2023

Internet Programming with JAVA

Dr. Babasaheb Ambedkar Open University

Internet Programming with JAVA

Course Writer

Mrs. Shital Patel

Dr. Ajay Patel

Dr. Mantavy Gajjar

Content Reviewer and Editor

Prof. (Dr.) Narayan Joshi

ISBN-

Printed and published by: Dr. Babasaheb Ambedkar Open University, Ahmedabad

Dr. Babasaheb Ambedkar Open University

MCA-303

Internet ProgrammingwithJAVA

Block-1: Java Swings

Block-2: JDBC(JavaDatabaseConnectivity)

Block-3: JavaNetworkProgramming

Block-4: Servlet and JSP

Block-1

.NET Java Swings

Unit 1: Fundamental of Swing

1

Unit Structure

1.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Differentiate about AWT and Swing.
- Introduce various GUI Components of swing.
- Know features of Swing.
- Different packages use in Swing.

1.2 INTRODUCTION

Now a day, most programmers use Swing for creating user interfaces. Java Swing is a part of Java Foundation Classes (JFC) which was designed for enabling large-scale enterprise development of Java applications. Java Swing is a set of APIs that provides graphical user interface (GUI) for Java programs. Java Swing is also known as Java GUI widget toolkit.

Java Swing or Swing was developed based on earlier APIs called Abstract Windows Toolkit (AWT). Swing provides richer and more sophisticated GUI components than AWT.Swing is a set of Classes that provides more powerful and flexible GUI components than does the AWT. Swing provides the look and feel of the modern Java GUI.

1.3 FUNDAMENTAL OF SWING

Swing API is a set of extensible GUI Components to ease the developer's life to create JAVA based Front End/GUI Applications. It is build on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls. Swing component follows a Model-View-Controller architecture to fulfill the following criteria.

- A single API is to be sufficient to support multiple look and feel.
- API is to be model driven so that the highest level API is not required to have data.
- API is to use the Java Bean model so that Builder Tools and IDE can provide better services to the developers for use.

Swing Architecture

Swing i s platform i ndependent and en hanced M VC (Model –View – Controller) framework for Java application.

- Model represents component's data.
- View represents determines how the component is displayed on the screen.
- Controller represents how the component reacts to the user.
- Swing co mponent has Model as a se parate el ement, w hile t he V iew an d Controller part are clubbed in the User Interface elements. Because of which, Swing has a pluggable look-and-feel architecture.

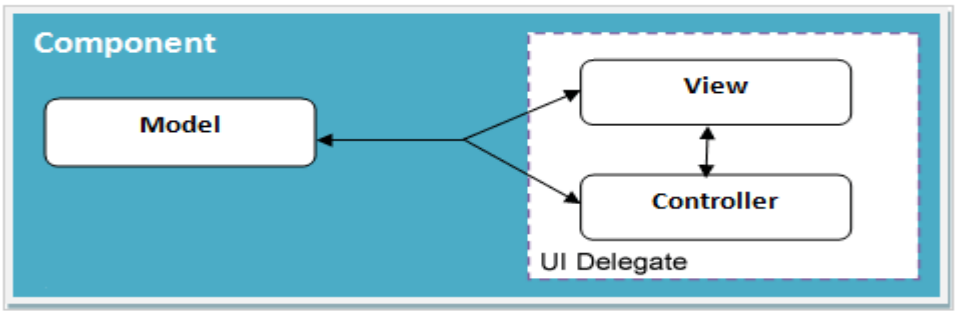


Figure-1.1 Java Swing MVC – Model Delegate

For example, w hen t he use r cli cks a check box, t he co ntroller r eacts bychanging t hemodel t o r eflect t he use r's choice (checked or unch ecked). This then results in the view being updated.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No	Java AWT	Java Swing
1.	AWT components are platform-dependent.	Java swing components are platform-independent.
2.	AWT components are heavyweight.	Swing components are lightweight.
3.	AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
4.	AWT provides less components than Swing.	Swing provides more powerfulcomponents such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.

