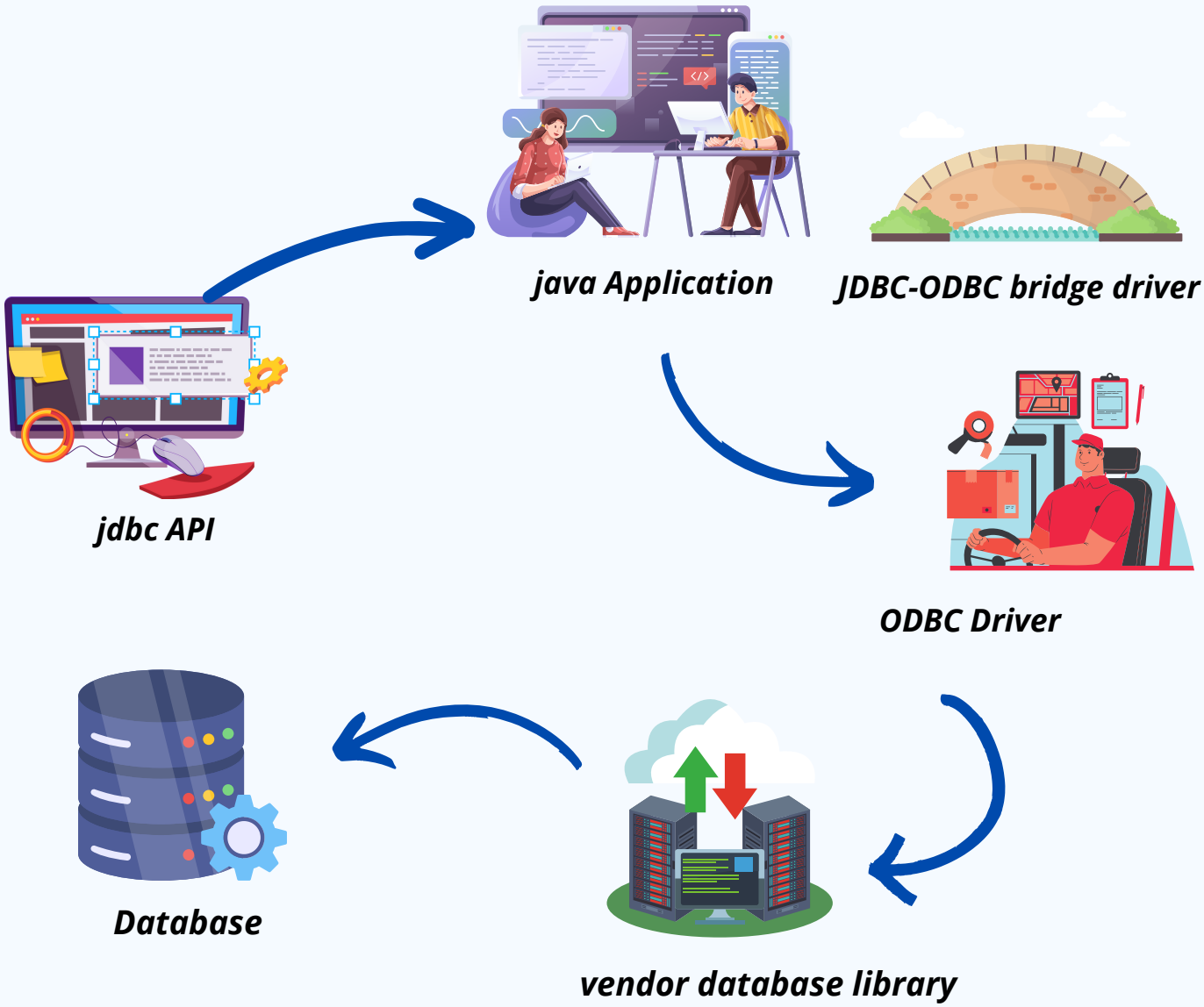


Figure 4.3 : JDBC-ODBC bridge driver

JDBC-ODBC BRIDGE DRIVER

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.

- As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.
- The ODBC bridge driver is needed to be installed in individual client machines.
- Type-1 driver isn't written in java, that's why it isn't a portable driver.
- This driver software is built-in with JDK so no need to install separately.
- It is a database independent driver.

NATIVE-API DRIVER

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

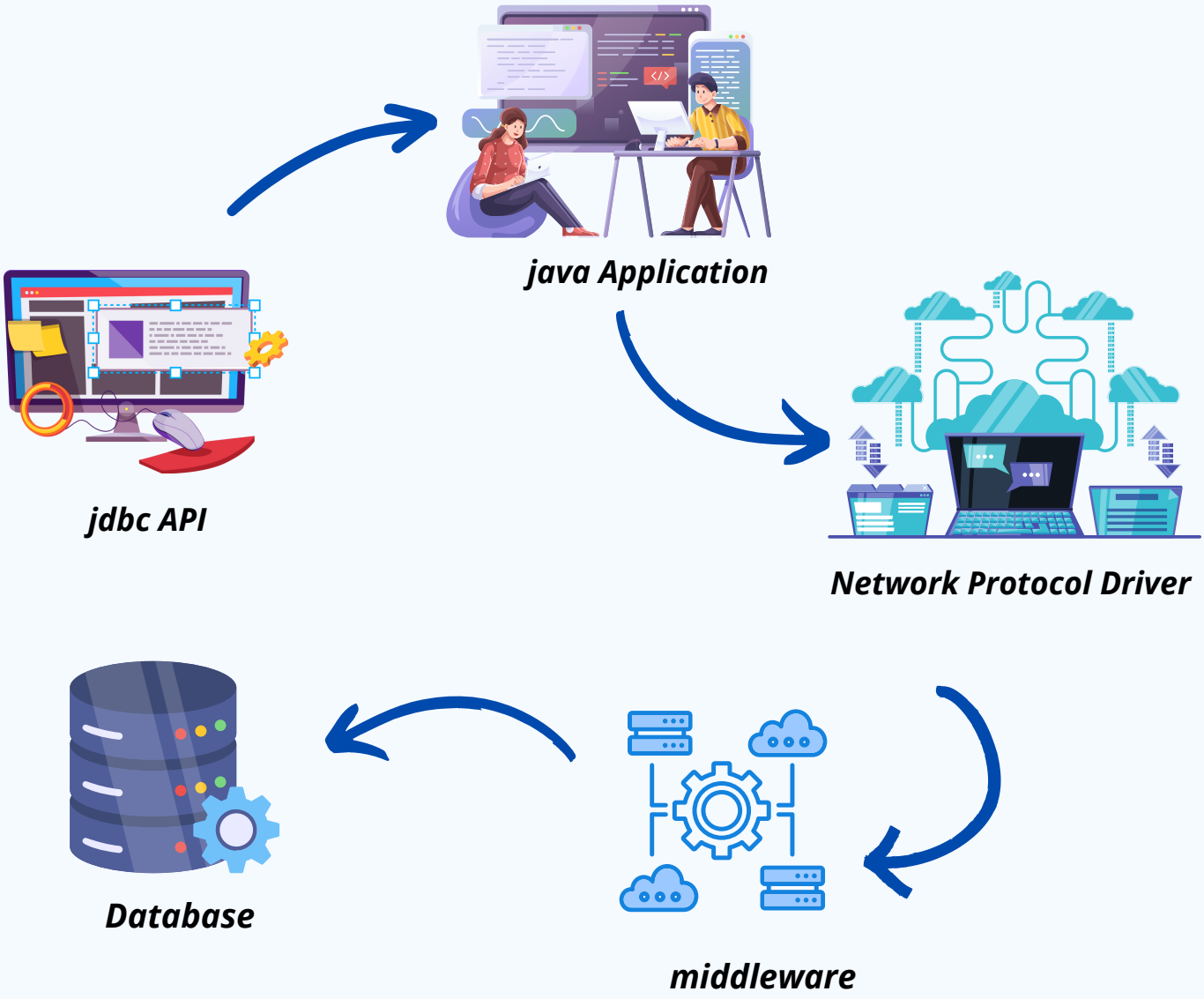


Figure 4.5 : Native API driverr

NATIVE-API DRIVER

The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.

- Driver needs to be installed separately in individual client machines
- The Vendor client library needs to be installed on client machine.
- Type-2 driver isn't written in java, that's why it isn't a portable driver
- It is a database dependent driver.

NETWORK PROTOCOL DRIVER

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

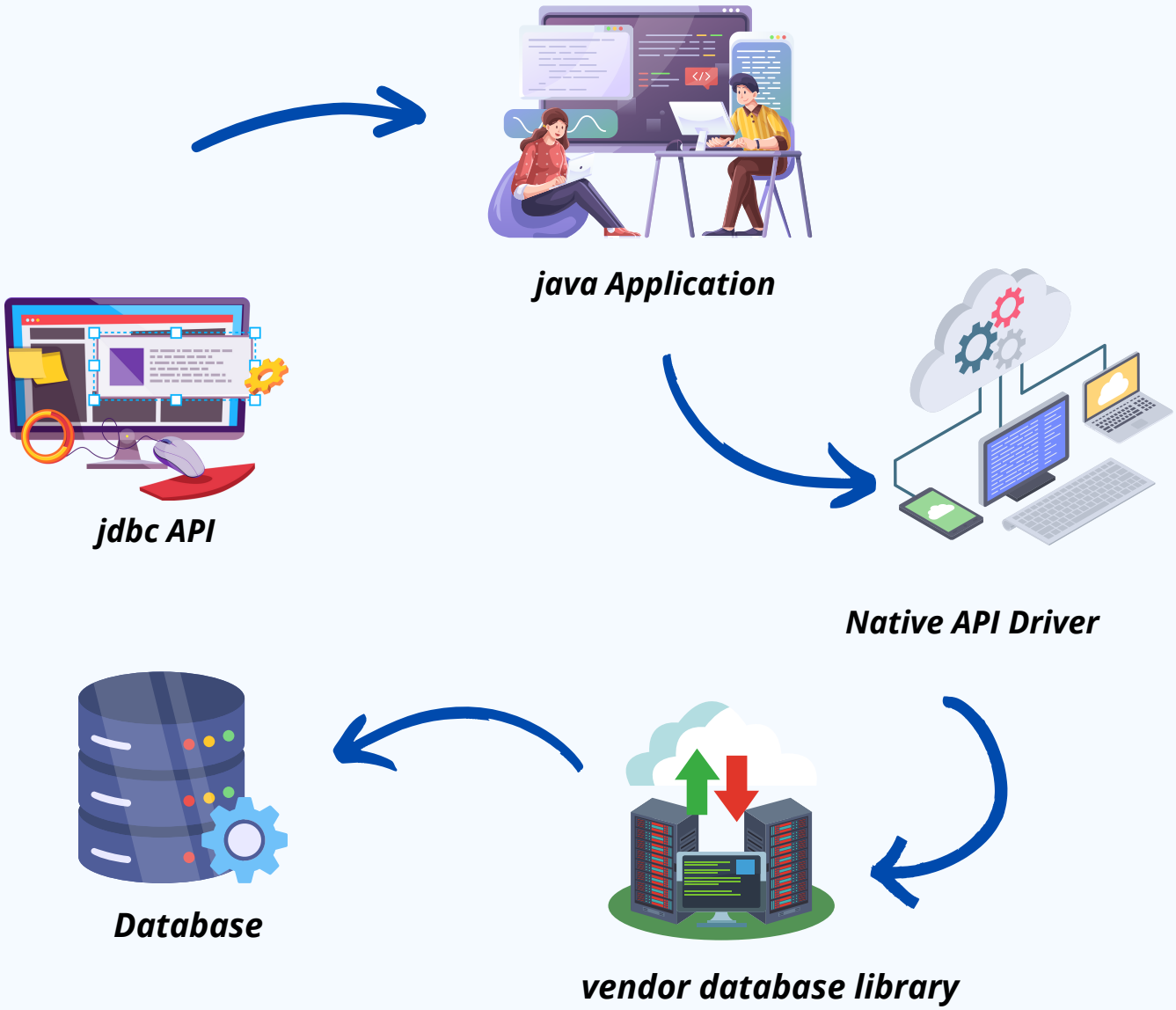


Figure 4.4 : Network Protocol driverr

NETWORK PROTOCOL DRIVER

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.

- Type-3 drivers are fully written in Java, hence they are portable drivers.
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- Network support is required on client machine.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.
- Switch facility to switch over from one database to another database.

THIN DRIVER

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

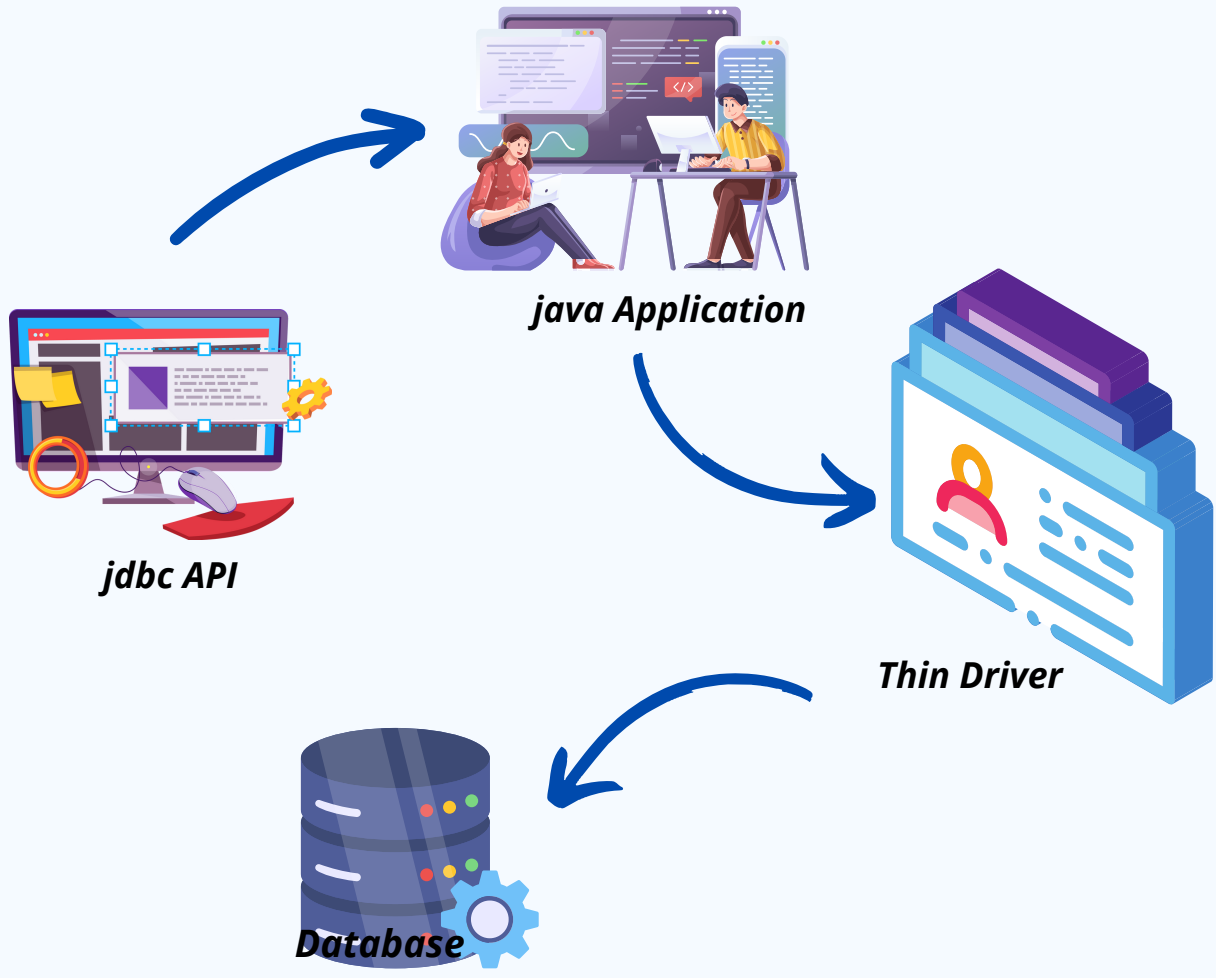


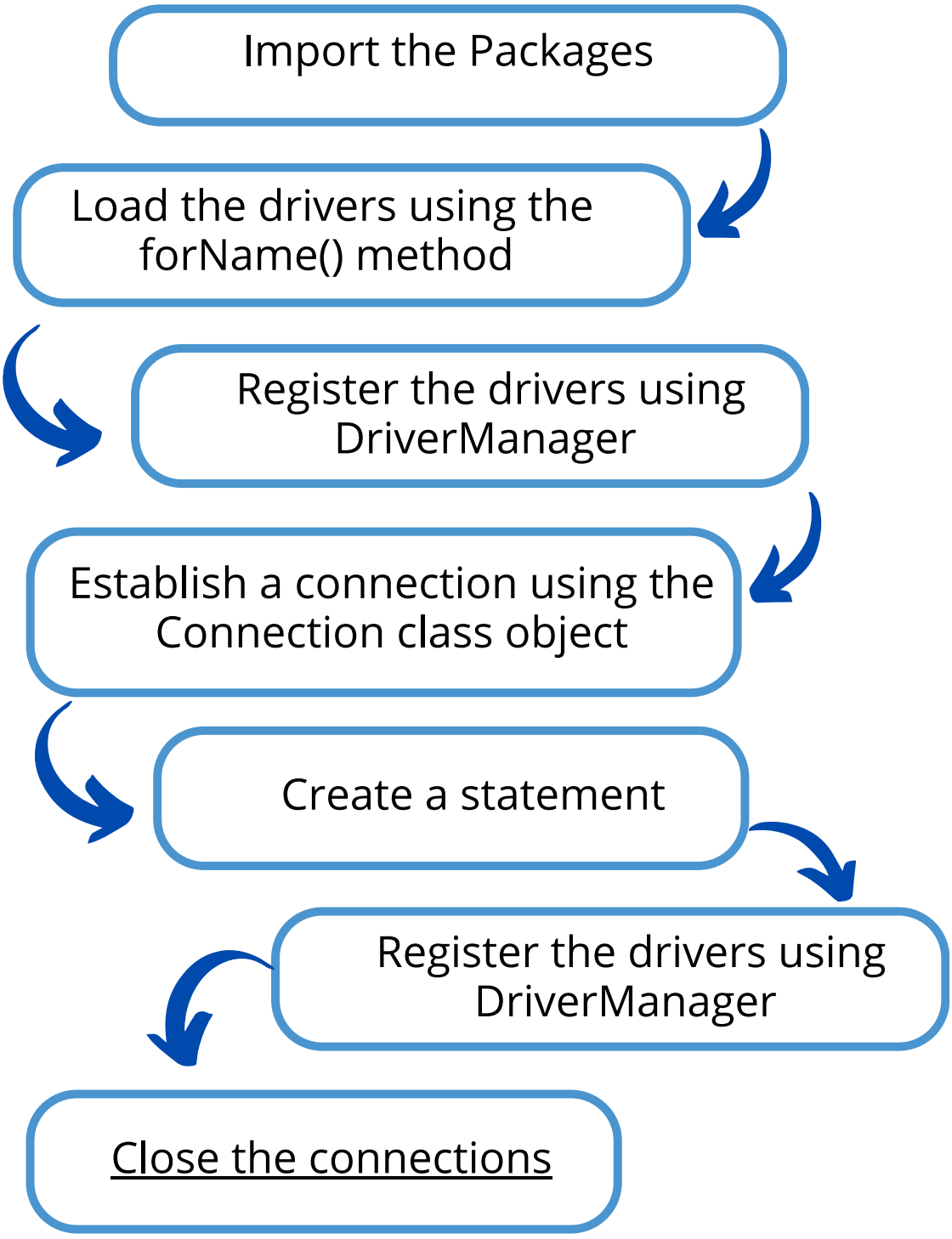
Figure 4.5 : Native API driverr

THIN DRIVER

Type-4 driver is also called native protocol driver. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

- Does not require any native library and Middleware server, so no client-side or server-side installation.
- It is fully written in Java language, hence they are portable drivers.

STEPS FOR CONNECTIVITY BETWEEN JAVA PROGRAM AND DATABASE



STEP 1	Import the Packages
STEP 2	Loading the drivers

In order to begin with, you first need to load the driver or register it before using it in the program. Registration is to be done once in your program. You can register a driver in one of two ways mentioned below as follows:

`Class.forName()`

Here we load the driver's class file into memory at the runtime. No need of using new or create objects. The following example uses Class.forName() to load the Oracle driver as shown below as follows:

Type 1

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Type 4

```
Class.forName("com.mysql.jdbc.Driver");
```

STEP 3

Establish a connection using the Connection class object

After loading the driver, establish connections as shown below as follows:

```
Connection con =
DriverManager.getConnection(url,user,password)
```

- user: Username from which your SQL command prompt can be accessed.
- password: password from which the SQL command prompt can be accessed.
- con: It is a reference to the Connection interface.
- Url: Uniform Resource Locator which is created as shown below:

```
String url=jdbc:oracle:thin:@localhost:1521:xe
```

Type 1

```
con = DriverManager.getConnection
("jdbc:odbc:Semester6", "abcd", "1234");
```

Type 4

```
con = DriverManager.getConnection
(“jdbc:mysql://localhost:3306/pelajar“,“root",
“1234");
```

STEP 3

Establish a connection using the Connection class object

After loading the driver, establish connections as shown below as follows:

```
Connection con =
DriverManager.getConnection(url,user,password)
```

- user: Username from which your SQL command prompt can be accessed.
- password: password from which the SQL command prompt can be accessed.
- con: It is a reference to the Connection interface.
- Url: Uniform Resource Locator which is created as shown below:

```
String url=jdbc:oracle:thin:@localhost:1521:xe
```

Type 1

```
con = DriverManager.getConnection
("jdbc:odbc:Semester6", "abcd", "1234");
```

Type 4

```
con = DriverManager.getConnection
(“jdbc:mysql://localhost:3306/pelajar“,“root",
“1234");
```

STEP 4

Create JDBC statement(s)

Three (3) types JDBC Statements

Statement

- used to execute simple SQL query without parameters.

PreparedStatement

- It is a precompiled SQL statement.
- Use the SQL statements many times.
- Check and compile the column, table, syntax of the SQL, re-executed without repeating these steps.
- Accepts input parameters at runtime.

CallableStatement

- It is use to call a database stored procedure.
- The stored procedure contains the SQL query to be executed on the database and is stored on the database.

Example Three (3) types JDBC Statements

Statement

```
import java.sql.Statement;
Statement stmt = null;
stmt = con.createStatement( );
```

PreparedStatement

```
import java.sql.PreparedStatement
PreparedStatement prepare = null;
prepare = con.prepareStatement("UPDATE DBUSER
SET username = ? WHERE user_id = ?");
```

CallableStatement

```
String str = "CREATE PROCEDURE emp_info
 emp_id integer,
 fname varchar (20) output" + "AS" +
 "SELECT first_name = fname" +
 "WHERE empId = emp_id";
```

STEP 5

Execute the query

The query here is an SQL Query. Now we know we can have multiple types of queries. Some of them are as follows:

- The query for updating/inserting a table in a database.
- The query for retrieving data.

The executeQuery() method of the Statement interface is used to execute queries of retrieving values from the database. This method returns the object of ResultSet that can be used to get all the records of a table.
The executeUpdate(sql query) method of the Statement interface is used to execute queries of updating/inserting.

Example

```
String insertString1;
String nama = txtNama.getText();
int umur = Integer.parseInt(txtUmur.getText());
String jan = txtJan.getText();
insertString1 = “INSERT INTO student
VALUES('"+nama+"', "+umur+", '"+jan+"' )";
  try {
   stmt = con.createStatement();
      stmt.executeUpdate(insertString1);
        }
```

SQL statements

SELECT Statement

```
Stringsql="SELECT * FROM biodatapelajar";
```

INSERT Statement

```
String sql = "INSERT INTO biodatapelajar VALUES
('" + noMatrik + "','" + nama + "','"+ noIC +
"'," + semester + ", " + umur + " )";
```

UPDATE Statement

```
String sql = "UPDATE biodatapelajar SET
no_matrik='" + noMatrik + "',nama='" +
      nama + "',no_ic='" + noIC + "',"+
"semester=" + semester + ",umur=" + umur + "
    WHERE no_matrik='" + noMatrik + "'";
```

INSERT Statement

```
String sql = "DELETE from biodatapelajar WHERE
no_matrik=?"
```

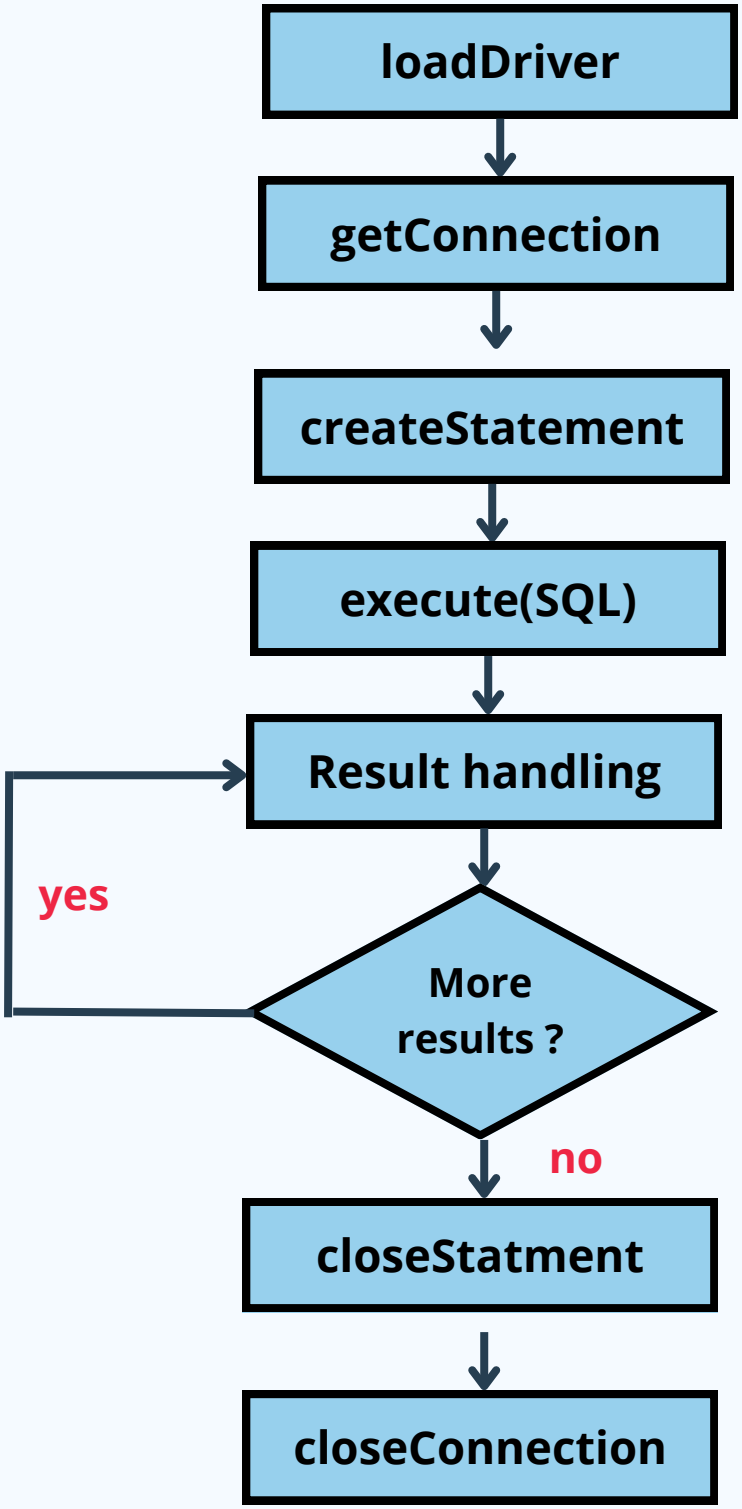
STEP 6

Closing the connections

So finally we have sent the data to the specified location and now we are on the verge of completing our task. By closing the connection, objects of Statement and ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection. It is shown below as follows:

```
stmt.close();
con.close();
```

FLOW JDBC APPLICATION



EXAMPLE SNIPCODE

```
import java.sql.*;
public class jdbctest {
  public static void main(String args[]){
 try{
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver “);
   Connection con = DriverManager.getConnection
       ("jdbc:odbc:pcmtable", "user", "passwd");
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery
       ("select name, number from table where number < 2");
   while(rs.next())
    System.out.println(rs.getString(1) + " (" + rs.getInt(2) + ")");
   stmt.close()
   con.close();
 } catch(Exception e){
   System.err.println(e);
}}}
```

END OF CHAPTER 4

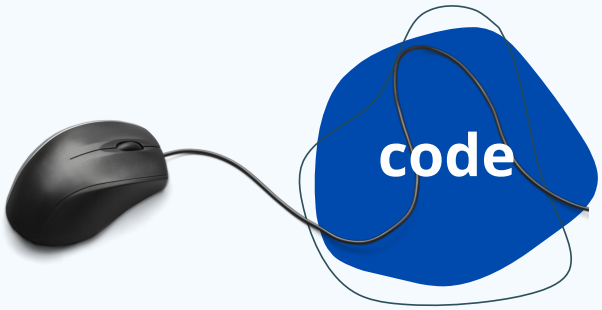
JAVA Lab Exercise

PN

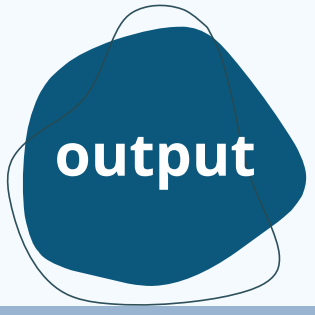
Lab Exercise

Chapter 1

Example 1- Basic Java Program

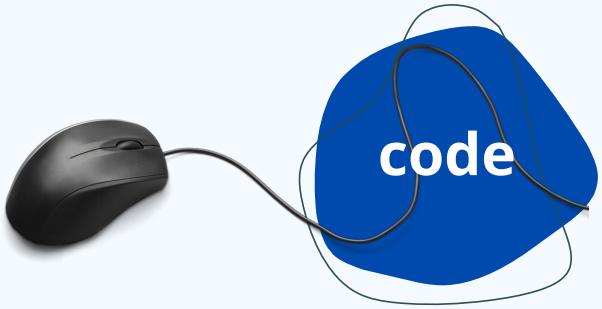



```
class BasicProgram {
    public static void main(String args[]) {
     System.out.println("Hello Java");
    }
}
```

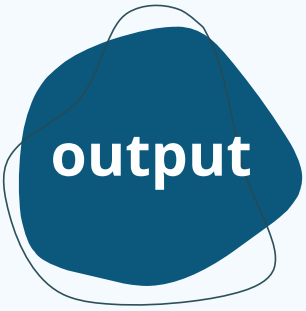


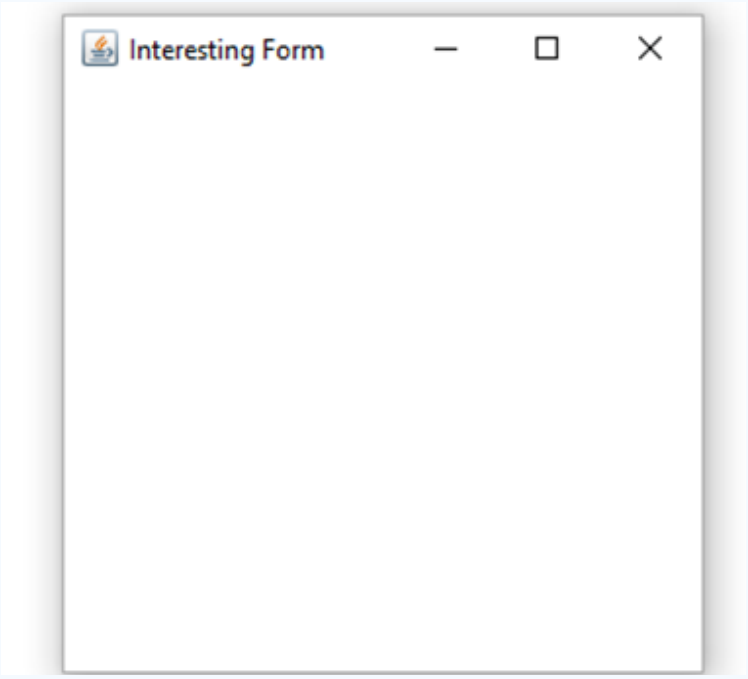

```
General Output
--------------------Configuration: <Default>-------------
Hello Java

Process completed.
```

Example 2- Create frame

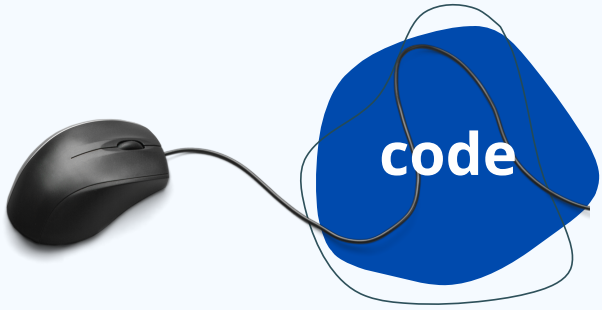



```
import java.awt.*;
class First2{
  First2( ){
    Frame f=new Frame("Interesting Form");
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
}
public static void main(String args[]){
  First2 f=new First2();
}}
```

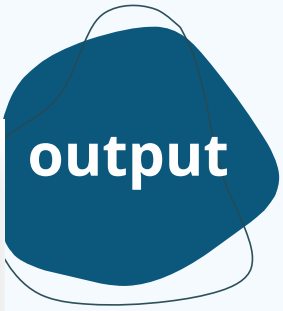


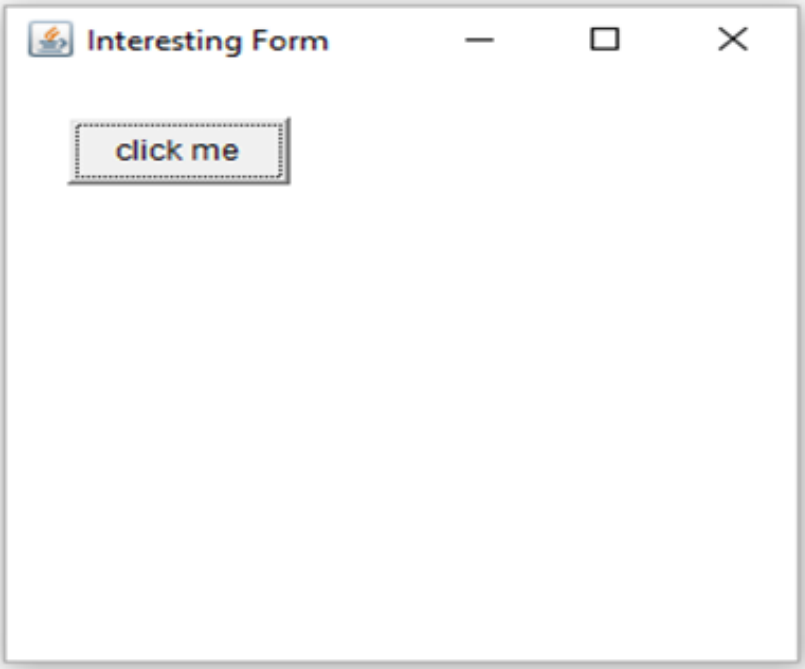


Example 3- Add Button Into Frame




```
import java.awt.*;
class First3{
First3(){
  Frame f=new Frame("Interesting Form");
  Button b=new Button("click me");
  b.setBounds(30,50,80,30); // setting button position
  f.add(b); //adding button into frame
  f.setSize(300,300); //frame size 300 width and 300 height
  f.setLayout(null); //no layout manager
  f.setVisible(true); /now frame will be visible,by default
not visible

}
 public static void main(String args[]){
   First3 f=new First3();
}}
```



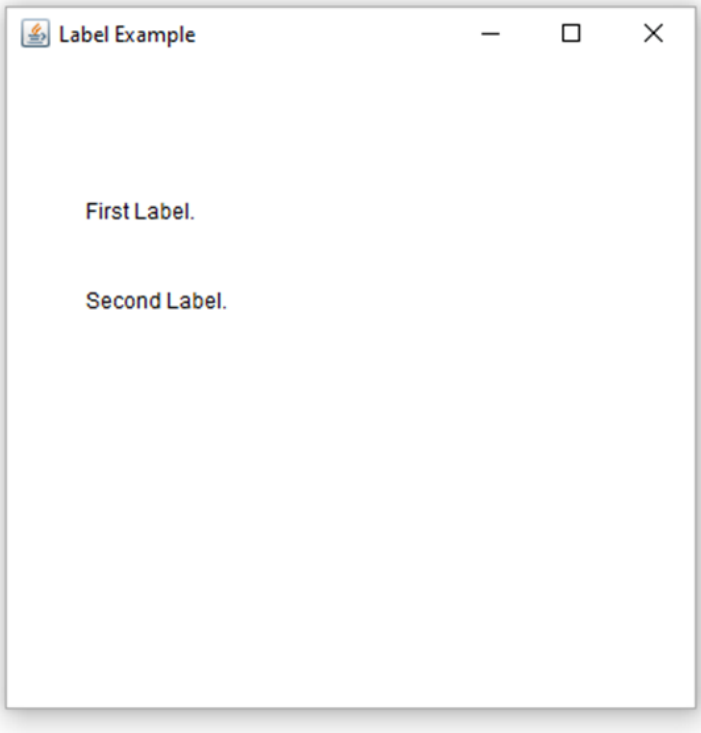


Example 4- Add Label Into Frame

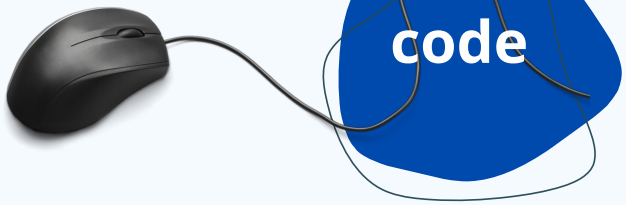
code

```
import java.awt.*;
class LabelExample{
public static void main(String args[]){
    Frame f= new Frame("Label Example");
    Label l1,l2;
    l1=new Label("First Label.");
    l1.setBounds(50,100, 100,30);
    l2=new Label("Second Label.");
    l2.setBounds(50,150, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

output

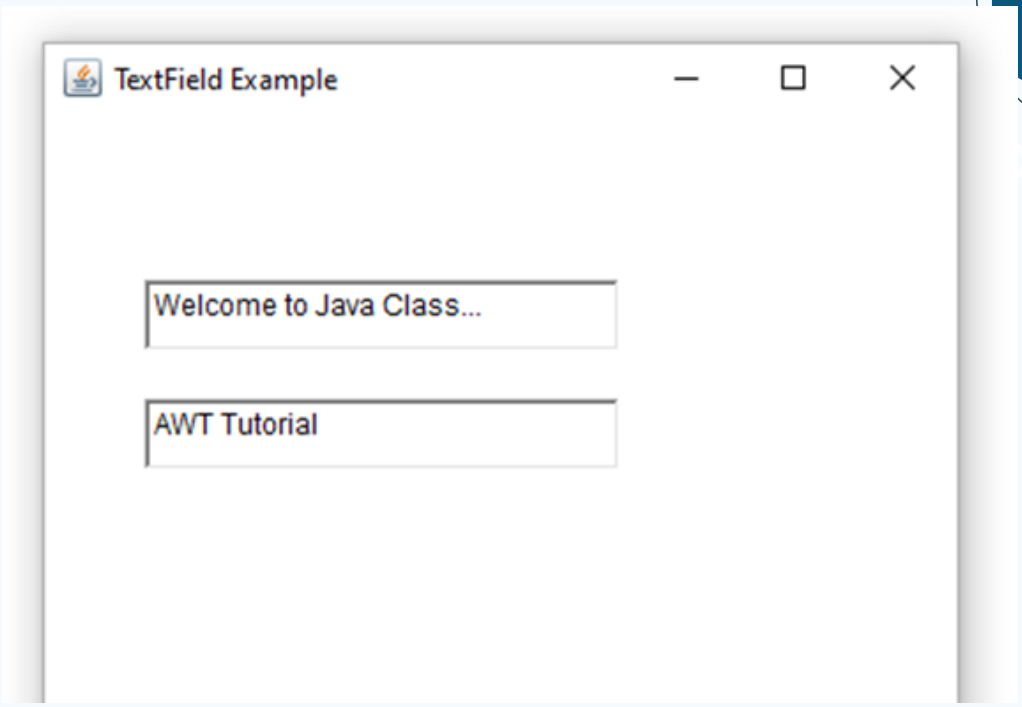


Example 5- Add TextField Into Frame



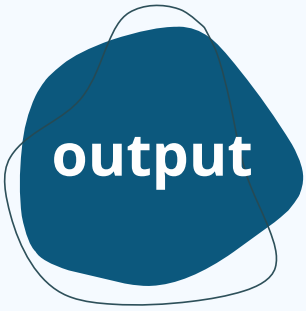

```
import java.awt.*;
class TextFieldExample{
public static void main(String args[]){
    Frame f= new Frame("TextField Example");
    TextField t1,t2;
    t1=new TextField("Welcome to Java Class.");
    t1.setBounds(50,100, 200,30);
    t2=new TextField("AWT Tutorial");
    t2.setBounds(50,150, 200,30);
    f.add(t1); f.add(t2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

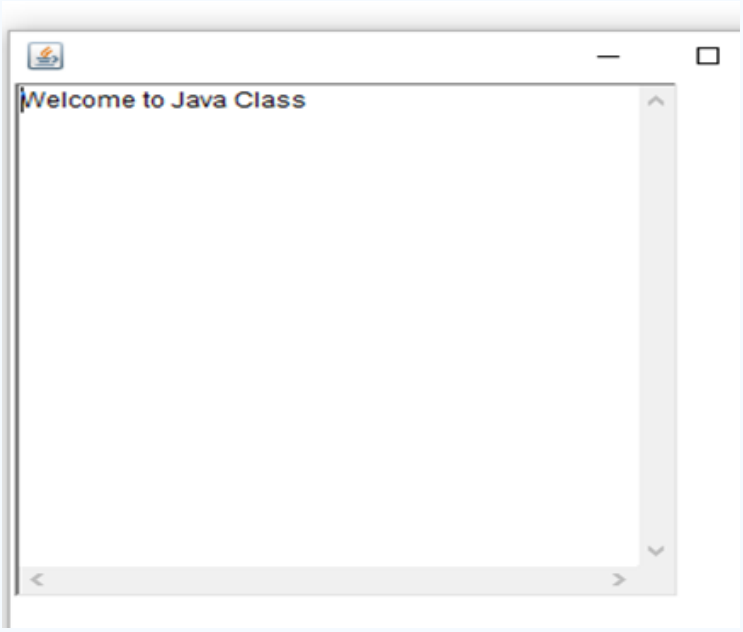
output



Example 6- Add TextArea Into Frame


```
import java.awt.*;
public class TextAreaExample
{
    TextAreaExample(){
        Frame f= new Frame();
        TextArea area=new TextArea("Welcome to
javatpoint");
       area.setBounds(10,30, 300,300);
       f.add(area);
       f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
public static void main(String args[])
{
   new TextAreaExample();
}
}
```



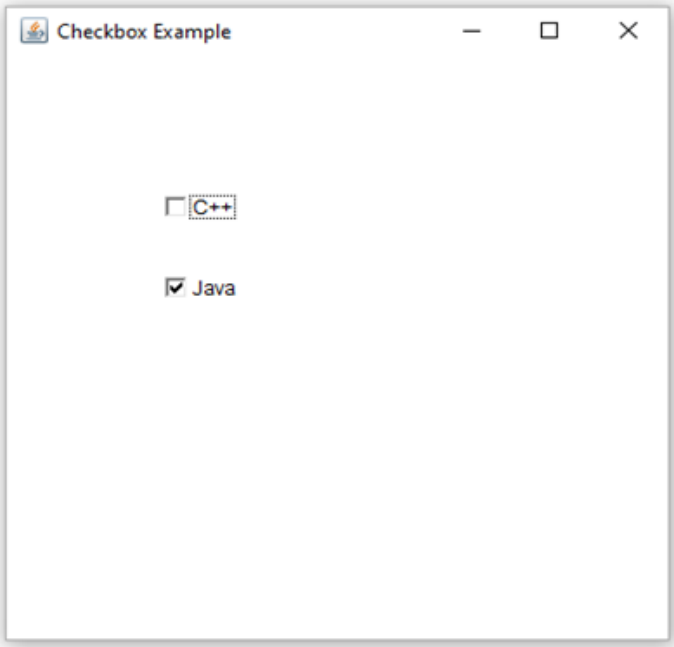


Example 7- Add CheckBox Into Frame

code

```
import java.awt.*;
public class CheckboxExample
{
      CheckboxExample(){
        Frame f= new Frame("Checkbox Example");
         Checkbox checkbox1 = new Checkbox("C++");
         checkbox1.setBounds(100,100, 50,50);
         Checkbox checkbox2 = new Checkbox("Java",
true);
         checkbox2.setBounds(100,150, 50,50);
         f.add(checkbox1);
         f.add(checkbox2);
         f.setSize(400,400);
         f.setLayout(null);
         f.setVisible(true);
      }
public static void main(String args[])
{
    new CheckboxExample();
}
}
```

output

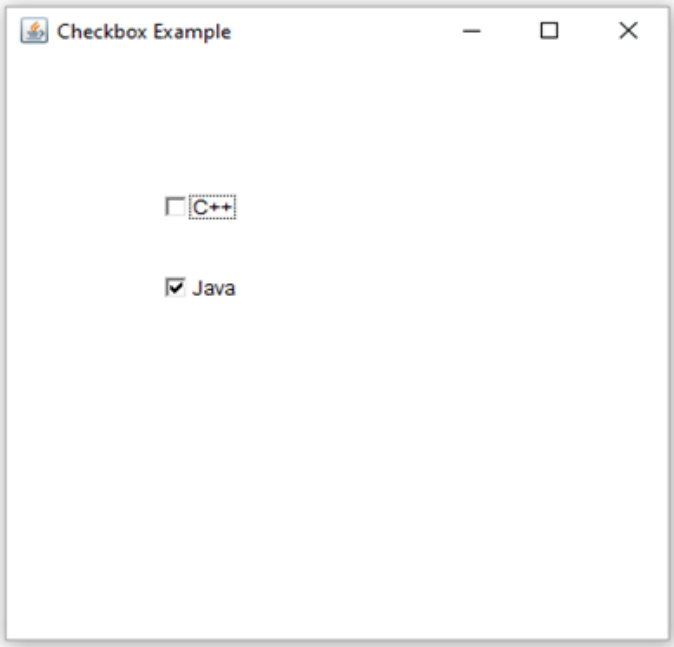


Example 7- Add CheckBox Into Frame

code

```
import java.awt.*;
public class CheckboxExample
{
      CheckboxExample(){
        Frame f= new Frame("Checkbox Example");
         Checkbox checkbox1 = new Checkbox("C++");
         checkbox1.setBounds(100,100, 50,50);
         Checkbox checkbox2 = new Checkbox("Java",
true);
         checkbox2.setBounds(100,150, 50,50);
         f.add(checkbox1);
         f.add(checkbox2);
         f.setSize(400,400);
         f.setLayout(null);
         f.setVisible(true);
      }
public static void main(String args[])
{
    new CheckboxExample();
}
}
```

output



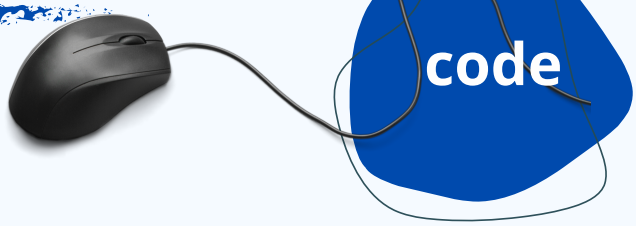
Example 8- Add List Into Frame

code

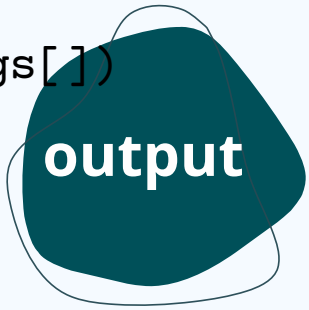
```
import java.awt.*;
public class ListExample
{
      ListExample(){
          Frame f= new Frame();
          List l1=new List(5);
          l1.setBounds(100,100, 75,75);
          l1.add("Item 1");
          l1.add("Item 2");
          l1.add("Item 3");
          l1.add("Item 4");
          l1.add("Item 5");
          f.add(l1);
          f.setSize(400,400);
          f.setLayout(null);
          f.setVisible(true);
      }
public static void main(String args[])
{
    new ListExample();
}
}
```