5.	AWT doesn't follows MVC(Model View Controller).	Swing follows MVC.

Table-1 Difference between AWT and Swing

1.4 KEY FEATURES OF SWING

- **Light Weight:**Swing components are lightweight. This means that they are written totally in Java and do not map directly to platform-specific peers.Because lightweight components are rendered using graphics primitives, they can be transparent, which enables nonrectangular shapes. Thus, lightweight components are more efficient and more flexible. So each component of swing will work in a consistent manner across all platforms.
- **Rich Controls**: Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.
- **Highly Customizable:** Swing controls can be customized in a very easy way as visual appearance is independent of internal representation.
- **Pluggable look-and-feel:** SWING based GUI Application look and feel can be changed at run-time, based on available values.

1.5COMPONENTS& CONTAINERS

In Java, a component is the basic user interface object and is found in all Java applications. Components include JLists, JButtons, JLabel, JMenu etc.
To use components, you need to place them in a container.

A container is a component that holds and manages other components. Containers display components using a layout manager. Simply say a container holds a group ofcomponents.

Components

Swing components are inherit from the javax.swing. JComponent class, which is the root of the Swing component hierarchy.JComponent, in turn, inherits from the Container class in the Abstract Windowing Toolkit (AWT). So Swing is based on classes inherited from AWT.

All of S wing's co mponents are r epresented by classes defined within t he packagejavax. swing.

The following table-2 shows the class names for Swing components.

JButton	JCheckBox	JCheckBoxMenuItem	JColorChooser
JComboBox	JDesktopPane	JEditorPane	JFileChooser
JFormattedTextField	JLabel	JList	JMenu
JMenuBar	JMenuItem	JPasswordField	JPopupMenu
JProgressBar	JRadioButton	JRadioButtonMenuItem	JScrollBar
JSeparator	JSlider	JSpinner	JSplitPane
JTabbedPane	JTable	JTextArea	JTextField
JTextPane	JTogglebutton	JToolBar	JToolTip
JTree	JViewport		

Table-2 Swing components List

The hierarchy of java swing API is given below Figure-1.2.

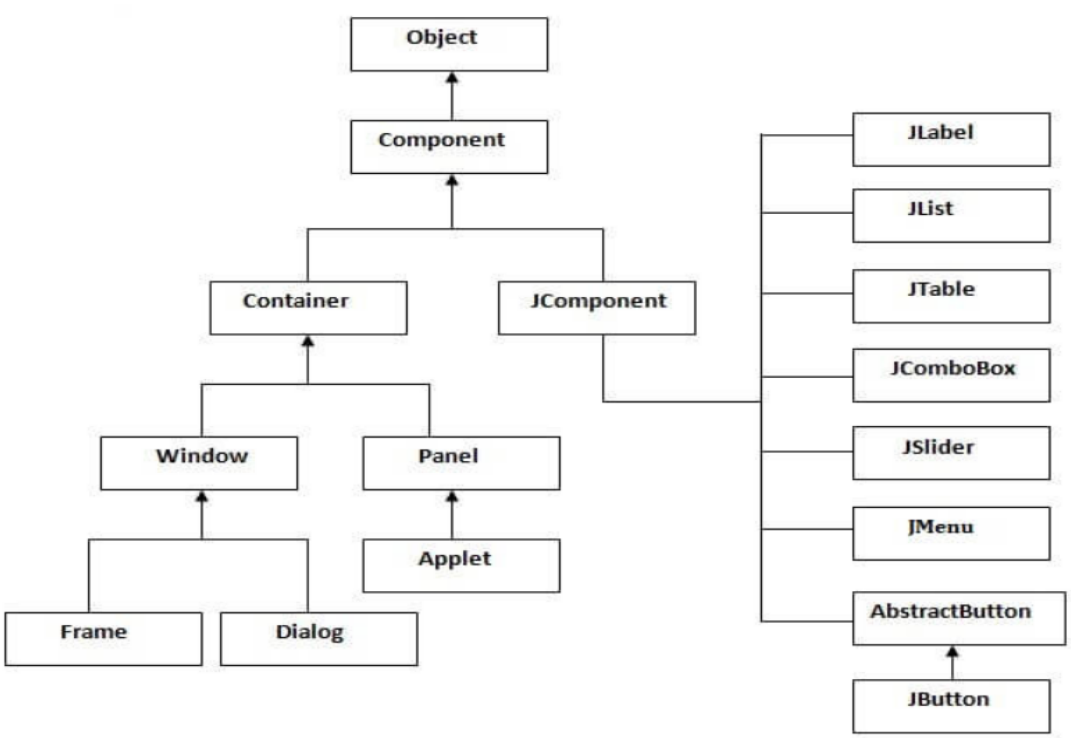


Figure-1.2 Hierarchy of Java Swing API.

Containers

Swing defines two types of containers. In top-level containers: JFrame, JApplet,JWindow, and JDialog. These containers do not inherit JComponent. They all are inherit the AWT classes Component and Container.

A top-level container must be place at the top of in hierarchy.A top-level container is not contained within any other container.The one most commonly used container for applications is JFrame and for applets is JApplet.

The second type of containers maintained by Swing are lightweight containers. Lightweightcontainers do inherit JComponent. An example of a lightweight container is JPanel. Lightweight containers are regularly used to organize andmanage groups of related components because a lightweight container can be contained within another container. So, you can use lightweight containers such as JPanel.

The following table-3 shows the names for Swing container.

JApplet	JDialog	JDesktopPane	JFrame
JEditorPane	JLayeredPane	JWindow	

Table-3 Swing containers List

Swing provides the following useful top-level containers, all of which inherit from JComponent:

Figure-1.3 Top-level Containers of Swing

All Swing components need to be contained inside a JWindow or JFrame.

The Top-Level Container Panes

Each top-level container defines a set of panes. Following figure show top-level container panes.

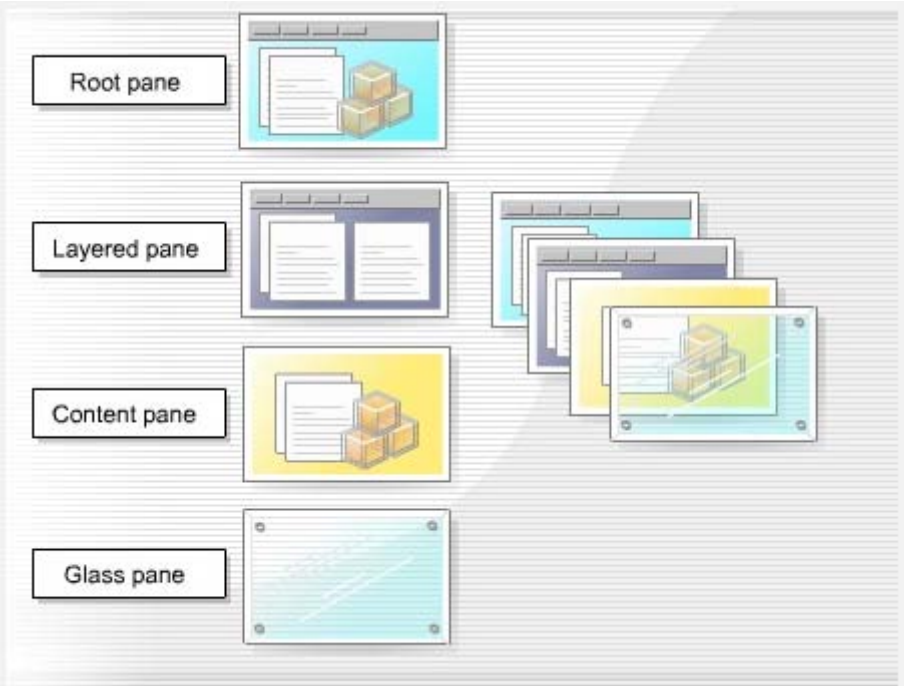


Figure-1.4 Top-level Containers of panes

Root pane

The root pane is an intermediate container that manages the layered pane, content pane, and glass pane. You use a root pane to paint over multiple components or to catch input events.

Layered pane

The layered pane contains the content pane and the optional menu bar.The layered pane provides six functional layers in which you place the components you add to it.

Content pane

The content pane holds all the visible components of the root pane, except the menu bar. It covers the visible section of the JFrame or JWindow and you use it to add

components to the display area. Java automatically creates a content pane when you create a JFrame or JWindow but you can create your own content pane, which has to be opaque.

Glass pane

The glass pane is invisible by default but you can make it visible. When it is visible, it covers the components of the content pane and can paint over an existing area containing one or more components.