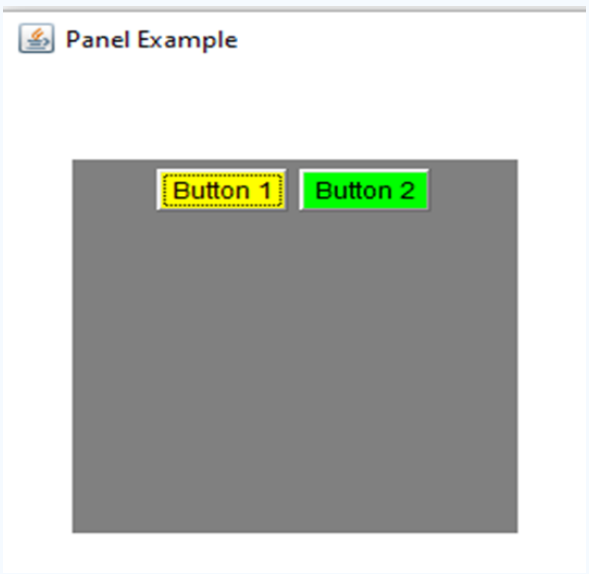
output

Example 9 - Add Panel Into Frame




```
ort java.awt.*;
blic class PanelExample {
    PanelExample()
       {
        Frame f= new Frame("Panel Example");
      Panel panel=new Panel();
      panel.setBounds(40,80,200,200);
      panel.setBackground(Color.gray);
     Button b1=new Button("Button 1");
      b1.setBounds(50,100,80,30);
       b1.setBackground(Color.yellow);
       Button b2=new Button("Button 2");
       b2.setBounds(100,100,80,30);
       b2.setBackground(Color.green);
       panel.add(b1); panel.add(b2);
       f.add(panel);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
       }
       public static void main(String args[])
      {
      new PanelExample();
       }
}
```



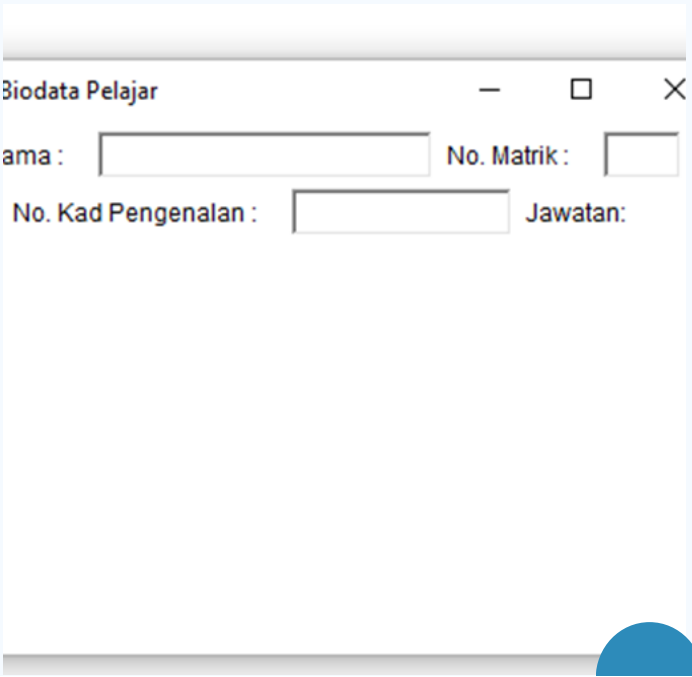


Example 10 - Add AWT Component

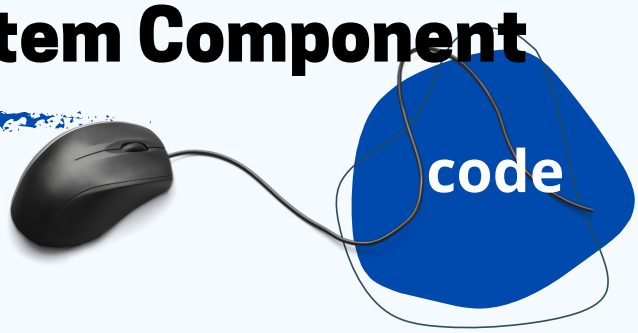
code

```
import java.awt.*;
class BiodataPelajar extends Frame
{
   public static void main(String[] args)
   {
     Frame f= new Frame("Biodata Pelajar");
     f.setSize(400,300);
     f.setVisible(true);
     Panel p = new Panel();
     Label xLabel=new Label("Nama :");
     TextField tname=new TextField(20);
     Label yLabel=new Label("No. Matrik :");
     TextField tstaf=new TextField(2);
     Label zLabel=new Label("No. Kad Pengenalan :");
     TextField tkp=new TextField(12);
     Label aLabel=new Label("Jawatan:");
     TextField tjawatan=new TextField(15);
     p.add(xLabel);p.add(tname);p.add(yLabel);
     p.add(tstaf);p.add(zLabel);p.add(tkp);
     p.add(aLabel);
     f.add(p);
}
}
```

output



xample 11 - Add Menu Item Component



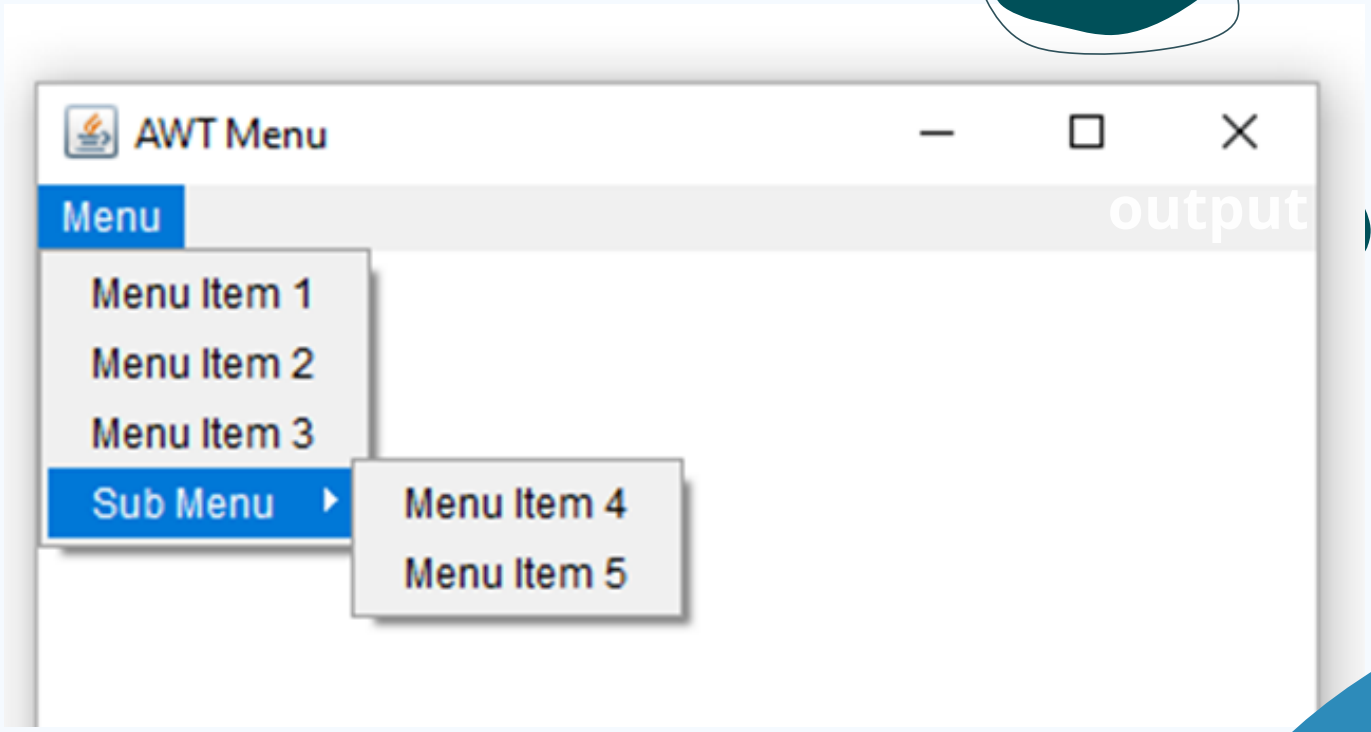

```
rt java.awt.*;
ss AWTMenu extends Frame

MenuBar mbar; Menu menu,submenu;
MenuItem m1,m2,m3,m4,m5;

   public AWTMenu()
   {
    // Set frame properties
   setTitle("AWT Menu"); // Set the title
   setSize(400,400);     // Set size to the frame
   setLayout(new FlowLayout()); // Set the layout
   setVisible(true); // Make the frame visible
   setLocationRelativeTo(null); // Center the frame

   mbar=new MenuBar(); // Create the menu bar
   menu=new Menu("Menu");// Create the menu
   submenu=new Menu("Sub Menu");// Create the submenu

// Create MenuItems
   m1=new MenuItem("Menu Item 1");
   m2=new MenuItem("Menu Item 2");
   m3=new MenuItem("Menu Item 3");
   m4=new MenuItem("Menu Item 4");
   m5=new MenuItem("Menu Item 5");
   // Attach menu items to menu
   menu.add(m1);
   menu.add(m2);
   menu.add(m3);
```

Example 11 - Add Menu Item Component (Cont..)

code

```
        // Attach menu items to submenu
        submenu.add(m4);
        submenu.add(m5);

        menu.add(submenu); // Attach submenu to menu
        mbar.add(menu);    // Attach menu to menu bar
        // Set menu bar to the frame
        setMenuBar(mbar);
    }

    public static void main(String args[])
    {
    new AWTMenu();
    }
}
```

output



Lab Exercise

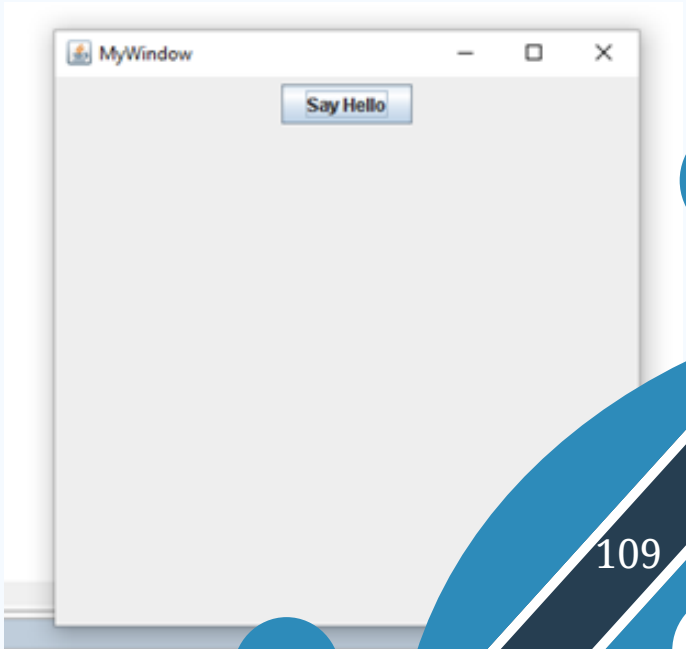
Chapter 2

Example 1 - Add Button using Swing Component

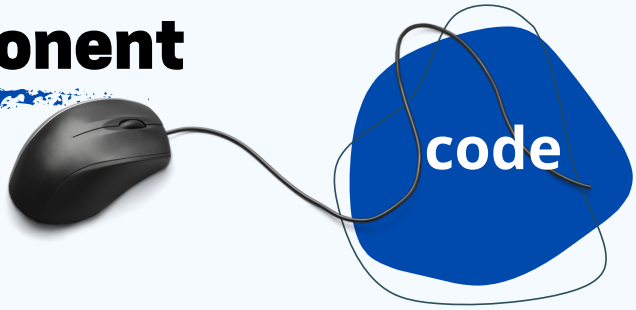
code

```
import javax.swing.*; //importing swing package
import java.awt.*;    //importing awt package
public class FirstSwing
{
 public FirstSwing() {
 JFrame jf = new JFrame("MyWindow"); //Creating a JFrame
with name MyWindow
 JButton btn = new JButton("Say Hello"); //Creating a
Button named Say Hello
 jf.add(btn);
//adding button to frame
 jf.setLayout(new FlowLayout());
//setting layout using FlowLayout object
 jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//setting close operation.
 jf.setSize(400, 400);
//setting size
 jf.setVisible(true);
//setting frame visibility
}
 public static void main(String[] args)
 {
 new FirstSwing();
 }
}
```

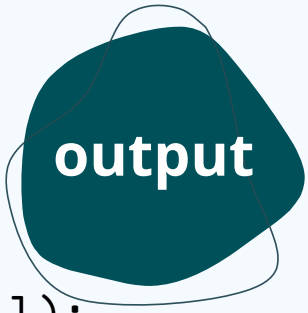
output



Example 2 - Add Button and Label using Swing Component

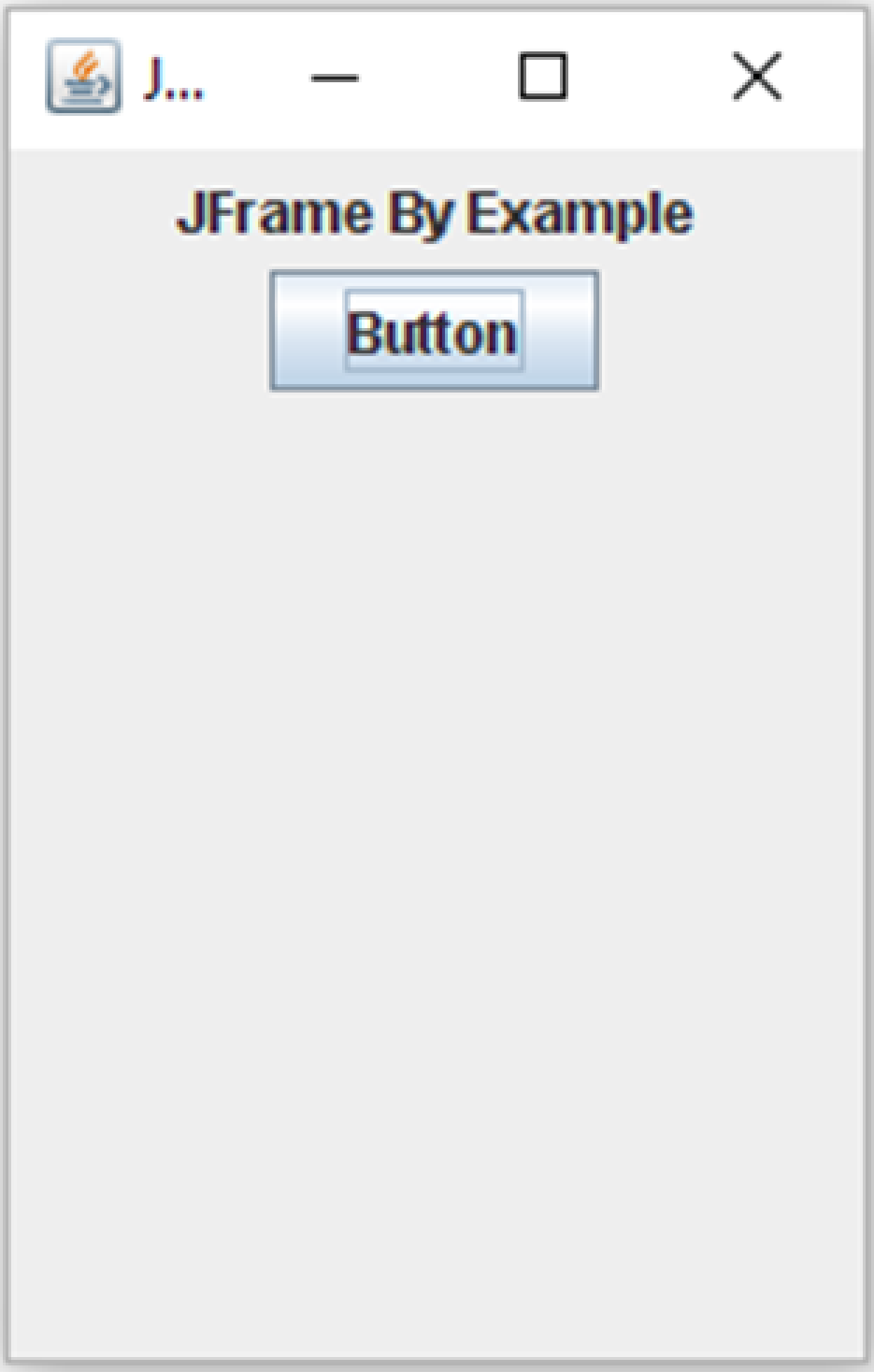



```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class JFrameExample
{
    public static void main(String s[])
    {
      JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JLabel label = new JLabel("JFrame By
Example");
        JButton button = new JButton();
        button.setText("Button");
        panel.add(label);
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
;
        frame.setVisible(true);
    }
```



Example 2 - Add Button and Label using Swing Component ..Output

output

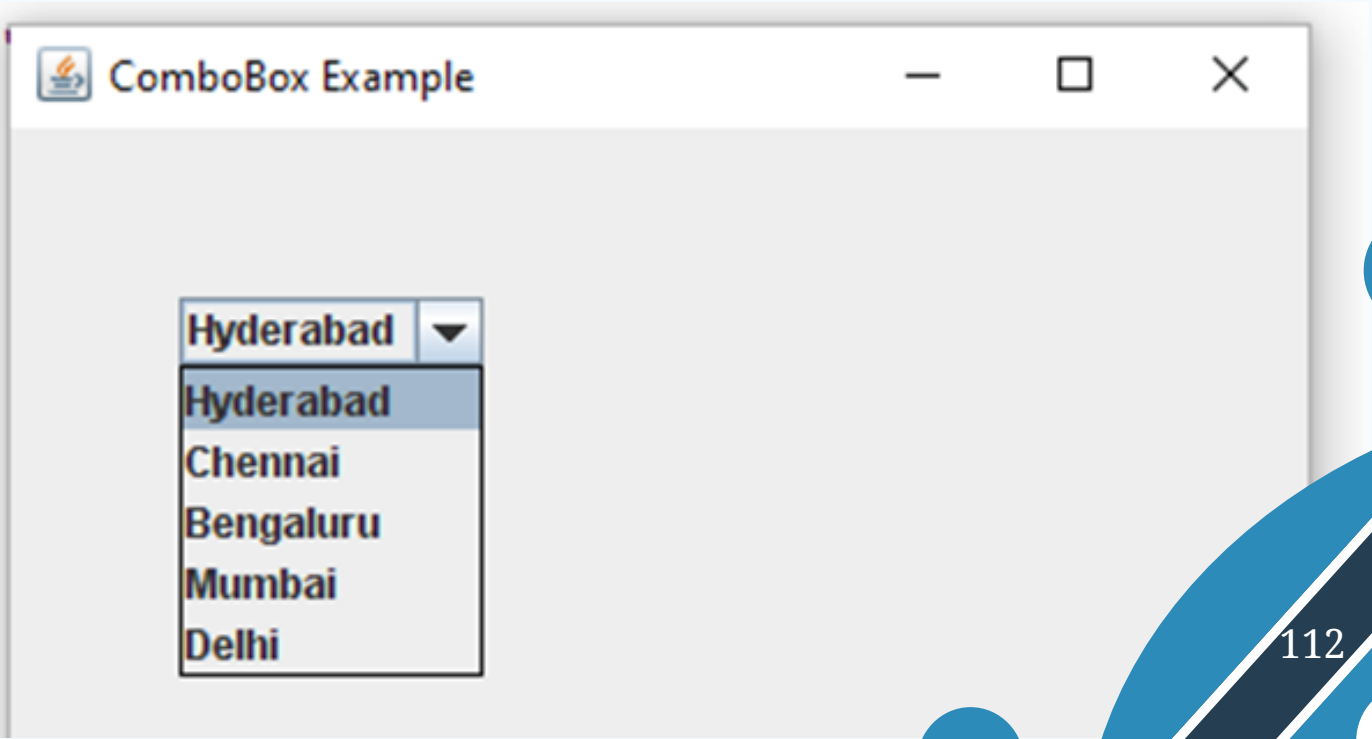


Example 3- Add ComboBox into frame using Swing Component

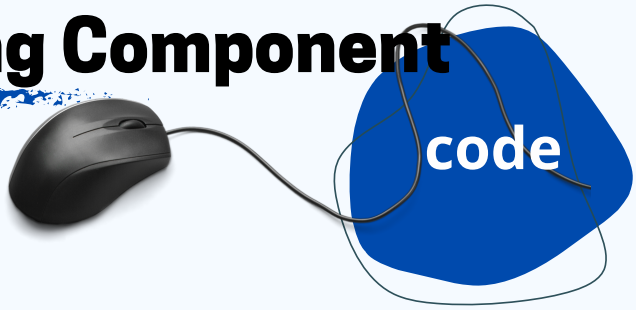
code

```
import javax.swing.*;
public class JComboBoxDemo
{
JFrame f;
JComboBoxDemo ()
{
f = new JFrame ("ComboBox Example");
String country[] ={ "Hyderabad", "Chennai", "Bengaluru"
"Mumbai", "Delhi" };
JComboBox cb = new JComboBox (country);
cb.setBounds (50, 50, 90, 20);
f.add (cb);
f.setLayout (null);
f.setSize (400, 500);
f.setVisible (true);
}
public static void main (String[]args)
{
new JComboBoxDemo ();
}
}
```

output



Example 4 - Add CheckBox Menu Item into frame using Swing Component



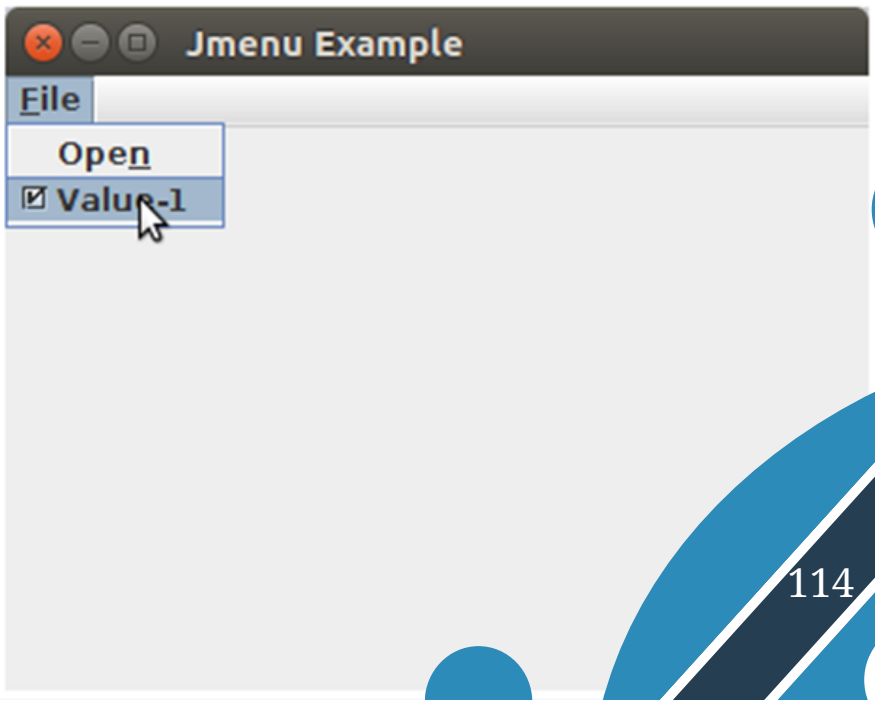

```
import javax.swing.*;
import java.awt.event.*;

public class JavaCheckBoxMenuItem {
    public static void main(final String args[]) {
    JFrame frame = new JFrame("Jmenu Example");

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JMenuBar menuBar = new JMenuBar();
    // File Menu, F - Mnemonic
    JMenu fileMenu = new JMenu("File");
    fileMenu.setMnemonic(KeyEvent.VK_F);
    menuBar.add(fileMenu);
    // File->New, N - Mnemonic
    JMenuItem menuItem1 = new JMenuItem("Open",
KeyEvent.VK_N);
    fileMenu.add(menuItem1);
    JCheckBoxMenuItem caseMenuItem = new
JCheckBoxMenuItem("Option_1");
    caseMenuItem.setMnemonic(KeyEvent.VK_C);
    fileMenu.add(caseMenuItem);
```

Example 4 - Add CheckBox Menu Item into frame using Swing Component

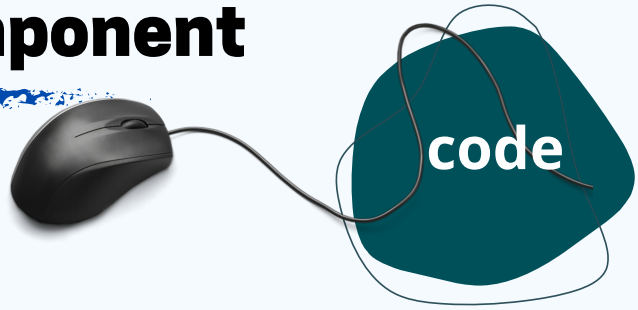
code

```
 ActionListener aListener = new ActionListener() {
      public void actionPerformed(ActionEvent event) {
      AbstractButton aButton = (AbstractButton)
event.getSource();
      boolean selected =aButton.getModel().isSelected();
      String newLabel;
      Icon newIcon;
      if (selected) {
                    newLabel = "Value-1";
      } else {
                    newLabel = "Value-2";
                }
        aButton.setText(newLabel);
          }
        };
        caseMenuItem.addActionListener(aListener);
        frame.setJMenuBar(menuBar);
        frame.setSize(350, 250);
        frame.setVisible(true);
    }
}
```

output

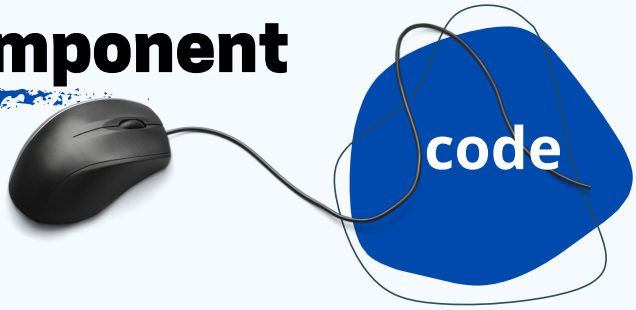


Example 5 - Create a simple page using Swing Component



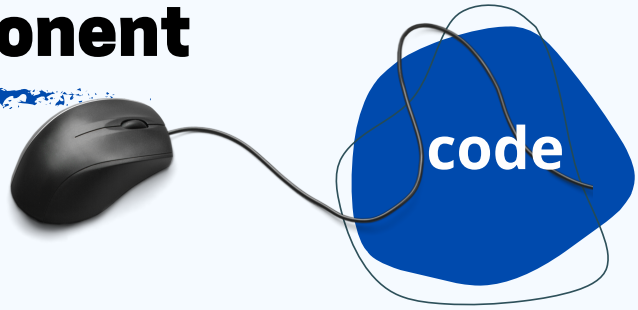

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class SwingDemo2 extends JFrame implements
ActionListener
{
   JRadioButton eng, doc;
   ButtonGroup bg;
   JTextField jtf;
   JCheckBox bcd, ccb, acb;
   JTextArea jta;
   SwingDemo2 ()
   {
        eng = new JRadioButton ("Engineer");
        doc = new JRadioButton ("Doctor");
        bg = new ButtonGroup ();
        bg.add (eng);  bg.add (doc);
        jtf = new JTextField (20);
        bcd = new JCheckBox ("Bike");
        ccb = new JCheckBox ("Car");
        acb = new JCheckBox ("Aeroplane");
        jta = new JTextArea (3, 20);
        Container c = this.getContentPane ();
        c.setLayout (new FlowLayout ());

 // Registering the listeners with the components
        eng.addActionListener (this);
        doc.addActionListener (this);
        bcd.addActionListener (this);
        ccb.addActionListener (this);
        acb.addActionListener (this);
```

Example 5 - Create a simple page using Swing Component

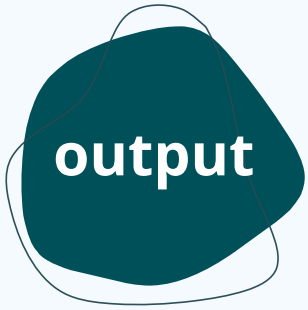


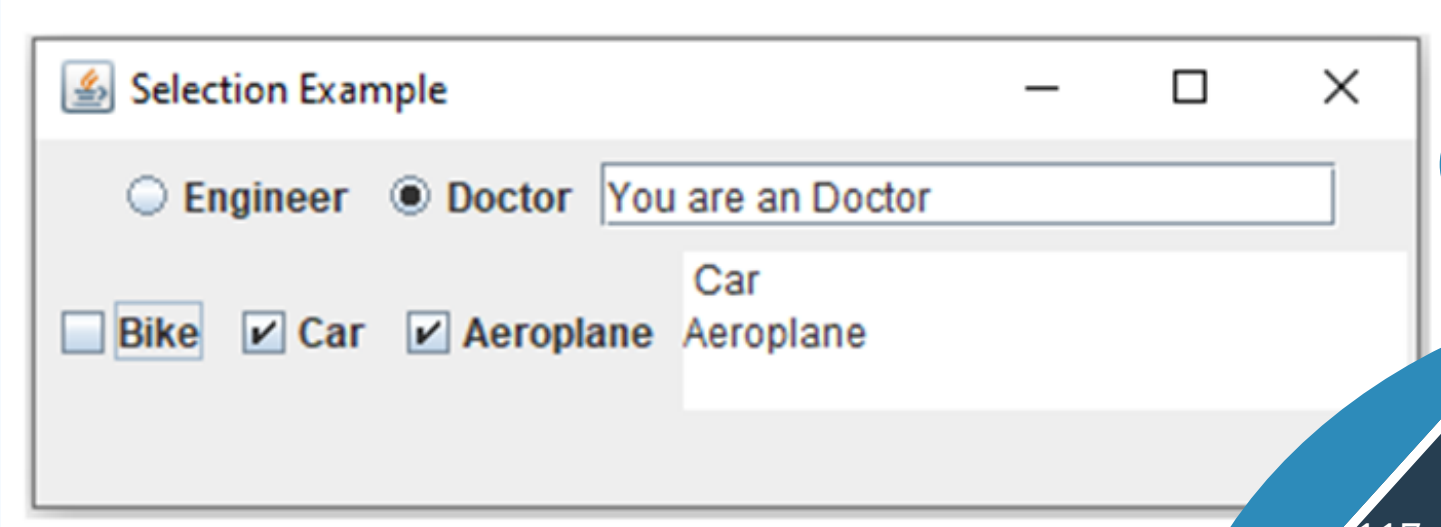

```
    c.add (eng);
    c.add (doc);
    c.add (jtf);
    c.add (bcd);
    c.add (ccb);
    c.add (acb);
    c.add (jta);
    this.setVisible (true);
    this.setSize (500, 500);
    this.setTitle ("Selection Example");
    this.setDefaultCloseOperation
    (JFrame.EXIT_ON_CLOSE);
}

public void actionPerformed (ActionEvent ae)
{
    if (ae.getSource () == eng)
    {
     jtf.setText ("You are an Engineer");
    }
    if (ae.getSource () == doc)
    {
     jtf.setText ("You are an Doctor");
    }
    String str = " ";
    if (bcd.isSelected ())
    {
     str += "Bike\n";
    }
```

Example 5- Create a simple page using Swing Component

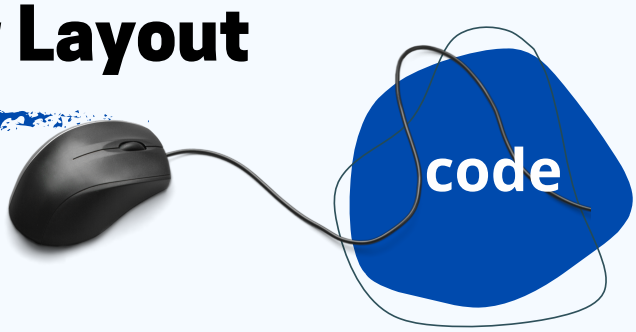



```
if (ccb.isSelected ())
      {
       str += "Car\n";
      }
      if (acb.isSelected ())
      {
       str += "Aeroplane\n";
      }
      jta.setText (str);
   }
   public static void main (String[]args)
   {
       new SwingDemo2 ();
   }
}
```





Example 6 - Flow Layout

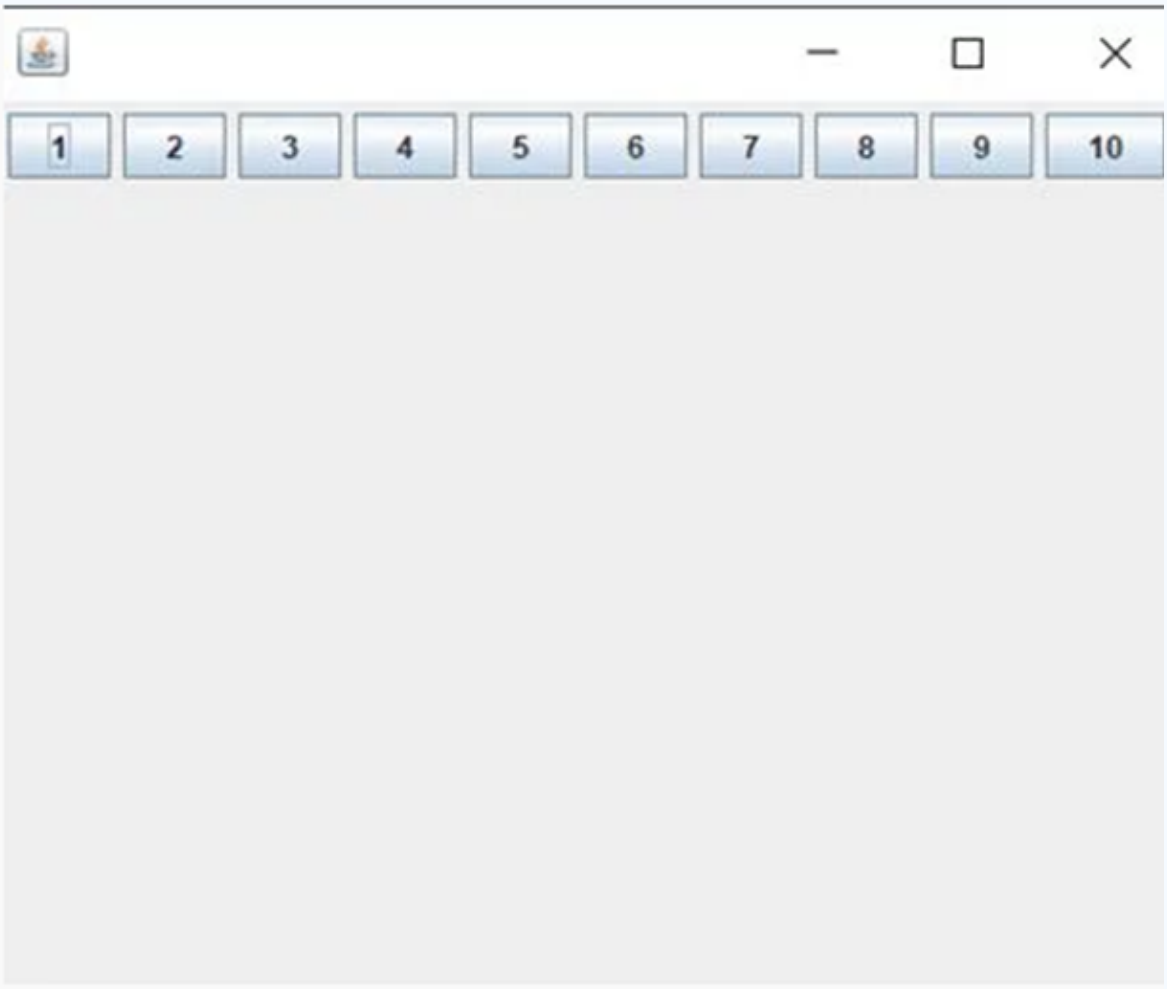



```
import java.awt.*;
import javax.swing.*;

public class FlowDemo1{
JFrame frame1;
FlowDemo1(){
    frame1=new JFrame();

JButton box1=new JButton("1");
JButton box2=new JButton("2");
JButton box3=new JButton("3");
JButton box4=new JButton("4");
JButton box5=new JButton("5");
JButton box6=new JButton("6");
JButton box7=new JButton("7");
JButton box8=new JButton("8");
JButton box9=new JButton("9");
JButton box10=new JButton("10");

    frame1.add(box1);
    frame1.add(box2);
    frame1.add(box3);
    frame1.add(box4);
    frame1.add(box5);
    frame1.add(box6);
    frame1.add(box7);
    frame1.add(box8);
    frame1.add(box9);
    frame1.add(box10);
```

Example 6 - Flow Layout (Cont..)

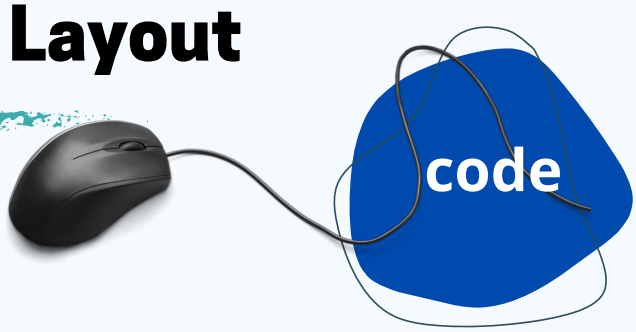
code

```
frame1.setLayout(new FlowLayout(FlowLayout.LEFT));

frame1.setSize(400,400);
frame1.setVisible(true);
}
public static void main(String[] args) {
   new FlowDemo1();
}
}
```



output

Example 7 - Grid Layout

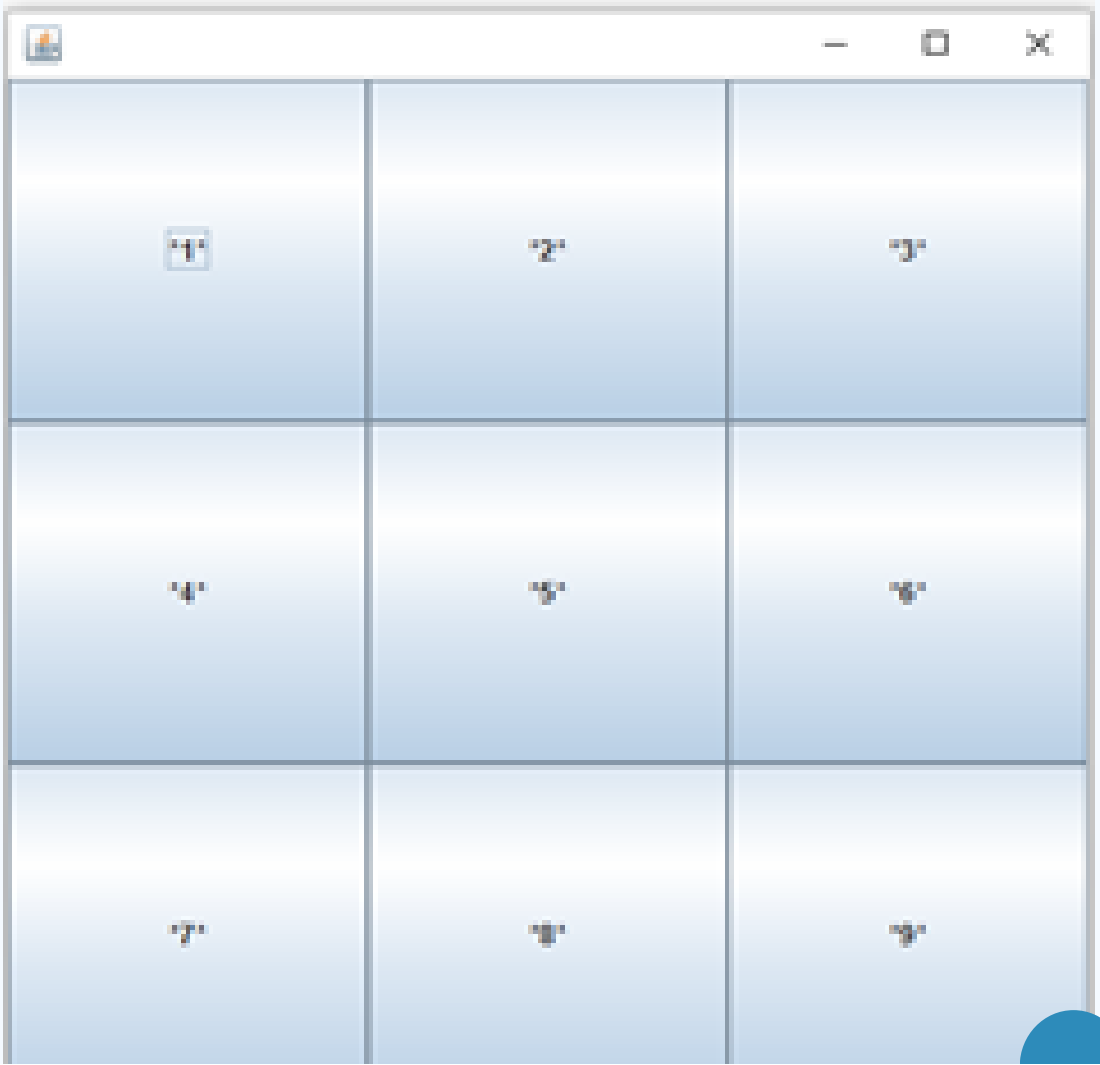



```
import java.awt.*;
import javax.swing.*;

public class GridDemo1{
JFrame frame1;
GridDemo1(){
    frame1=new JFrame();

JButton box1=new JButton("*1*");
JButton box2=new JButton("*2*");
JButton box3=new JButton("*3*");
JButton box4=new JButton("*4*");
JButton box5=new JButton("*5*");
JButton box6=new JButton("*6*");
JButton box7=new JButton("*7*");
JButton box8=new JButton("*8*");
JButton box9=new JButton("*9*");

    frame1.add(box1);
    frame1.add(box2);
    frame1.add(box3);
    frame1.add(box4);
    frame1.add(box5);
    frame1.add(box6);
    frame1.add(box7);
    frame1.add(box8);
    frame1.add(box9);
```

Example 7 - Grid Layout (Cont..)

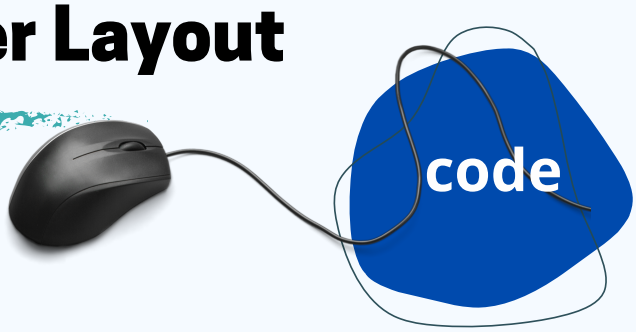
code

```
frame1.setLayout(new GridLayout(3,3));
frame1.setSize(500,500);
frame1.setVisible(true);
}
public static void main(String[] args) {
    new GridDemo1();
}
}
```



output

Example 8 - Border Layout



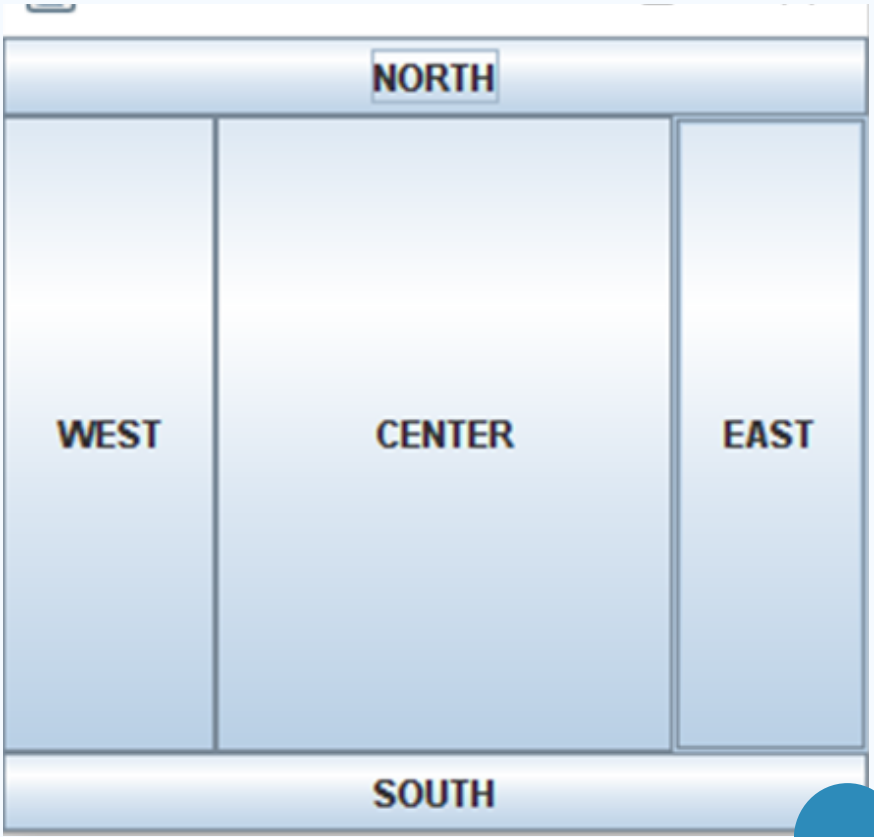

```
import java.awt.*;
 import javax.swing.*;
 public class border
 {
 JFrame JF;
 border()
 {
 JF=new JFrame(); //JFrame object
 //Lying at top, will be the button named 'North'
 JButton b1=new JButton("NORTH");
 //Lying at bottom, will be the button named
'South'
 JButton b2=new JButton("SOUTH");
 //Lying at left, will be the button named 'East'
 JButton b3=new JButton("EAST");
 //Lying at right, will be the button named
'West'
 JButton b4=new JButton("WEST");
 //In the center, will be the button named
'Center'
 JButton b5=new JButton("CENTER");
 //Adding our buttons
```

Example 8 - Border Layout(Cont..)

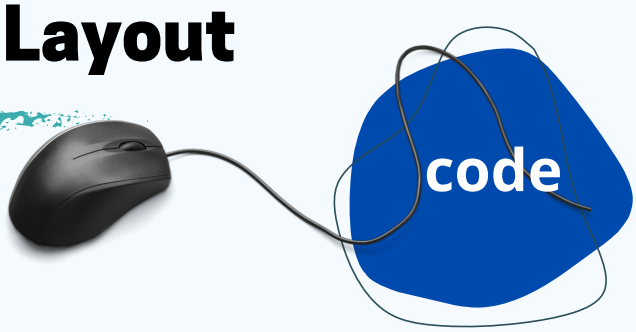
code