1.6 SWING PACKAGES & APPLICATIONS

Swing Packages

Swing is a very large subsystem and makes use of many packages. These are the packages used by Swing.

javax.swing	javax.swing.border	javax.swing.colorchooser
javax.swing.event	javax.swing.filechooser	javax.swing.plaf
javax.swing.plaf.basic	javax.swing.plaf.metal	javax.swing.plaf.multi
javax.swing.plaf.synth	javax.swing.table	javax.swing.text
javax.swing.text.html	javax.swing.text.html.parser	javax.swing.text.rtf
javax.swing.tree	javax.swing.undo	

The main package is javax.swing. when user make any swing program then they must be imported javax.swing package. This package contains basic Swing components, such as buttons, labels, list, and check boxes.

Swing Applications

Swing programs differ from both the console-based programs and the AWT-based programs. Swing use a different set of components and a different container hierarchy than does the AWT.The best way to understand the structure of a Swing program is to work through a simple example.

the f ollowing pr ogram showa simple Swing application. In t his program,it demonstrates several key features of Swing. It uses two Swing components:JFrame and JL abel. JF rame i s the t op-level co ntainer t hat i s commonly used f or Swingapplications. JLabel is the Swing component that creates a label, which is use for displays information.

```
// A simple Swing application.
import javax.swing.*;
public class SwingDemo
{
   SwingDemo()
   {
        // Create a new JFrame
JFrame jf=new JFrame("A Simple Swing Program");
       // Give the frame an initial size.
      jf.setSize(400,300);
// Terminate the program when the user closes the application.
      jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// Create a text-based label.
      JLabel lb=new JLabel("Hi ....");
// Add the label to the content pane.
      jf.add(lb);
// Display the frame.
      jf.setVisible(true);
    }
   public static void main(String args[]) {
     SwingDemo sd=new Swing_Demo();
  }  }
```

Swing programs are compiled and run in the same way as other Java applications. So,to compile this program, you can use this command line:

javac SwingDemo.java

To run the program, use this command line:

java SwingDemo

Output of this program shown in Figure-1.5.

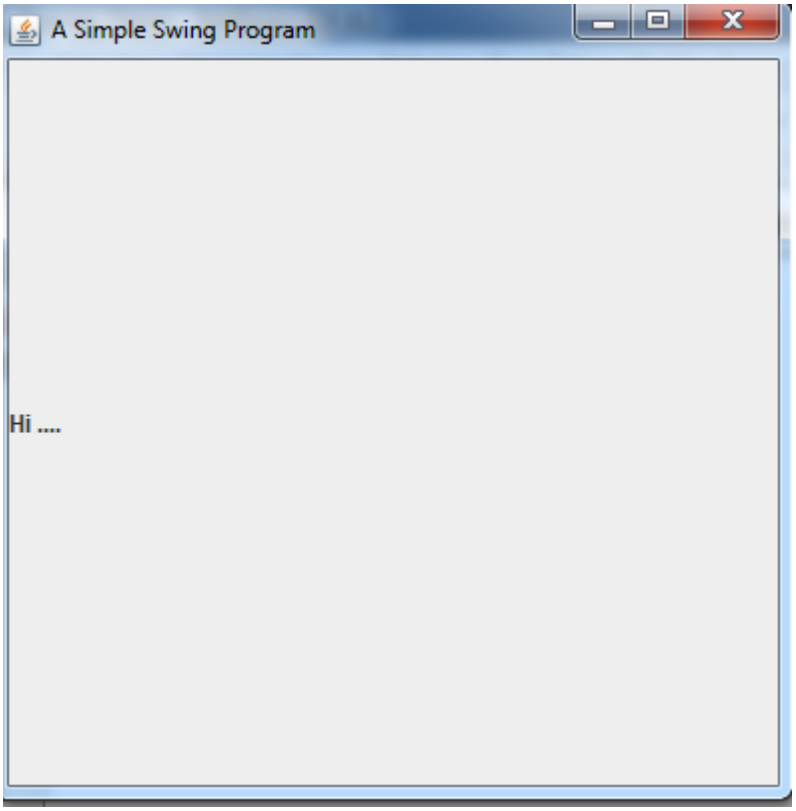


Figure-1.5 Output of SwingDemo program

In this program declares SwingDemo class and a constructor for that class.

creating a JFrame,using this line of code:

JFrame jf=new JFrame("A Simple Swing Program");

jf object s how a r ectangular w indow co mplete w ith a t itlebar; close , minimize, maximize, and restore buttons;

the window is sized using this statement:

jf.setSize(400,300);

The se tSize() m ethod w hich i s setsthe di mensions of t he w indow, w hich ar e specified in pixels. Its general form is shown here:

void setSize(int width, int height)

In this example, the width of the window is set to 400 and the height is set to 300.

when a top-level window is closed, the window is removed from the screen. For that callsetDefaultCloseOperation( ), as the program does:

jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

After calling t his method, closi ng t he w indow ca uses the ent ire appl ication t o terminate.

Thegeneral form of setDefaultCloseOperation( ) is shown here:

void setDefaultCloseOperation(int what)

The va lue passed i n w hat determines what happens when the w indow i s closed. There are many other options in addition to JFrame.EXIT_ON_CLOSE.

They are shown here:

JFrame.DISPOSE_ON_CLOSE : hides and disposes of the JFrame when the user closes it. Disposing a JFrame releases any resources used by it.

JFrame.HIDE_ON_CLOSE : hides a JF rame w hen t he use r cl oses it. Th is is the default behavior. The JFrame is invisible but the program is still running.

JFrame.DO_NOTHING_ON_CLOSE : exits the appl ication. Thi s option w ill e xit the application.

Next,

jf.setVisible(true);

The setVisible( ) methodis inherited from the AWT Component class.If its argument istrue, th en window w ill be di splayed. O therwise, i t w ill be hi dden. By default, a JFrame isinvisible, so setVisible(true) must be called.

1.7 PAINTING FUNDAMENTALS

Components of swing are very powerful. Swing components are directly display into frame and panel. Swing will not allow to draw directly to the surface of component. Using AWT class component have a method like paint(), that is used to draw output directly on the surface of a component and the methods are like drawLine(), drawRect, etc.

JComponent inherits Componentclass, all Swing's lightweight components inheritthe paint( ) method. However, you will not override it to paint directly to the surface of acomponent. The reason is that Swing uses a bit more sophisticated approach to painting thatinvolves three distinct methods: paintComponent( ), paintBorder( ), and paintChildren( ).These methods paint the indicated part of a component and divide the painting process in its three distinct logical actions.

To paint to the surface of a Swing component, you will create a subclass of the componentand then override its paintComponent( ) method. This is the method that paints the interiorof the component. You will not normally override the other two painting methods such as paintBorder( ) and paintChildren( ).

The paintComponent( )method is shown here:

protected void paintComponent(Graphics g)

The parameter g is the graphics context to which output is written.

In the following program, we make a subclass of JPanel and override one method, paintComponent().

```
import java.awt.*;
import javax. swing.*;
public class swingpaintdemo extends JPanel
{
   public void paintComponent(Graphics g)
   {
      g.setColor(Color.orange);
```

```
        g.drawLine(10,50,50,20);
        g.setColor(Color.red);
        g.fillOval(getWidth()/4, getHeight()/4, getWidth()/2, getHeight()/2);
  }
    public static void main(String args[])
    {
// Create a new JFrame container.
    JFrame jf =new JFrame("Use PaintComponent() Method ");
          // Give the frame an initial size.
jf.setSize(350,300);
        jf.setVisible(true);
// Add the panel to the content pane. Because the default// border layout is used, //
       the panel will automatically besized to fit the center region.
        swingpaintdemo sw=new swingpaintdemo();
        jf.add(sw);
      }
}
```

Output of this program shown in Figure-1.6.