```
//Adding our buttons
JF.add(b1,BorderLayout.NORTH);
JF.add(b2,BorderLayout.SOUTH);
JF.add(b3,BorderLayout.EAST);
JF.add(b4,BorderLayout.WEST);
JF.add(b5,BorderLayout.CENTER);
//Function to set size of the Frame
JF.setSize(300, 300);
//Function to set visible status of the frame
JF.setVisible(true);
}
//Driver Code
public static void main(String[] args)
{

new border();
}
}
```

output

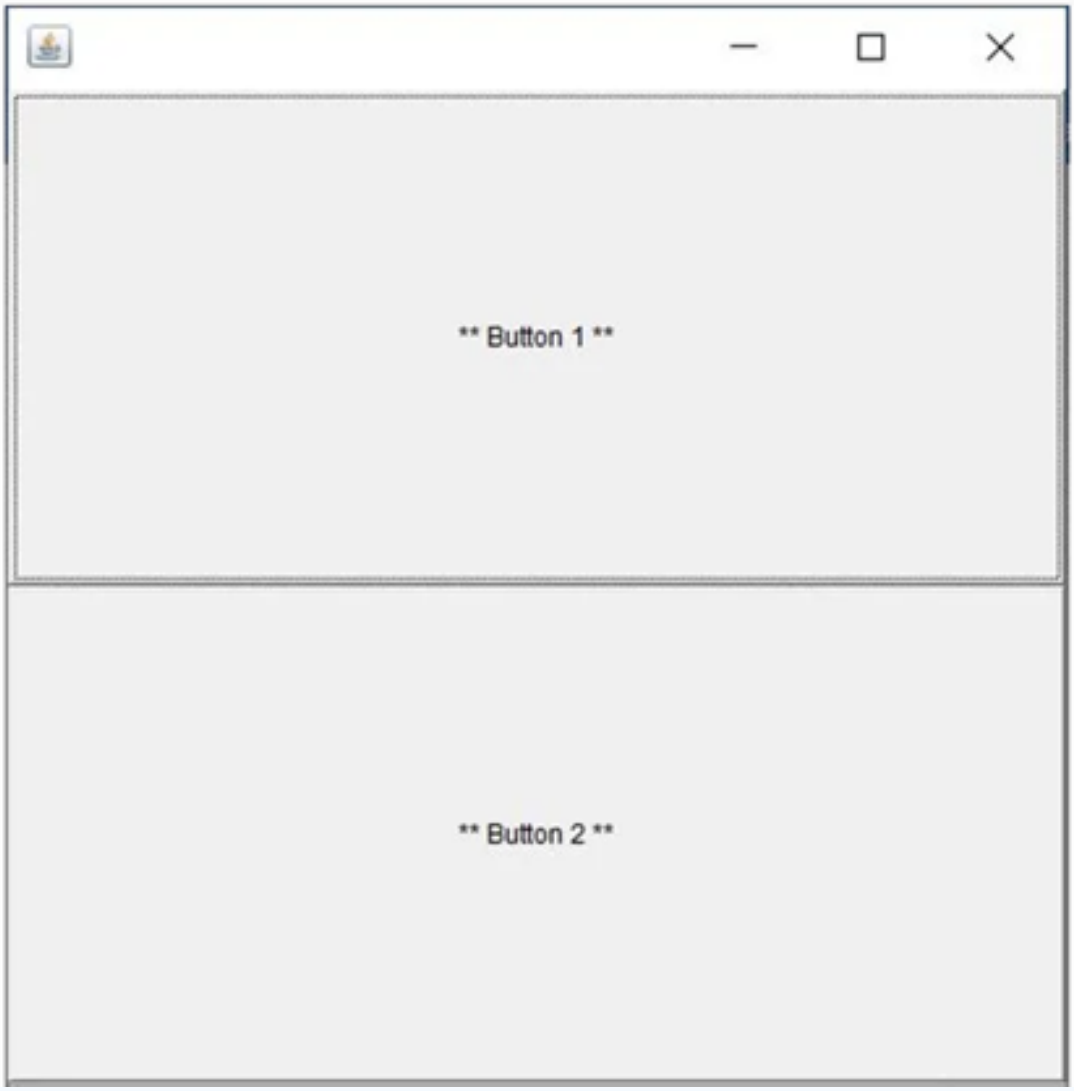


Example 9 - Box Layout




```
import java.awt.*;
import javax.swing.*;
public class BoxDemo1 extends Frame {
   Button buttonBox[];
   public BoxDemo1 ()
   {
      buttonBox = new Button [2];
      for (inti = 0;i<2;i++)
        {

buttonBox[i] = new Button ("** Button " + (i +
1)+" **");

add (buttonBox[i]);
        }
setLayout (new BoxLayout (this,
BoxLayout.Y_AXIS));
    setSize(500,500);
    setVisible(true);
    }

public static void main(String args[])
{
                BoxDemo1 obj=new BoxDemo1();
}
}
```

Example 9 - Box Layout(Cont..)



output

Lab Exercise

Chapter 3

PN

Example 1- Action Listener

code

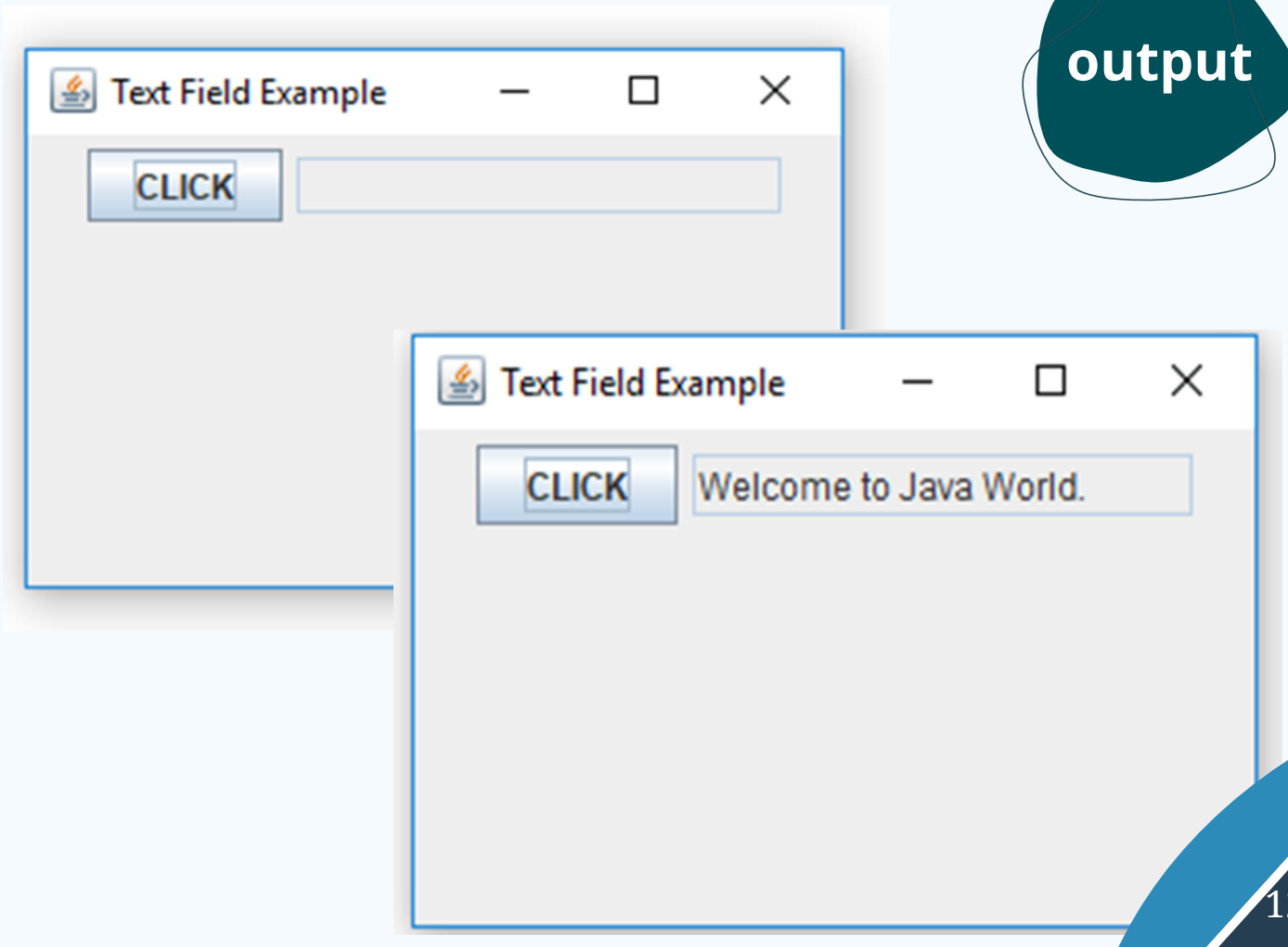
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class ButtonAction extends JFrame
implements ActionListener
{
 JTextField entry = new JTextField (15);
 JLabel label = new JLabel ("Insert
Data");
 JButton button1 = new JButton( "CLICK");

 ButtonAction()
 {
 setTitle("Text Field Example");
 setLayout(new FlowLayout());
 add(button1);
 add (entry);
 entry.setEditable(false);
 button1.addActionListener(this);
 setSize(300,200);
 setVisible(true);
 }
```

Example 1- Action Listener.. Cont

code

```
public void actionPerformed(ActionEvent e)
{
  entry.setText("Welcome to Java World.");
}

 public static void main (String[] arg)
 {
 ButtonAction ba=new ButtonAction();
 }
```

output



Example 2- Action Listener

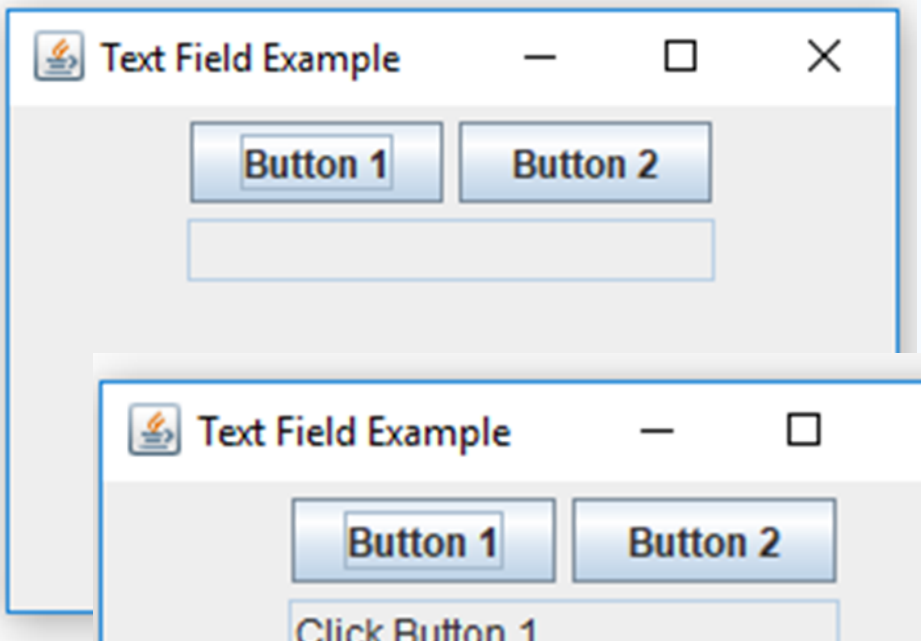
code

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

class ButtonAction2 extends JFrame
implements ActionListener
{
 JTextField entry = new JTextField (15);
 JButton button1 = new JButton( "Button 1");
 JButton button2 = new JButton( "Button 2");

 ButtonAction2()
 {
 setTitle("Text Field Example");
 setLayout(new FlowLayout());
 add(button1);
 add(button2);
 add (entry);
 entry.setEditable(false);
  button1.addActionListener(this);
 button2.addActionListener(this);
 setSize(300,200);
 setVisible(true);
 }
```

Example 2- Action Listener...Cont

code

```
public void actionPerformed(ActionEvent e){
if (e.getSource()==button1)
            entry.setText("Click Button 1");
if (e.getSource()==button2)
            entry.setText("Click Button 2");
          }

public static void main (String[] arg)
{
ButtonAction2 ba2=new ButtonAction2();
}
}
```



output

Example 3- Action Listener, Window Listener

code

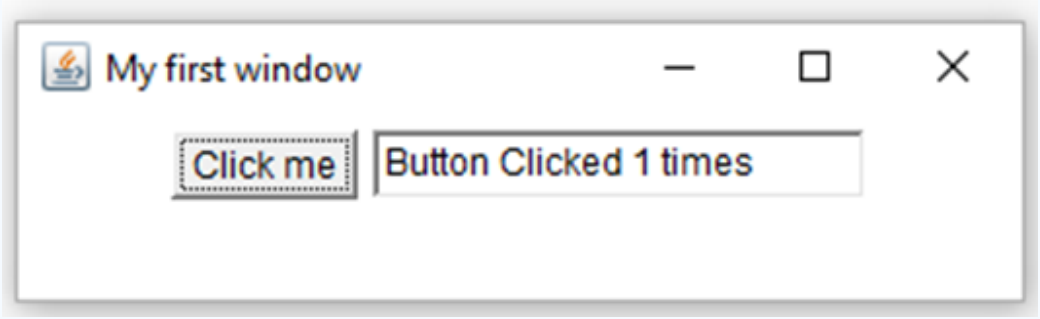
```
 import java.awt.*;
import java.awt.event.*;

public class AL extends Frame implements
WindowListener,ActionListener {
        TextField text = new TextField(20);
        Button b;
        private int numClicks = 0;
public static void main(String[] args) {
     AL myWindow = new AL("My first
window");
     myWindow.setSize(350,100);
     myWindow.setVisible(true);
   }

public AL(String title) {
    super(title);
    setLayout(new FlowLayout());
    addWindowListener(this);
    b = new Button("Click me");
    add(b);
    add(text);
    b.addActionListener(this);
  }
```

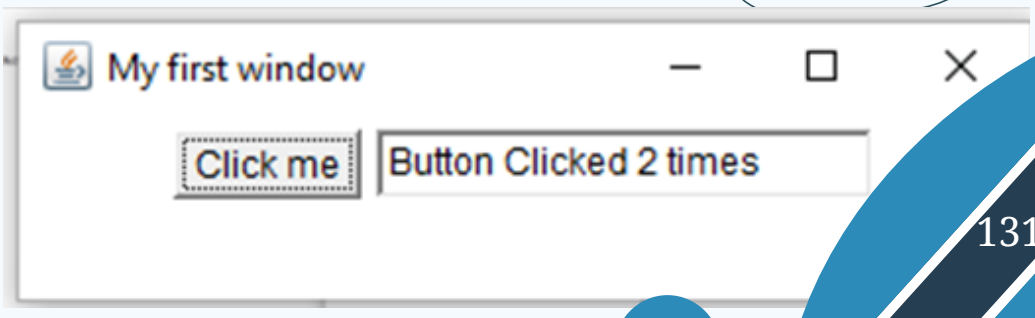
Example 3- Action Listener, Window Listener...Cont

code

```
 public void actionPerformed(ActionEvent e) {
  numClicks++;
  text.setText("Button Clicked" + numClicks + "
times");
 }
 public void windowClosing(WindowEvent e) {
  dispose();
  System.exit(0);
 }
 public void windowOpened(WindowEvent e) {}
 public void windowActivated(WindowEvent e) {}
 public void windowIconified(WindowEvent e) {}
 public void windowDeiconified(WindowEvent e) {}
 public void windowDeactivated(WindowEvent e) {}
 public void windowClosed(WindowEvent e) {}
}
```

output





Example 4 - ActionListener (Simple Calculator)

code

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

class Calculator1 implements ActionListener
{
 //Declaring Objects
 Frame f=new Frame();
 Label l1=new Label("First Number");
 Label l2=new Label("Second Number");
 Label l3=new Label("Result");
 TextField t1=new TextField();
 TextField t2=new TextField();
 TextField t3=new TextField();
 Button b1=new Button("Add");
 Button b2=new Button("Sub");
 Calculator1()
 {
  //Giving Coordinates
  l1.setBounds(50,100,100,20);
  l2.setBounds(50,140,100,20);
  l3.setBounds(50,180,100,20);
  t1.setBounds(200,100,100,20);
  t2.setBounds(200,140,100,20);
  t3.setBounds(200,180,100,20);
  b1.setBounds(50,250,50,20);
  b2.setBounds(110,250,50,20);
```

Example 4 - ActionListener (Simple Calculator)...Cont

code

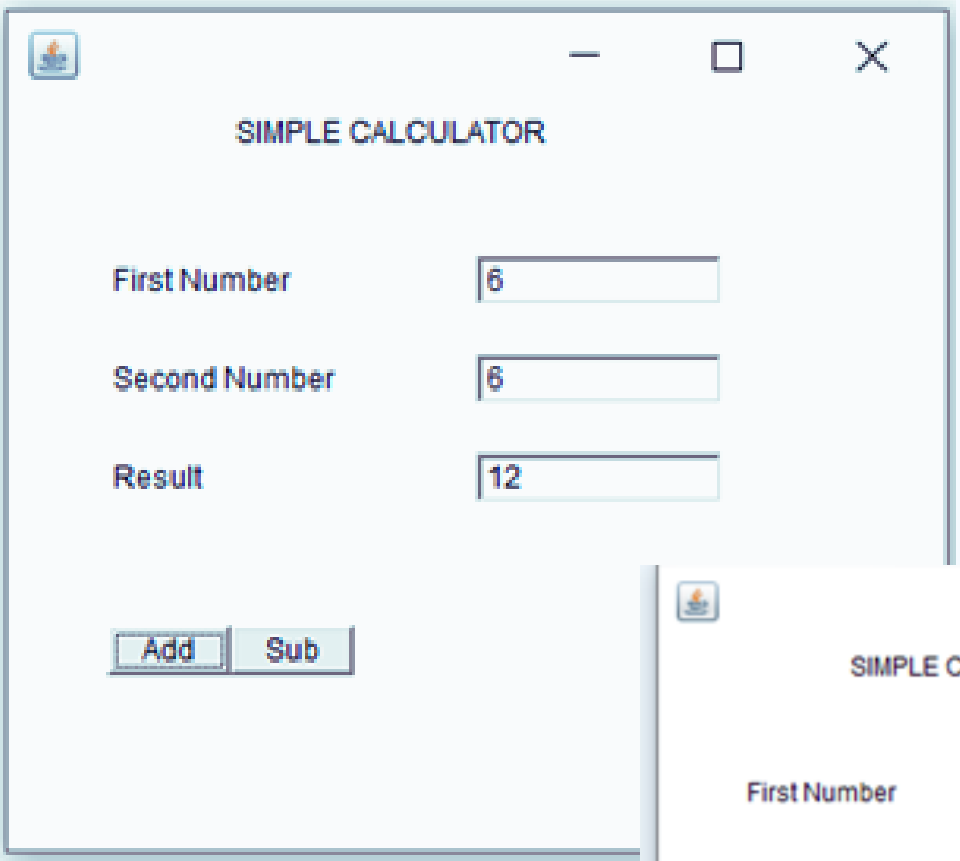
```
  //Adding components to the frame
    f.add(l1);        f.add(l2);
    f.add(l3);        f.add(t1);
    f.add(t2);        f.add(t3);
    f.add(b1);        f.add(b2);
    b1.addActionListener(this);
    b2.addActionListener(this);
    f.setLayout(null);
    f.setVisible(true);
    f.setSize(400,350);
   }

public void actionPerformed(ActionEvent e)
{
 int n1=Integer.parseInt(t1.getText());
 int n2=Integer.parseInt(t2.getText());
 if(e.getSource()==b1)
 {
  t3.setText(String.valueOf(n1+n2));
 }
  if(e.getSource()==b2)
 {
  t3.setText(String.valueOf(n1-n2));
 }
}
```

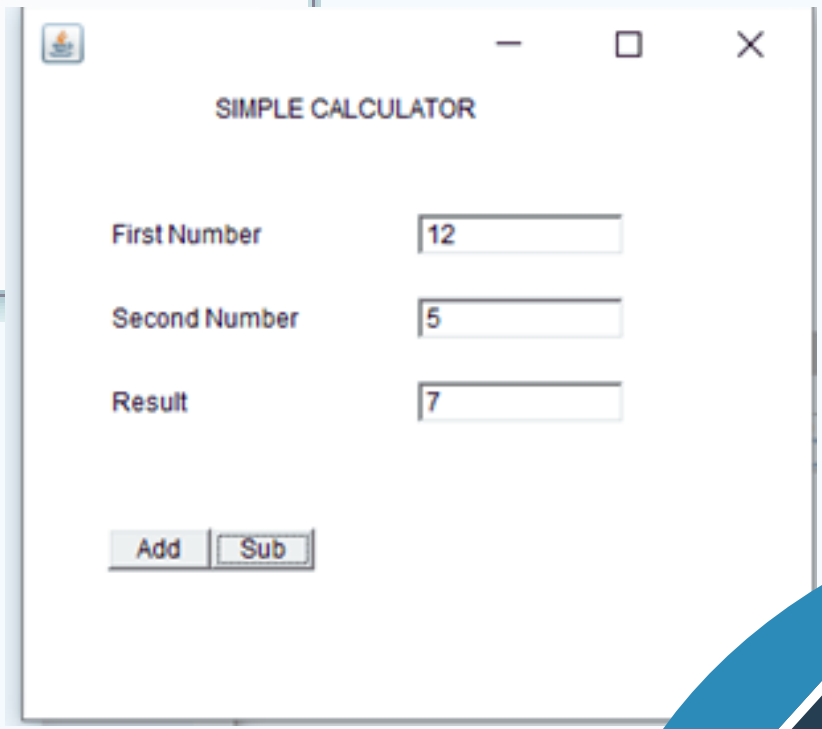
output

Example 4 - ActionListener (Simple Calculator)..Cont

```
public static void main(String...s)
 {
  new Calculator1();
 }
}
```

output





Example 5 - ActionListener (Age Calculation)

code

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

class AgeCalculation implements ActionListener
{
 //declaring objects
 Frame f = new Frame("AGE CALCULATION");

 Label l1 = new Label("Birth Year");
 Label l2 = new Label("Current Year");
 Label l3 = new Label("Age");
 TextField t1= new TextField();
 TextField t2 = new TextField();
 TextField t3 = new TextField();
 Button b1=new Button ("Calculate");

 AgeCalculation()
 {

 l1.setBounds(160,70,100,20);
 l2.setBounds(150,130,100,20);
 l3.setBounds(170,230,100,20);
 t1.setBounds(110,100,160,20);
 t2.setBounds(110,150,160,20);
 t3.setBounds(110,250,160,20);
 b1.setBounds(170,190,50,20);
```

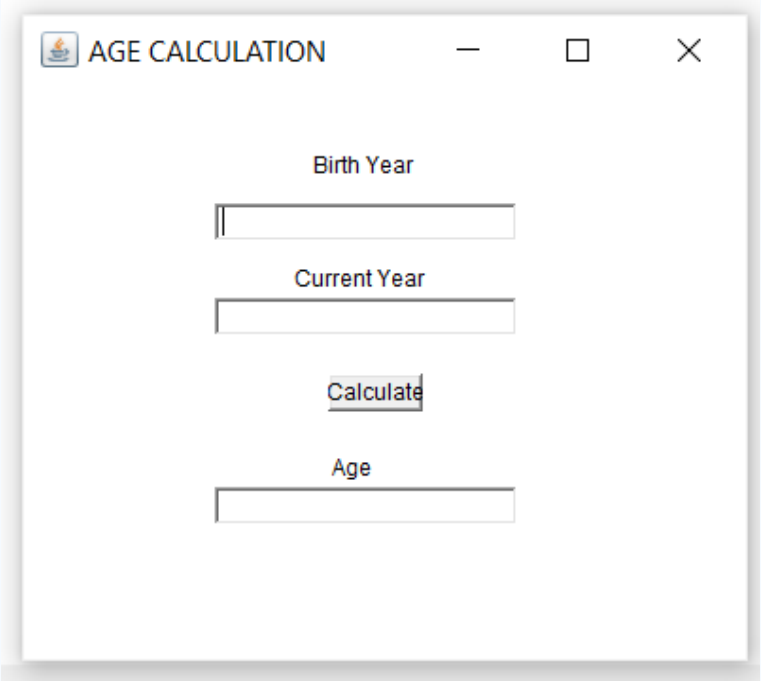
Example 5 - ActionListener (Age Calculation).. Cont

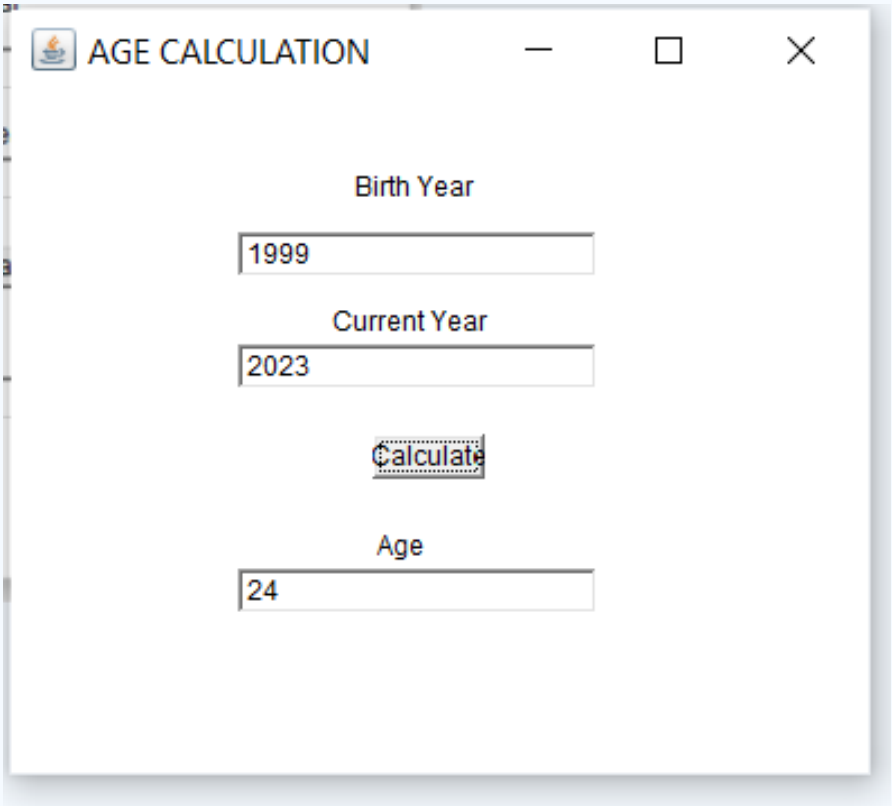
code

```
 f.add(l1);  f.add(l2);  f.add(l3);
 f.add(t1);  f.add(t2);  f.add(t3);
 f.add(b1);
 b1.addActionListener(this);
 f.setLayout(null);
 f.setVisible(true);
 f.setSize(400,350);
 }
 public void actionPerformed(ActionEvent e)
 {
 int tahunsemasa
=Integer.parseInt(t2.getText());
 int tahunlahir=Integer.parseInt(t1.getText());
 if(e.getSource()==b1)
 {
  t3.setText(String.valueOf(tahunsemasa-
tahunlahir));
 }
}

public static void main(String[] args)
{
 new AgeCalculation();
}
}
```

Example 5 - ActionListener (Age Calculation).. Cont

output





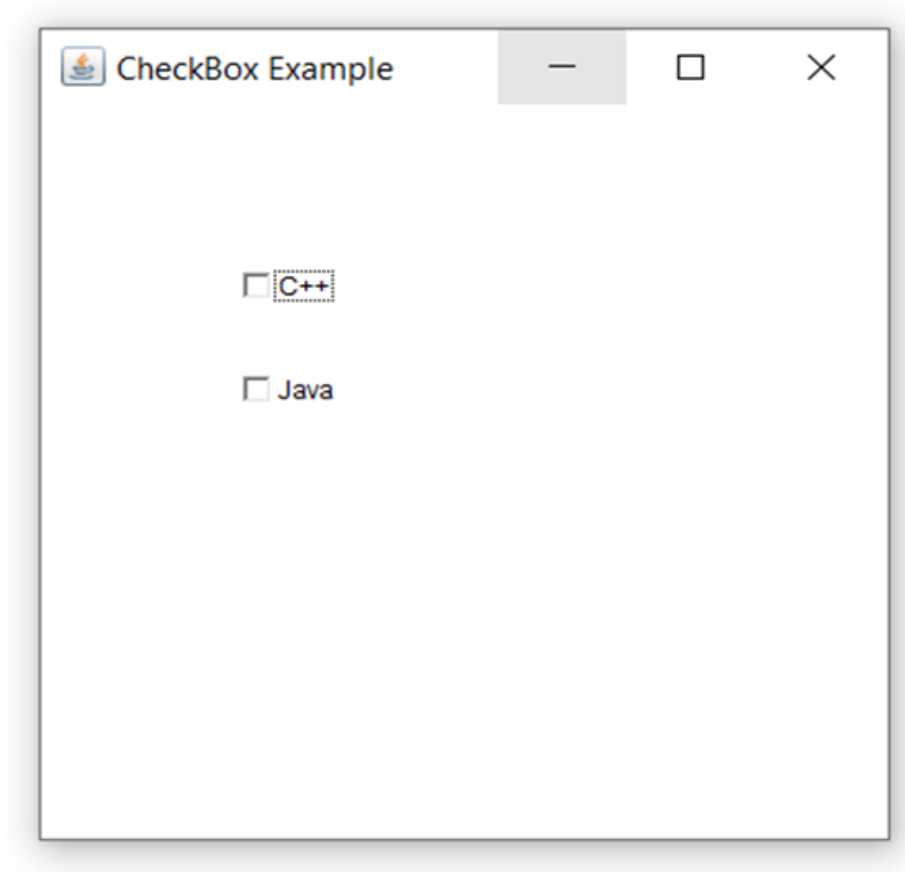
Example 6 - ItemListener

code

```
public class ItemListenerExample implements Ite
mListener{
    Checkbox checkBox1,checkBox2;
    Label label;
    ItemListenerExample(){
        Frame f= new Frame("CheckBox Example");
        label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        checkBox1 = new Checkbox("C++");
        checkBox1.setBounds(100,100, 50,50);
        checkBox2 = new Checkbox("Java");
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.add(label);
        checkBox1.addItemListener(this);
        checkBox2.addItemListener(this);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
```

Example 6 - ItemListener

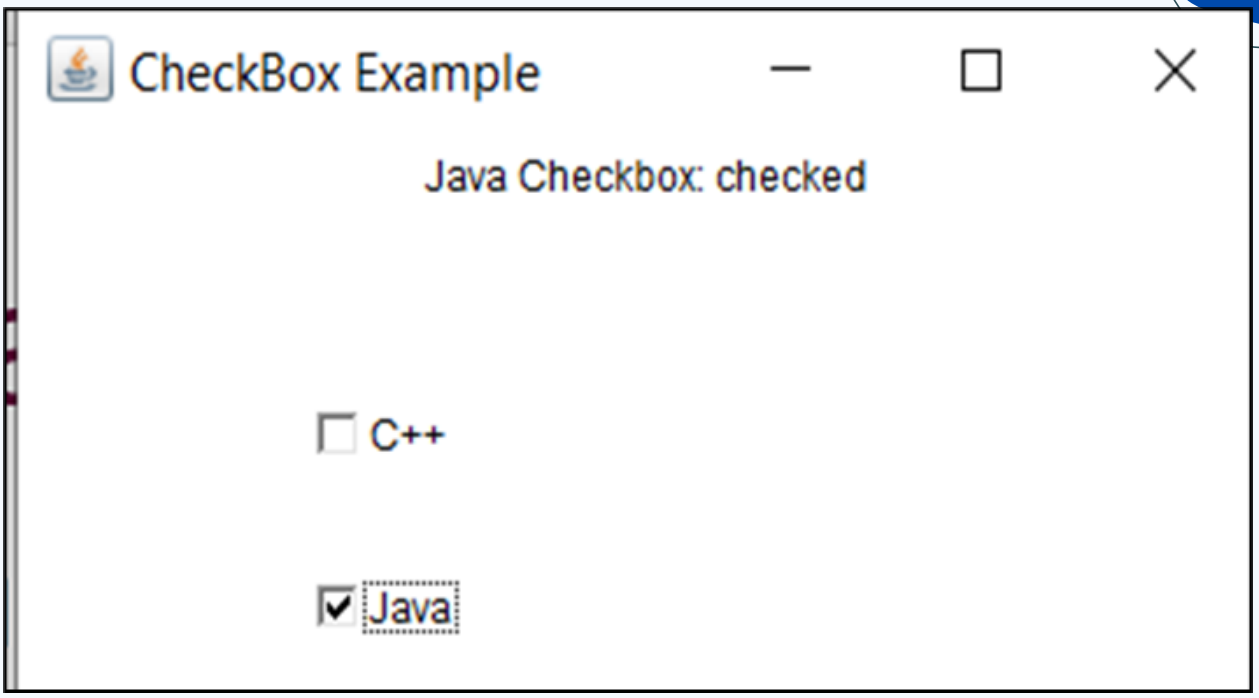
Output & Code

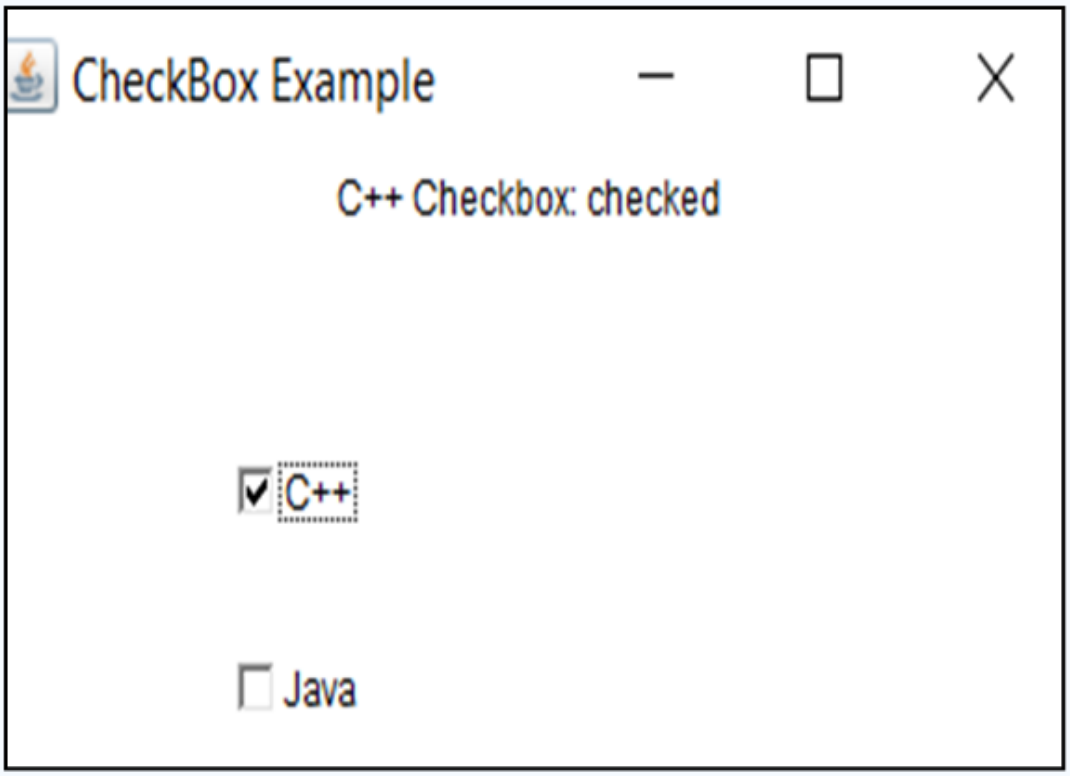



```
  public void itemStateChanged(ItemEvent e)
{
        if(e.getSource()==checkBox1)
            label.setText("C++ Checkbox: "

            + (e.getStateChange()==1?"checke
d":"unchecked"));
        if(e.getSource()==checkBox2)
        label.setText("Java Checkbox: "
        + (e.getStateChange()==1?"checked":"
unchecked"));
     }
```

Example 6 - ItemListener

Output





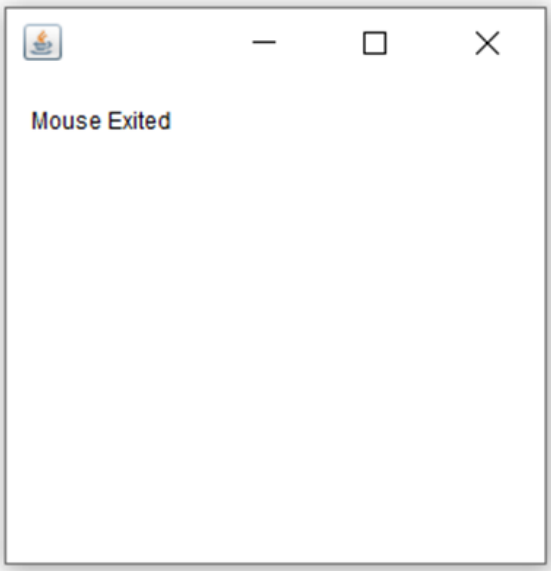
Example 7 - MouseListener

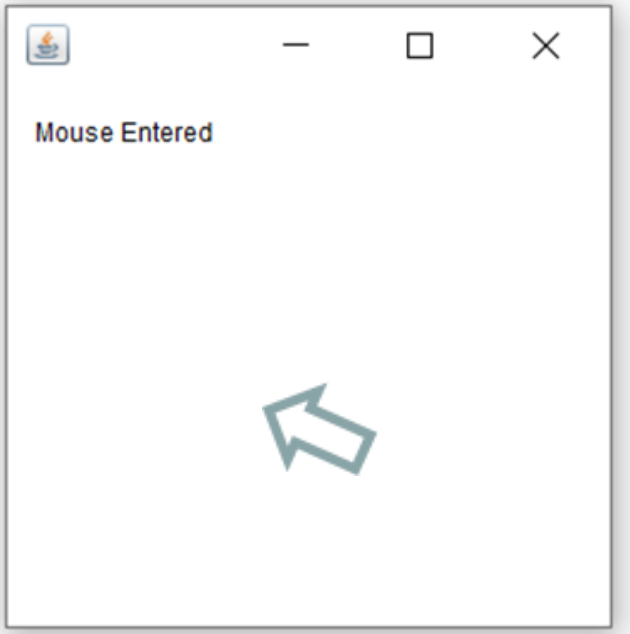
code

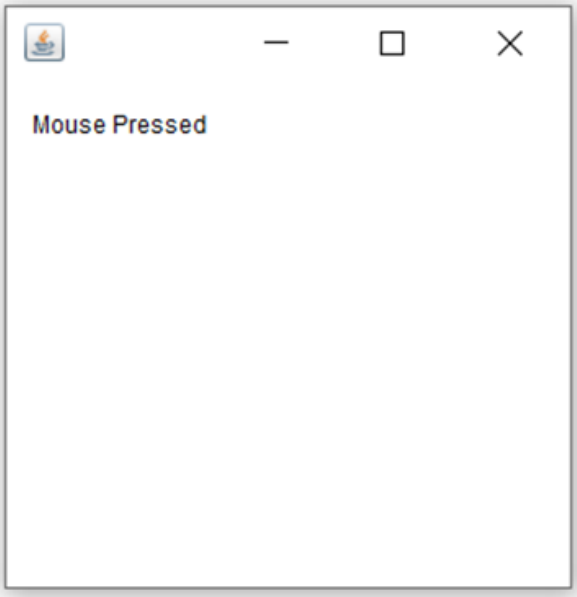
```
public class MouseListenerExample
extends Frame implements MouseListener{
    Label l;
    MouseListenerExample(){
        addMouseListener(this);

        l=new Label();
        l.setBounds(20,50,100,20);
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
}
public void mouseClicked(MouseEvent e)
{l.setText("Mouse Clicked");}
public void mouseEntered(MouseEvent e)
{l.setText("Mouse Entered");}
public void mouseExited(MouseEvent e)
{l.setText("Mouse Exited"); }
public void mousePressed(MouseEvent e)
{l.setText("Mouse Pressed");}
public void mouseReleased(MouseEvent e)
{l.setText("Mouse Released");}
public static void main(String[] args)
{ new MouseListenerExample();}
```

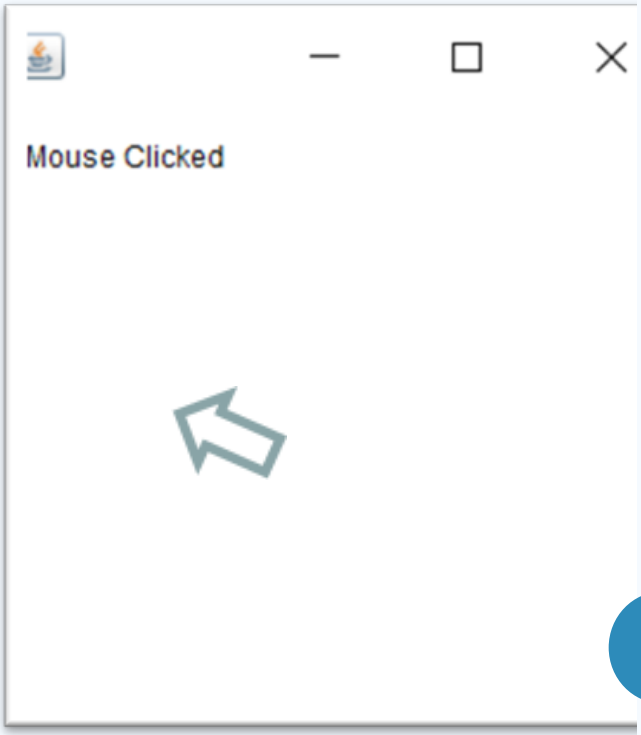
Example 7 - MouseListener

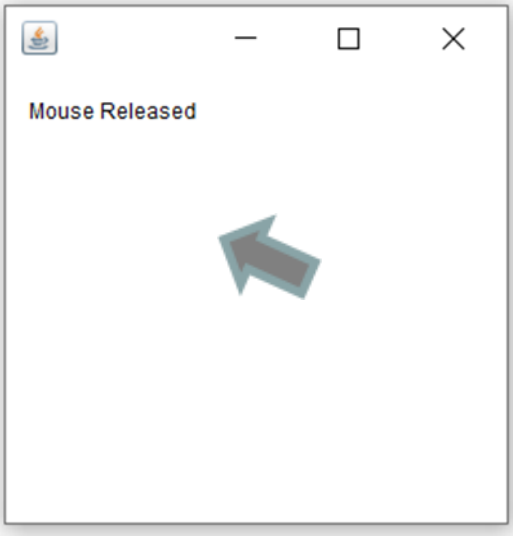
output











Example 7 - MouseListener

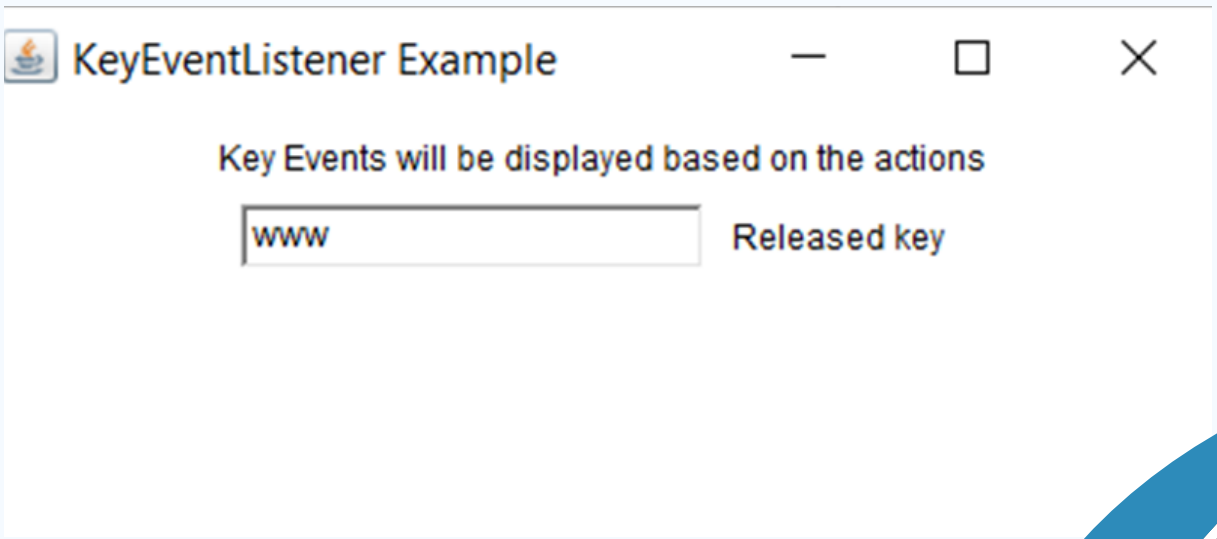
code

```
public class JavaKeyListenerExample implements
KeyListener
{
Label lb1, lbl2, lb;
TextField tf1;
Frame fr;
String s;
JavaKeyListenerExample()
{
fr = new Frame("KeyEventListener Example");
lb1= new Label(" Key Events will be displayed
based on the actions", Label.CENTER);

lbl2= new Label();
lb= new Label();
tf1 = new TextField(20);
fr.setLayout(new FlowLayout());
fr.add(lb1);
fr.add(tf1);
fr.add(lbl2);
//on clicking the button, text field data will
be read
tf1.addKeyListener(this);
fr.setSize(460,250);
fr.setVisible(true);
}
```

Example 8 - KeyListener

code