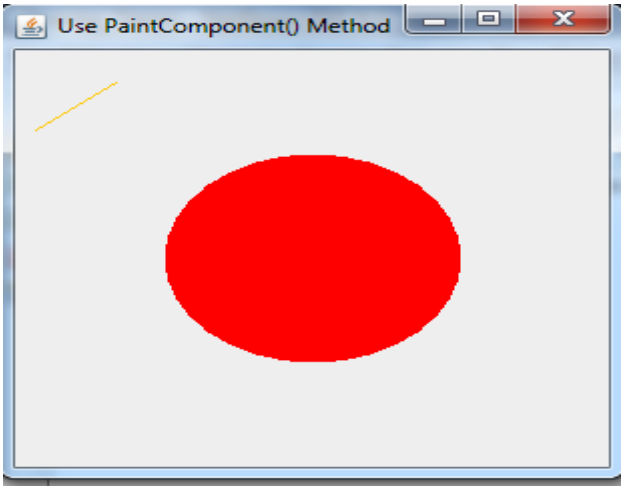


Figure-1.6 Output of paintComponent( ) program

In this program swingpaintdemo class extends JPanel. JPanel is oneof Swing's lightweight containers, which means that it is a component that can be added tothe content pane of a JFrame. To handle painting, swingpaintdemo overrides the paintComponent( ) method. This enables swingpaintdemo to write directly to the surface of the component whenpainting takes place. The size of the panel is not specified because the program uses thedefault border layout and the panel is added to the center. This results in the panel beingsized to fill the center. If you change the size of the window, the size of the panel will beadjusted accordingly.

1.8LET US SUM UP

- A swing is a set of classes that provides more powerful and flexible components that is possible with the AWT. It is defined within the package javax.swing.
- As compared to AWT components, swing components are known as lightweight components.
- The JApplet class is an extended version of java.applet. Applet that adds support for the JFC/Swing component architecture.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.
- Swing provides graphical user interface components to develop Java applications.
- The size of a frame is defined by its width and height in pixels and we can set them using setSize(int width, int height) method.
- The content pane from JFrame holds the Swing components of a JFrame.
- The pack( ) method of the JFrame examines all the components on the JFrame and decides their preferred size and sets the size of the JFrame just enough to display all the components.

1.9CHECK YOUR PROGRESS

1. Where are the following four methods commonly used?

   1) public void add(Component c)

   2) public void setSize(int width,int height)

   3) public void setLayout(LayoutManager m)

   4) public void setVisible(boolean)

   a. Graphics class **b.** Component class **c.** Both A & B **d.** None of the above

2. Which is the container that doesn't contain title bar and MenuBars but it can have other components like button, textfield etc?

   a. Window **b.** Frame **c.** Panel **d.** Container

   These two ways are used to create a Frame

   By creating the object of Frame class (association)

   By extending Frame class (inheritance)

   a. True **b.** False

3. Give the Full of AWT?

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

   a. True **b.** False

5. The following specifies the advantages of
   It is lightweight.
   It support pluggable look and feel.
   It follow MVC (Model view controller) Architecture.

   a. Swing **b.** AWT **c.** Both a and b **d.**None of above

6. The swing related classes are contained in

   a. javax.swing **b.** javax.awt **c.** javax.Swing **d.**None of above

1.10CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. b.Component class
2. c. Panel
3. a. true
4. Abstract Windowing Toolkit
5. a. true
6. a. Swing

1.11 FURTHER READING

Many courses require students to read some extra material in addition to theirunits. Sometimes a text requires 'readings' which must be obtained by alllearners. Such texts are usually referred to as 'essential texts'. Someinstitutions call them 'set texts'. On other occasions, students are expected toread widely from a variety of books, but the readings are entirely optional.

These books are referred to as 'recommended texts' or background reading.The distinction is important, as books are usually difficult to obtain and theavailability and price of essential books must be checked before they arespecified as compulsory. A course that has no recommended textbooks isknown as a self-contained course.

Following are some examples:

Koul, B. N. and Ghaudhary, Sohanvir (1989). Self-instructional course units - IGNOU Handbook5. New Delhi: Indira Gandhi National Open University.

Thompson, Bruce (2003). Introduction to open learning and instructional design for openlearning. Vancouver: Commonwealth of Learning (COL).

1.12ASSIGNMENTS

1. What is difference between AWT and Swing?
2. _____method use to visible JFrame.
3. Give name of constant which are used in setDefaultCloseOperation() method.
4. What is a container class?
5. What are the key feature of swing class?
6. List out Swing class.
7. Write a two ways to create a frame.

1.13ACTIVITIES

1. Create JFrame with 300 X 300 size, and display “Good Moring “message on JFrame.
2. Create Application for drawing Line, Rectangle , Circle and also fill all shapes.

Unit 2: Swing Components and Event Handling

2

Unit Structure

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- To understand the Java event-handling model.
- To understand the relationship of a JFrame and theobjects it contains.
- Working with containers control – JFrame, JApple and JPanel
- Working with basic control- JButton, JLabel,JTextField, JPasswordField.
- Working with selection control - JCheckBox, JRadioButton, , JList,and JComboBox.
- Working with JScrollPane control.

2.2 INTRODUCTION

The previous chapter contains several of the core concepts relating to Swing. This Chapter presenting overview of several swing components.Swing components are derived from the JComponent class. The only exceptions are the four top-level containers: JFrame, JApplet, JWindow, and JDialog. JComponent inherits AWT classes Container and Component.JComponent inherits AWT classes Container and Component.All the Swing components are represented by classes in the javax.swing package.All the component classes start with J: JLabel, JButton, JScrollbar,etc. The Swing componentsprovide rich functionality and allow a high level of customization.

2.3 WORKING WITH JFRAME

The javax.swing.JFrame class is a type of container which inherits thejava.awt.Frame class. JFrame works like the main window where components like JLabels, JButtons, JTextfields are added to create a GUI.

JFrame class has many constructors used to create a JFrame.Following is the description.

- JFrame(): creates a frame which is invisible.
- JFrame(GraphicsConfiguration gc): creates a frame with a blank title and graphics configuration of screen device.
- JFrame(String title): creates a JFrame with a title.

- JFrame(String title, GraphicsConfiguration gc): creates a JFrame with specific Graphics configuration and specified title.

Here is a simplest example just to create a JFrame with set title.

```
import javax.swing.*;

public class JFrameDemo

{

        public static void main(String args[])

        {

                JFrame jf=new JFrame("My Programe");

                jf.setSize(300,100);

                jf.setVisible(true);

                jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

}

 }
```

The output of program display in Figure 2.1

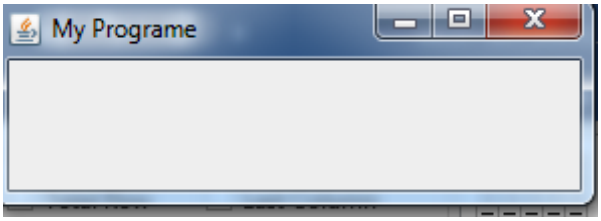


Figure 2.1: Output of JFrame with Title.

2.4 JApplet and JPanel

2.4.1 The JApplet class

Swing-based appl etsare sim ilar t o A WT-based app lets, but w ith an i mportant difference: A S wing applet e xtendsJApplet r ather t han A pplet. JAppl et i s derived from Applet.JApplet is atop-level container.

Swing appl ets use t he sa me f our l ifecycle m ethods which i s in AWT: i nit( ),start( ), stop( ), and destroy( ). So, you need override only those methods that areneeded by

your applet. Painting is accomplished differently in Swing than it is in the AWT,and a Swing applet will not normally override the paint( ) method.

Following program is a simplet example to create a JApplet with JLabel.

```
import java.awt.FlowLayout;
import javax.swing.*;
//This HTML can be used to create Applet with 300 x 300 size.
//<applet code="JappletDemo.class" width="300" height="300"></applet>
public class JappletDemo extends JApplet
 {
    JLabel l;
// Initialize the applet using init().
    public void init()
    {
// Set the applet to use flow layout.
        setLayout(new FlowLayout());
// Create a text-based label.
        l=new JLabel("Demo Program for JApplet");
// Add the label to the content pane.
        add(l);
    }
    }
```

The output of program display in Figure 2.2

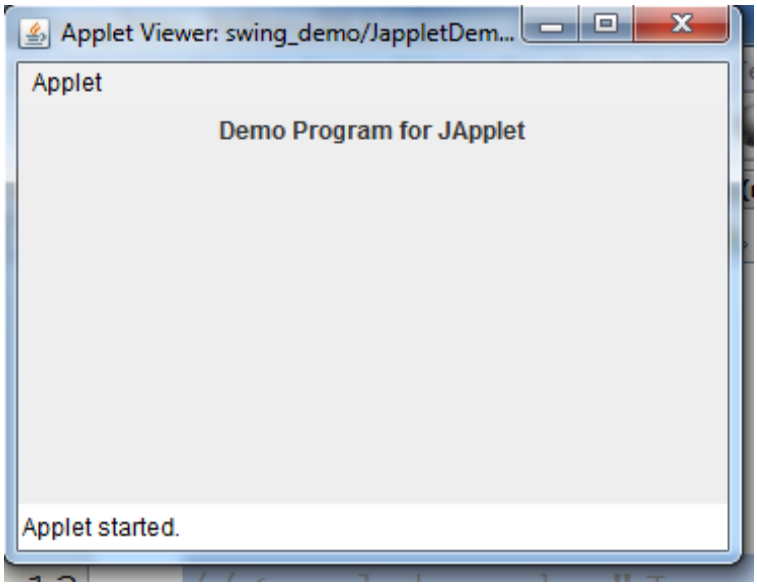


Figure 2.2: Output of JApplet with JLabel.