```
//events to be done on pressing key
public void keyPressed(KeyEvent ev)
{
lbl2.setText("Pressed Key");
}
//events to be done on releasing key
public void keyReleased(KeyEvent ev)
{
lbl2.setText(" Released key");
}
//events to be done on typing key
public void keyTyped(KeyEvent ev)
{
lbl2.setText("Key is typed");
fr.setVisible(true);
}
```

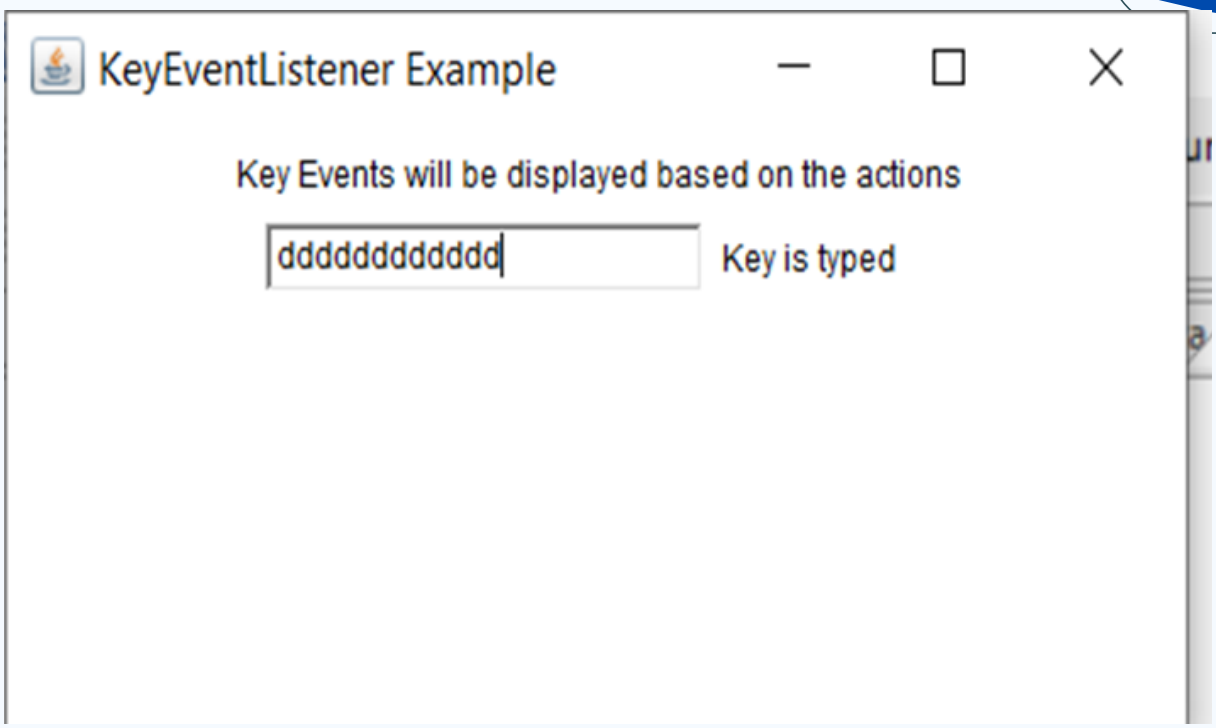


events to be done on releasing key

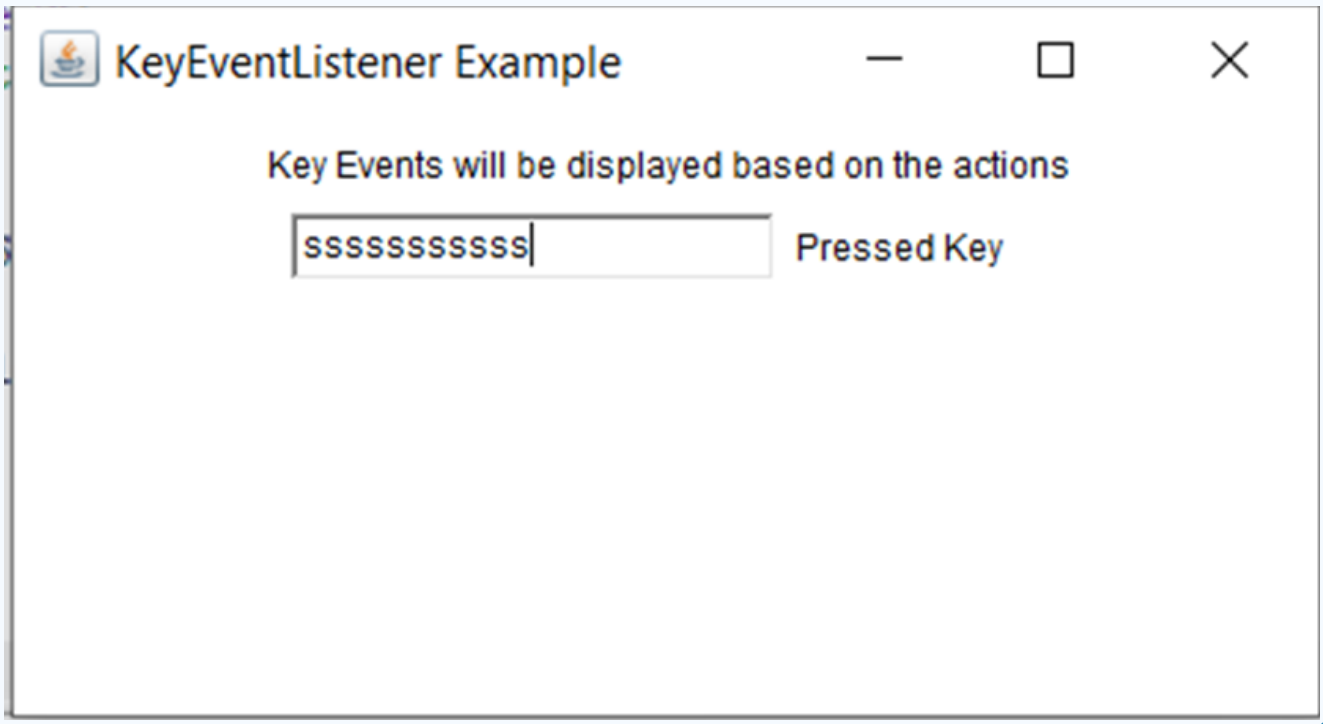
Example 8 - KeyListener

Output

Events to be done on typing key



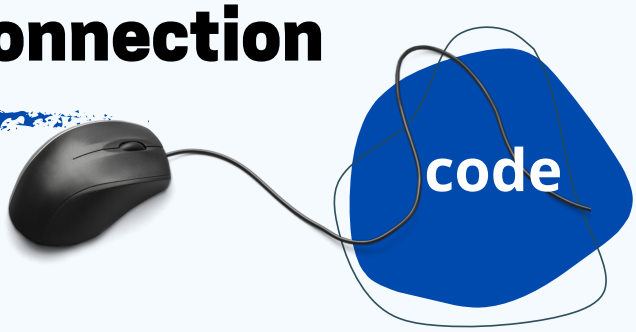
Events to be done on pressing key



Lab Exercise

Chapter 4

Example 1 - Test Connection




```
TES :
*  XAMMP
**  Create database :
     -- Database name : akses
     -- Table name : login
     -- Add 2 field : user, pass
     -- Insert 2 record for user and pass
```

user	pass
admin	1234
user	4321

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

public class TestLoad
{
            Connection con;
            Statement st;
            ResultSet rs;

            public TestLoad()
            {
                        try
                        {

Class.forName("com.mysql.jdbc.Driver");

con=DriverManager.getConnection("jdbc:mysql://lo
calhost:3306/akses","root","");

st=con.createStatement();

rs=st.executeQuery("select * from login");
```

Example 1 - Test Connection..Cont

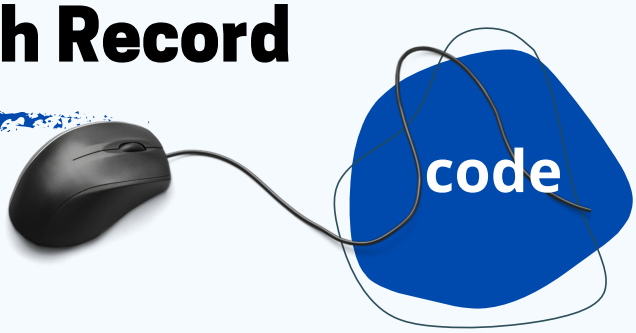
code

```
while(rs.next())
      {
System.out.println(rs.getString(1)+"
"+rs.getString(2));
        }

         }
           catch(Exception ei)
            {

System.out.println("Connection Failed");
            }
         }

     public static void main(String args[])
     {
            new TestLoad();
      }
}
```

output

```
General Output
--------------------Configuration: <Default>--------------------
admin 1234
user 4321

Process completed.
```

Example 2 - Search Record

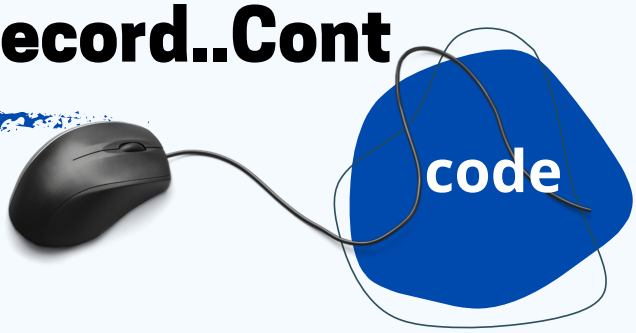



```
TES :
*   XAMMP
**  Create database :
     -- Database name : akses
     -- Table name : emp
     -- Add 4 field : u_name,u_mail,u_pass,ucountry
     -- Insert 2 record
```

u_name	u_mail	u_pass	u_country
ali	aali@gmail.com	ali123	dungun
abu	abu@psmza.edu.my	abu123	kemaman

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.sql.*;
public class Search_record extends JFrame
implements ActionListener {
//Initializing Components
   JLabel lb, lb1, lb2, lb3, lb4, lb5;
   JTextField tf1, tf2, tf3, tf4, tf5;
   JButton btn;
   //Creating Constructor for initializing JFrame
components
Search_record() {
//Providing Title
super("Fetching Employee Information");
  lb5 = new JLabel("Enter Name:");
  lb5.setBounds(20, 20, 100, 20);
  tf5 = new JTextField(20);
  tf5.setBounds(130, 20, 200, 20);
  btn = new JButton("Submit");
  btn.setBounds(50, 50, 100, 20);
  btn.addActionListener(this);
```

Example 2 - Search Record..Cont




```
lb = new JLabel("Fetching Employee Information
From Database");
lb.setBounds(30, 80, 450, 30);
lb.setForeground(Color.red);
lb.setFont(new Font("Serif", Font.BOLD, 20));
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(500, 500);
        lb1 = new JLabel("U_Name:");
        lb1.setBounds(20, 120, 100, 20);
        tf1 = new JTextField(50);
        tf1.setBounds(130, 120, 200, 20);
        lb2 = new JLabel("U_Mail:");
        lb2.setBounds(20, 150, 100, 20);
        tf2 = new JTextField(100);
        tf2.setBounds(130, 150, 200, 20);
        lb3 = new JLabel("U_Pass:");
        lb3.setBounds(20, 180, 100, 20);
        tf3 = new JTextField(50);
        tf3.setBounds(130, 180, 200, 20);
        lb4 = new JLabel("U_Country:");
        lb4.setBounds(20, 210, 100, 20);
        tf4 = new JTextField(50);
        tf4.setBounds(130, 210, 100, 20);
        setLayout(null);
        //Add components to the JFrame
        add(lb5);add(tf5);add(btn);

add(lb);add(lb1);add(tf1);add(lb2);add(tf2);
    add(lb3);add(tf3);add(lb4);add(tf4);
```

Example 2 - Search Record..Cont

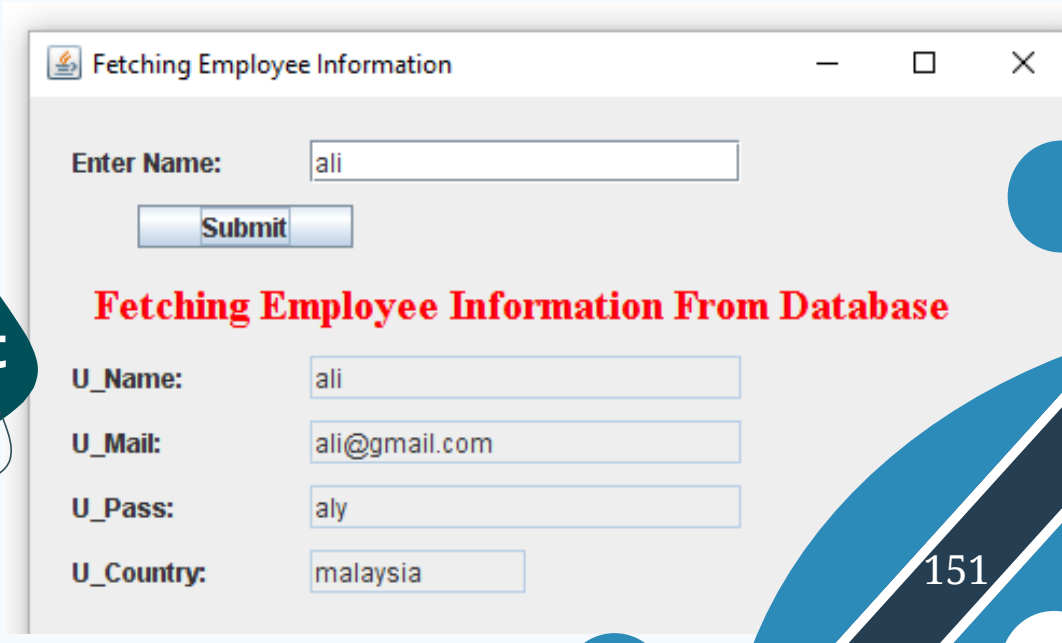

```
//Set TextField Editable False
tf1.setEditable(false);
tf2.setEditable(false);
tf3.setEditable(false);
tf4.setEditable(false);
   }
   public void actionPerformed(ActionEvent e) {
   //Create DataBase Coonection and Fetching
Records
   try {
     String str = tf5.getText();
     Class.forName("com.mysql.jdbc.Driver");
     Connection con =
DriverManager.getConnection("jdbc:mysql://local
host:3306/akses","root","");
     PreparedStatement st =
con.prepareStatement("select * from emp where
u_name=?");
st.setString(1, str);
//Excuting Query
ResultSet rs = st.executeQuery();
if (rs.next()) {
 String s = rs.getString(1);
 String s1 = rs.getString(2);
 String s2 = rs.getString(3);
 String s3 = rs.getString(4);
```

Example 2 - Search Record..Cont

code

```
    //Sets Records in TextFields.
    tf1.setText(s);
    tf2.setText(s1);
    tf3.setText(s2);
    tf4.setText(s3);
    }
    else
    {
JOptionPane.showMessageDialog(null, "Name not
Found");
            }
//Create Exception Handler
       } catch (Exception ex) {
        System.out.println(ex);
       }
   }
   //Running Constructor
   public static void main(String args[]) {
       new Search_record ();
   }
}
```

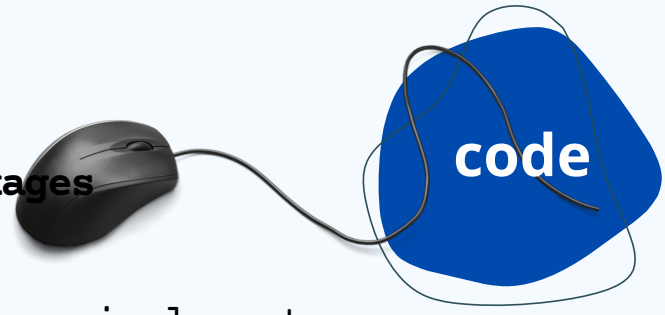
output



Example 3 - Insert Record

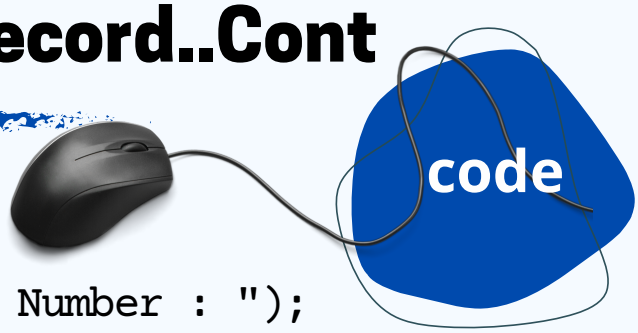
```
TES :
*   XAMMP
**  Create database :
    -- Database name : akses
    -- Table name : baru
    -- Add 5 field: name,matric,program,nohp
    -- Insert 1 record
```

	name	matric	program	nohp
Delete	Ahmad	10422	DDT	018754821



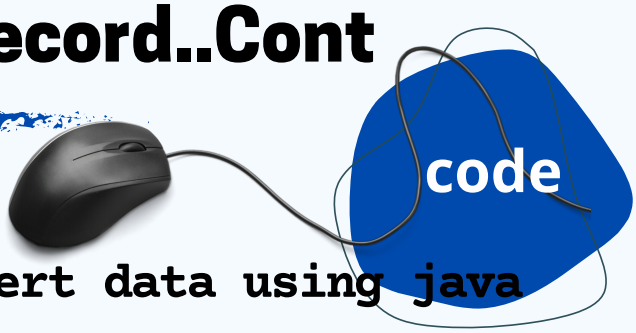

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.FlowLayout;
// STEP 1.1 : import Required packages
import java.sql.*;

public class formInsert extends Frame implements
ActionListener
{
 JLabel tajuk, tajuk2,Lnama, Lnomatrik, Lprogram,
 Lphone, jabatan,phone;
 JTextField Tname, Tmatric, Tprogram, Tphone;
 JButton Bsave, Breset;
//STEP 1.2 :Declare variable for database connection
 Connection conn;
 Statement st;
 ResultSet rs;
 String db;

  public formInsert()
  {
  setLayout(new FlowLayout( ));
  tajuk = new JLabel (" Hi..Welcome to My Class " );
  Bsave=new JButton("SAVE");
```

Example 3 - Insert Record..Cont




```
Breset=new JButton("RESET");
Lnama = new JLabel("Name : ");
Tname = new JTextField (15);
Lnomatrik = new JLabel("Register Number : ");
Tmatric = new JTextField (15);
Lprogram = new JLabel("Program : ");
Tprogram = new JTextField (15);
Lphone = new JLabel("Phone No : ");
Tphone = new JTextField (15);
add(tajuk);add(Lnama);add(Tname);add(Lnomatrik);
add(Tmatric);add(Lprogram);add(Tprogram);add(Lphone);
add(Tphone);add(Bsave);add(Breset);
Bsave.addActionListener(this);
Breset.addActionListener(this);
setSize(200,500);
setVisible(true);
}

 public static void main(String[] args)
 {
 formInsert f = new formInsert();
 f.addWindowListener(new WindowEventHandler());
 }
 public void actionPerformed(ActionEvent e)
 {
     Object pilihan=e.getSource();
      if (pilihan==Bsave)
        {
         try{
 //STEP 2 : Register JDBC  drver using mysql
  Class.forName("com.mysql.jdbc.Driver");

 //STEP 3 : Open Connection

conn=DriverManager.getConnection("jdbc:mysql://localhost:
3306/akses","root","");
 Statement st=conn.createStatement();
```

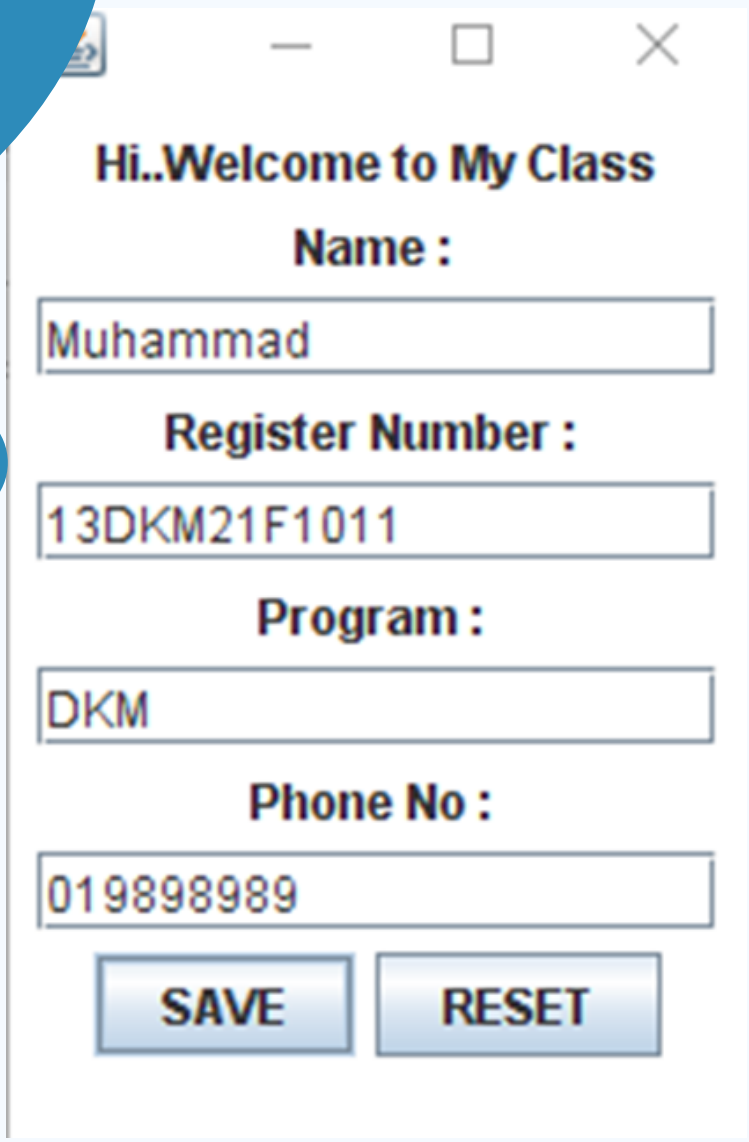
Example 3 - Insert Record..Cont



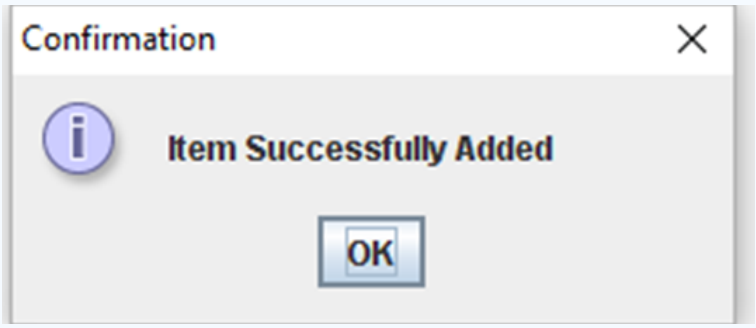

```
// STEP 4.1 : Execute Query (insert data using java code)
int  i= st.executeUpdate("insert into baru(name,matric,program,nohp)values('"+Tname.getText()+"','"+Tmatric.getText()+"','"+Tprogram.getText()+"','"+Tphone.getText()+"')");

JOptionPane.showMessageDialog(null,"Item Successfully Added","Confirmation",JOptionPane.INFORMATION_MESSAGE);
    //setVisible(false);
   }
   catch(Exception ei)
   {
    System.out.println("SQL code does not execute");
   }
         }
         else
         {
          Tname.setText("");
          Tmatric.setText("");
          Tprogram.setText("");
          Tphone.setText("");
          JOptionPane.showMessageDialog(null," The Form Already Reset");
         }
     }
}
class WindowEventHandler extends WindowAdapter
{
 public void windowClosing(WindowEvent e)
 {
  System.exit(0);
 }
}
```

Example 3 - Insert Record..Cont

output



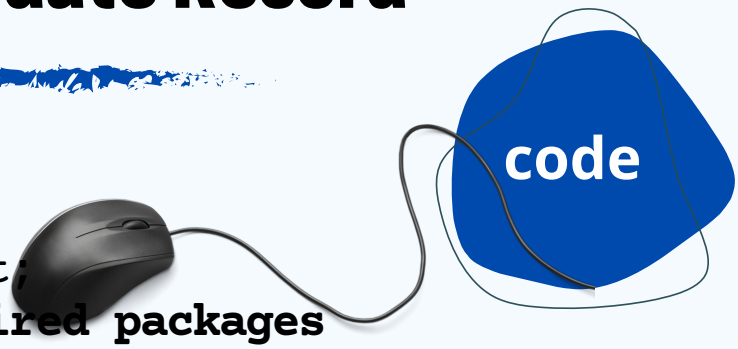
NOTES :
After click Save Button.



NOTES :
New data update in the table ..

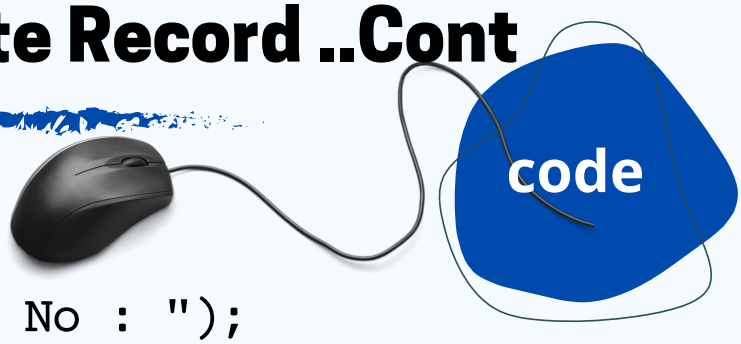
name	matric	program	nohp
Ahmad	10422	DDT	018754821
Muhammad	13DKM21F1011	DKM	019898989

Example 4 - Update Record



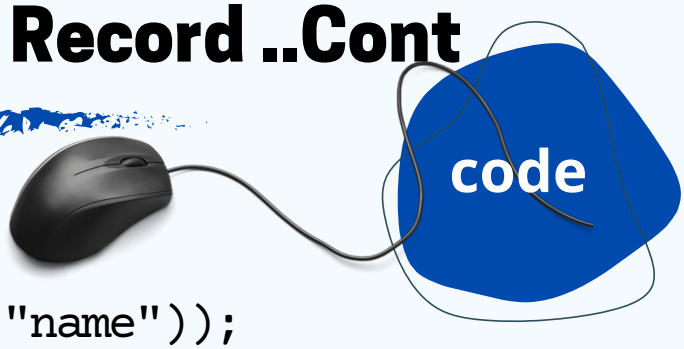

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.FlowLayout;
// STEP 1.1 : import Required packages
import java.sql.*;

public class formUpdate extends Frame implements
ActionListener
{
 JLabel tajuk, tajuk2,Lnama, Lnomatrik, Lprogram,
 Lphone, jabatan,phone,Lsearch;
 JTextField Tname, Tmatric, Tprogram, Tphone,Tsearch;
 JButton Bupdate,Bsearch;

 //STEP 1.2 :Declare variable for database connection
  Connection conn;
  PreparedStatement pst;
  ResultSet rs;
  ResultSet rs1;
  ResultSet rs2;
  public formUpdate ()
   {
   setLayout(new FlowLayout());
   Lsearch = new JLabel (" Insert Name : " );
   Tsearch = new JTextField (15);
   Bsearch=new JButton("SEARCH");
   add(Lsearch);add(Tsearch);add(Bsearch);
   tajuk = new JLabel (" Hi..Welcome to My Class " );
   Bupdate=new JButton("UPDATE");
   Lnama = new JLabel("Name : ");
   Tname = new JTextField (15);
   Lnomatrik = new JLabel("Register Number : ");
   Tmatric = new JTextField (15);
   Lprogram = new JLabel("Program : ");
   Tprogram = new JTextField (15);
```

Example 4 - Update Record ..Cont




```
 Lphone = new JLabel("Phone No : ");
 Tphone = new JTextField (15);
 add(tajuk);add(Lnama);add(Tname);add(Lnomatrik);
 add(Tmatric);add(Lprogram);add(Tprogram);
 add(Lphone);add(Tphone);add(Bupdate);
 Bsearch.addActionListener(this);
 Bupdate.addActionListener(this);

 setSize(200,500);setVisible(true);
 addWindowListener(new WindowEventHandler());
}
 public static void main(String[] args)
 {
 formUpdate f = new formUpdate ();
 }
 public void actionPerformed(ActionEvent e)
 {
      Object pilihan=e.getSource();
        if (pilihan==Bsearch)
        {
         try{
   //STEP 2 : Register JDBC  drver using mysql
    Class.forName("com.mysql.jdbc.Driver");
   //STEP 3 : Open Connection

conn=DriverManager.getConnection("jdbc:mysql://localh
ost:3306/akses","root","");
    Statement st=conn.createStatement();
   // STEP 4.1 : Execute Query (insert data using
java code)
    rs=st.executeQuery("SELECT * FROM baru wh
name='"+Tsearch.getText()+"'");
```

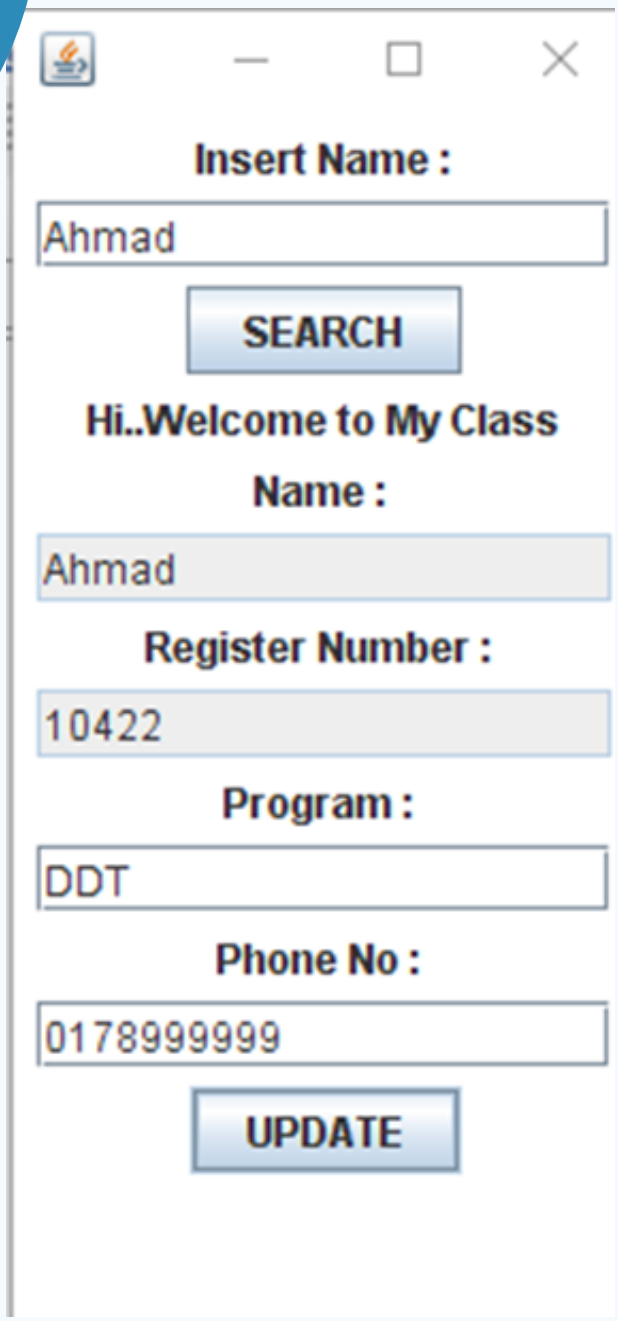
Example 4 - Update Record ..Cont



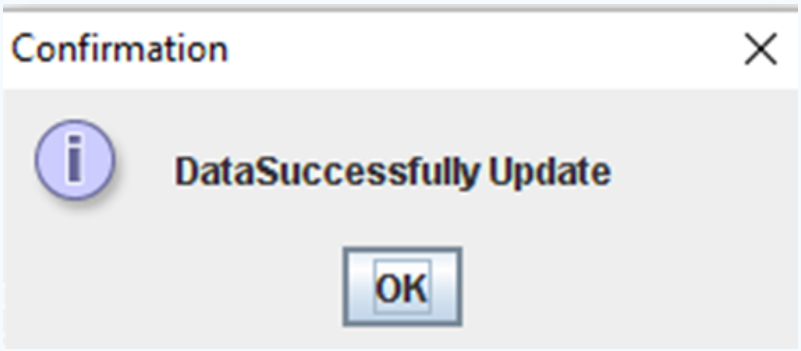

```
  if(rs.next())
    {
      Tname.setText(rs.getString("name"));
      Tname.setEditable(false);
      Tmatric.setText(rs.getString("matric"));
      Tmatric.setEditable(false);
      Tprogram.setText(rs.getString("program"));
      Tphone.setText(rs.getString("nohp"));
      JOptionPane.showMessageDialog(null,"Item
Successfully Retrieve" ,
"Confirmation",JOptionPane.INFORMATION_MESSAGE);;}
      else
     { JOptionPane.showMessageDialog(null,"Data Not
Found");}
     }
   catch(Exception ei)
    {
     System.out.println("SQL code does not execute");
    }
 }
   else if (pilihan==Bupdate)
 { try {
    Class.forName("com.mysql.jdbc.Driver");
conn=DriverManager.getConnection("jdbc:mysql://localhost
:3306/akses","root","");
    Statement st=conn.createStatement();
    int rs2=st.executeUpdate("UPDATE baru set
program='"+Tprogram.getText()+"',nohp='"+Tphone.getText(
)+"'where name='"+Tname.getText()+"'");
    JOptionPane.showMessageDialog(null,"DataSuccessfully
Update","Confirmation",JOptionPane.INFORMATION_MESSAGE);
          }
   catch(Exception ei) {
    System.out.println("SQL code does not execute");
    }
  }
 }
```

Example 4 - Update Record ..Cont

output



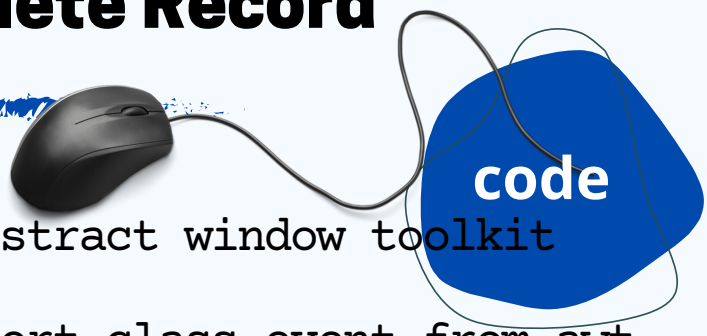
NOTES :
After click Update Button.



NOTES :
New data update in the table ..

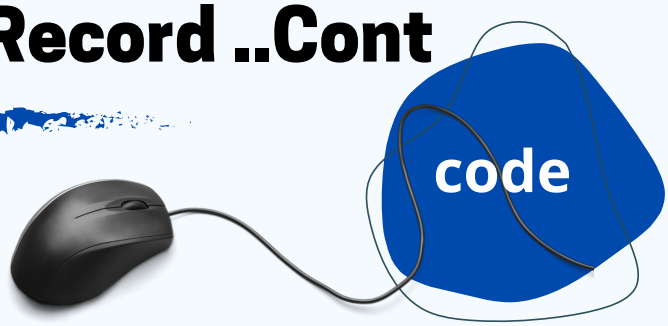
name	matric	program	nohp
Ahmad	10422	DDT	0178999999
Muhammad	13DKM21F1011	DKM	019898

Example 5 - Delete Record




```
import java.awt.*; // import abstract window toolkit
package
import java.awt.event.*; // import class event from awt
package
import javax.swing.*;
import java.awt.FlowLayout;
// STEP 1.1 : import Required packages
import java.sql.*;

public class formDelete extends Frame implements
ActionListener
{
 JLabel tajuk, tajuk2,Lnama, Lnomatrik, Lprogram,
 Lphone, jabatan,phone,Lsearch;
 JTextField Tname, Tmatric, Tprogram, Tphone,Tsearch;
 JButton Bsearch,Bdelete;

 // STEP 1.2 : Declare variable for connection database
  Connection conn;
  PreparedStatement pst;
  ResultSet rs;
  ResultSet rs1;
  ResultSet rs2;

  public formDelete ()
    {
     setLayout(new FlowLayout( ));
     Lsearch = new JLabel (" Insert Name : " );
     Tsearch = new JTextField (15);
     Bsearch=new JButton("SEARCH");
     add(Lsearch);
     add(Tsearch);
     add(Bsearch);
     tajuk = new JLabel (" Hi..Welcome to My Class " );
```

Example 5 - Delete Record ..Cont

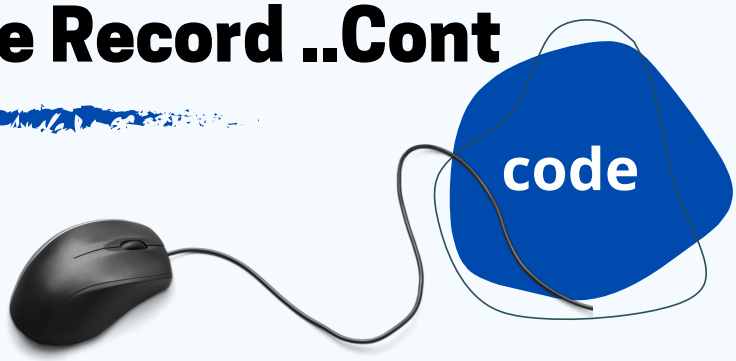



```
 Bdelete=new JButton("DELETE");
 Lnama = new JLabel("Name : ");
 Tname = new JTextField (15);
 Lnomatrik = new JLabel("Register Number : ");
 Tmatric = new JTextField (15);
 Lprogram = new JLabel("Program : ");
 Tprogram = new JTextField (15);
 Lphone = new JLabel("Phone No : ");
 Tphone = new JTextField (15);
 add(tajuk);
 add(Lnama);
 add(Tname);
 add(Lnomatrik);
 add(Tmatric);
 add(Lprogram);
 add(Tprogram);
 add(Lphone);
 add(Tphone);
 add(Bdelete);
 Bsearch.addActionListener(this);
 Bdelete.addActionListener(this);
 setSize(200,500);
 setVisible(true);
 addWindowListener(new WindowEventHandler());
}
 public static void main(String[] args)
 {
  formDelete  f = new formDelete();
 }
```

Example 5 - Delete Record ..Cont


```
public void actionPerformed(ActionEvent e)
 {
   Object pilihan=e.getSource();
   if (pilihan==Bsearch)
   {
    try{
   //STEP 2 : Register JDBC  drver using mysql
    Class.forName("com.mysql.jdbc.Driver");
   //STEP 3 : Open Connection

conn=DriverManager.getConnection("jdbc:mysql://localhost
:3306/akses","root","");
    Statement st=conn.createStatement();
   // STEP 4.1 : Execute Query (insert data using java
code)
    rs=st.executeQuery("SELECT * FROM baru where
name='"+Tsearch.getText()+"'");
   if(rs.next())
   {
     Tname.setText(rs.getString("name"));
     Tname.setEditable(false);
     Tmatric.setText(rs.getString("matric"));
     Tmatric.setEditable(false);
     Tprogram.setText(rs.getString("program"));
     Tphone.setText(rs.getString("nohp"));
     JOptionPane.showMessageDialog(null,"Item
Successfully Retrieve"
,"Confirmation",JOptionPane.INFORMATION_MESSAGE);;}
   else
    { JOptionPane.showMessageDialog(null,"Data Not
Found");}
         }
   catch(Exception ei)
   {
    System.out.println("SQL code does not execute");
   }
 }
```

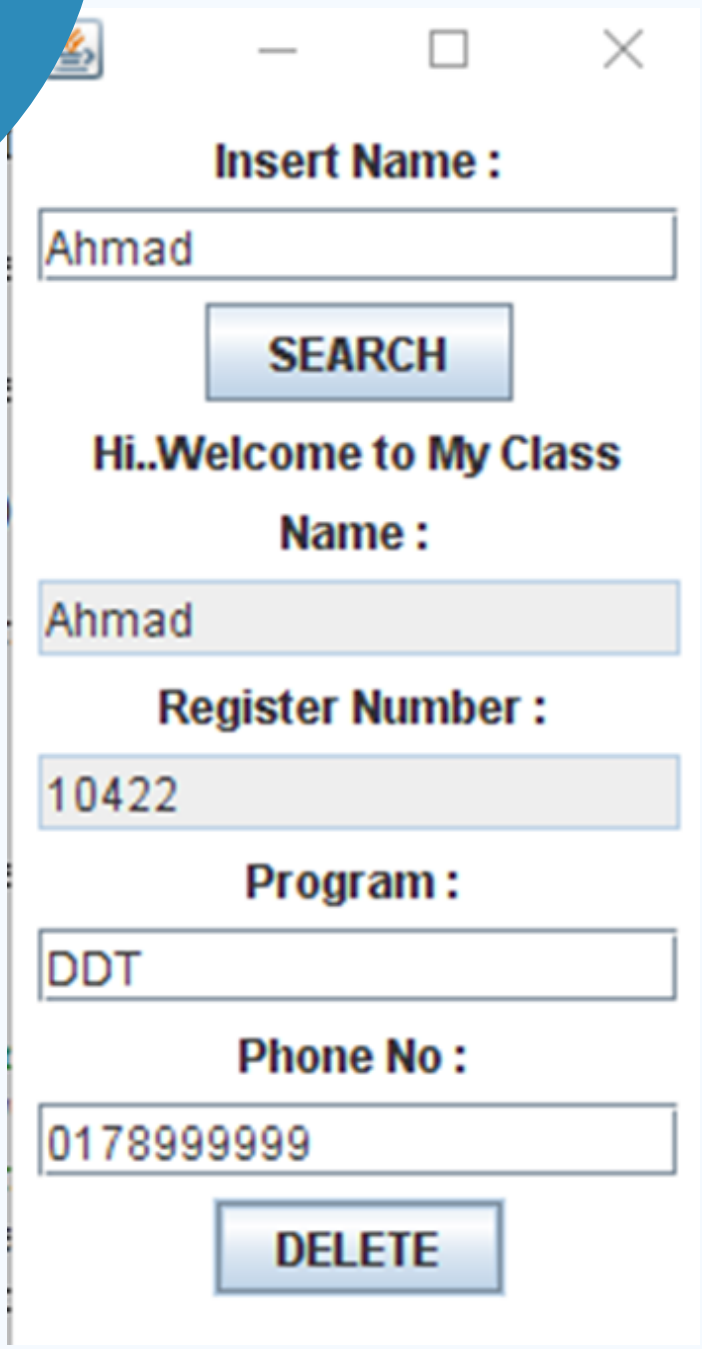
Example 5 - Delete Record ..Cont



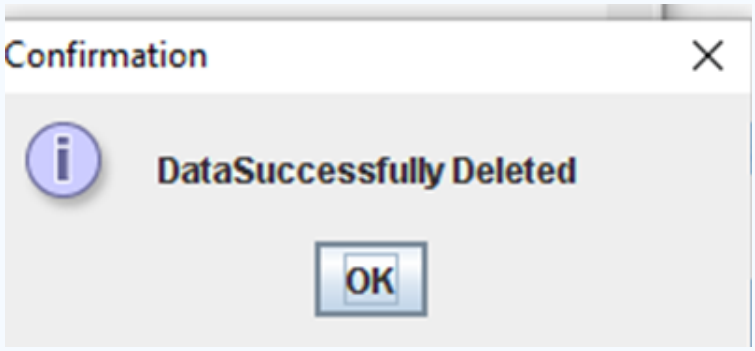

```
 else if (pilihan==Bdelete)
    {
      try
      {
   //STEP 2 : Register JDBC  drver using mysql
    Class.forName("com.mysql.jdbc.Driver");
   //STEP 3 : Open Connection

conn=DriverManager.getConnection("jdbc:mysql://localhost
:3306/akses","root","");
    Statement st=conn.createStatement();
   // STEP 4.1 : Execute Query (insert data using java
code)
    int rs2=st.executeUpdate("DELETE from baru where
name='"+Tname.getText()+"'");
    JOptionPane.showMessageDialog(null,"DataSuccessfully
Deleted","Confirmation",JOptionPane.INFORMATION_MESSAGE)
;

    }
   catch(Exception ei)
   {
    System.out.println("SQL code does not execute");
   }
  }
 }
}
```

Example 4 - Delete Record ..Cont

output



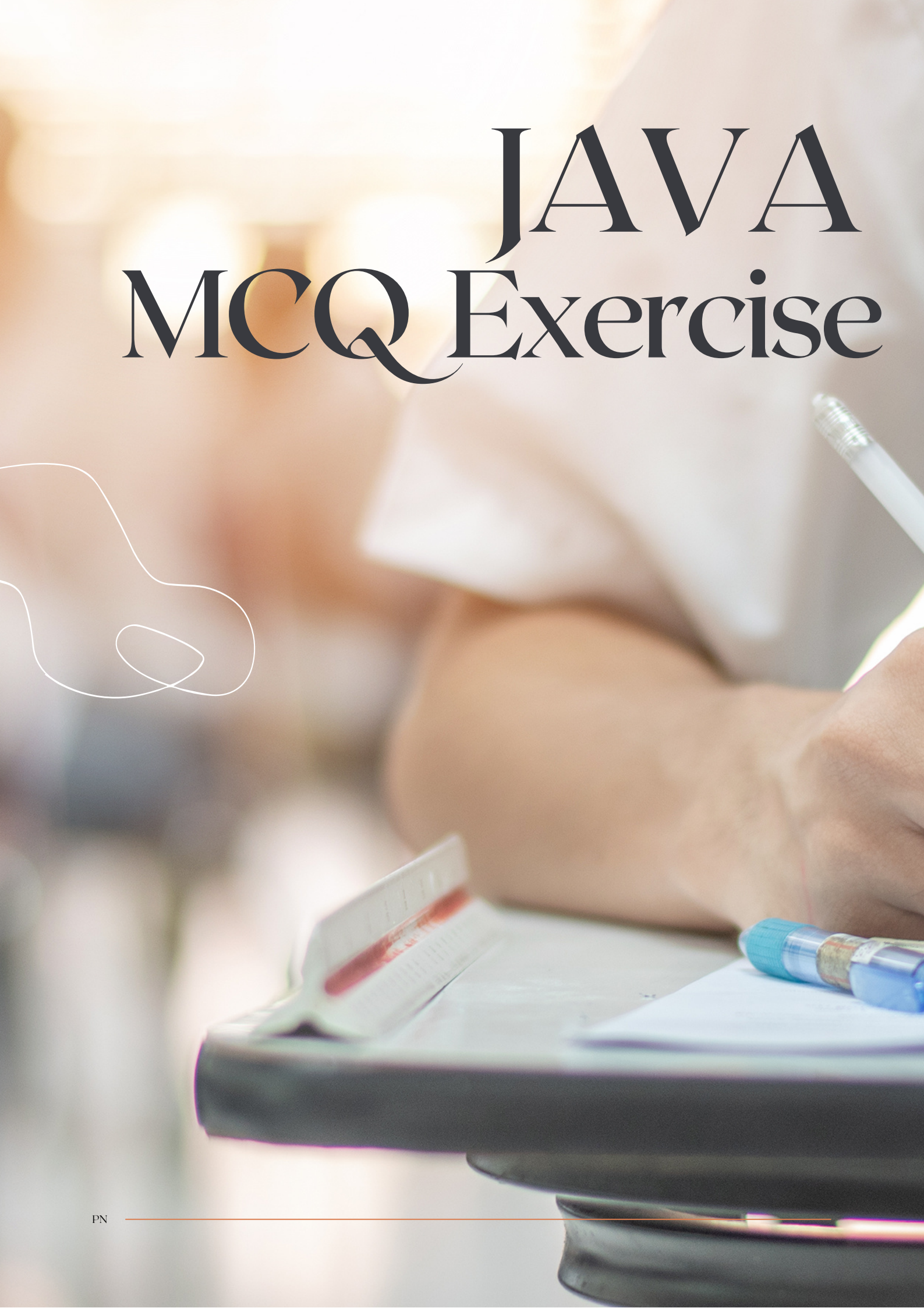
NOTES :
After click Delete Button.



NOTES :
New data update in the table ..

name	matric	program	nohp
Muhammad	13DKM21F1011	DKM	01989898

JAVA MCQ Exercise



MCQ Question

Chapter 1

Multi Choice Question

Chapter 1

scan here

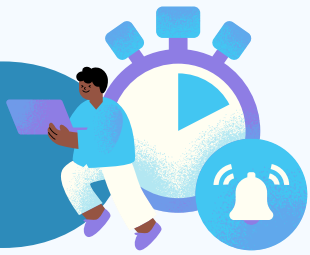
MCQ Question

Chapter 2

Multi Choice Question

Chapter 2

scan here



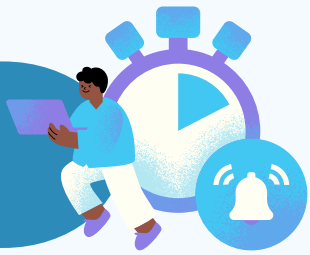
MCQ Question

Chapter 4

Multi Choice Question

Chapter 4

scan here



Written By

QUICK GUIDE

startup java program

Written By

NORHAYATI SA'ADAH CHE ABD RAZAK

SUZIWATI YUSOF

e ISBN 978-967-0047-39-3

9 789670 047393