2.4.2 The JPanel Class

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

Constructors

JPanel() : It is used to create a new JPanel with a double buffer and a flow layout.

JPanel(boolean isDoubleBuffered) : It is used to create a new JPanel with FlowLayout and the specified buffering strategy.

JPanel(LayoutManager layout) : It is used to create a new JPanel with the specified layout manager.

Following program is a simplet example to create a JPanel.

```
import java.awt.*;
import javax.swing.*;
public class PanelExample
{
    PanelExample()
```

```
        {
        JFrame f= new JFrame("Panel Example");
        JPanel panel=new JPanel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        JButton b1=new JButton("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        JButton b2=new JButton("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[])
        {
    PanelExample p=new PanelExample();
        }
      }
```

Output show in Figure:2.3

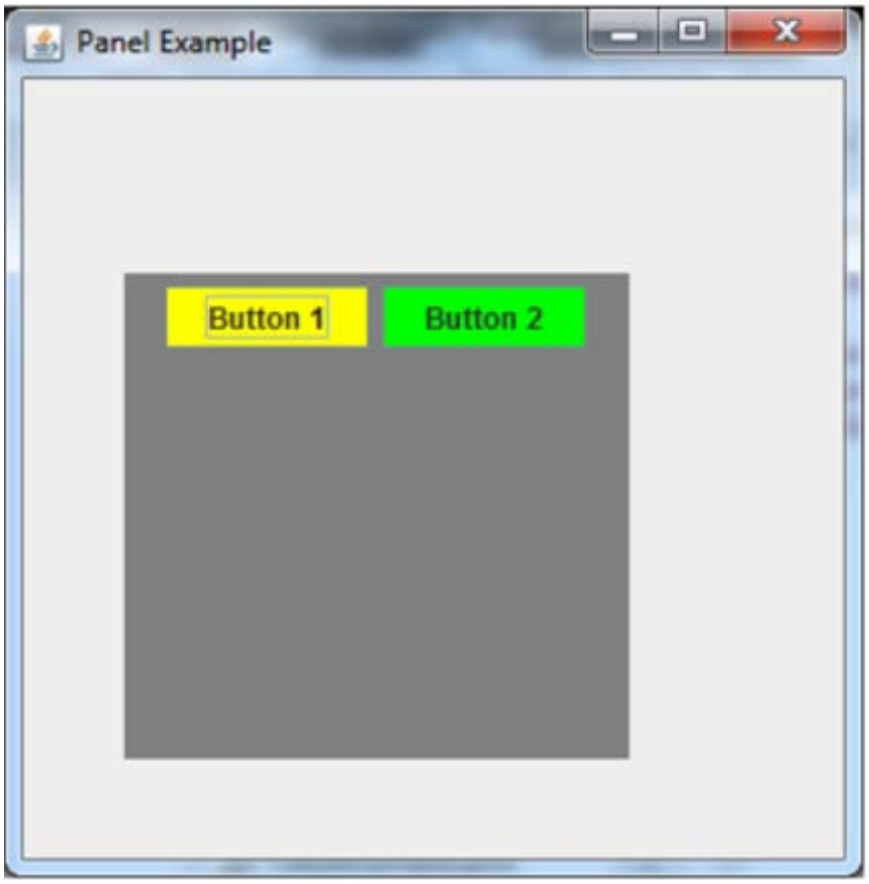


Figure 2.3 : Output of JPanel class

2.5JTextField, JPasswordField, JButton

2.5.1 JTextField

JTextfField is a lightweight component that allows the ending of a single line of text. The class has JTextComponent as its base class which in turn inherits JComponents class.

Constructor of JTextField are shown below.

JTextField() : Creates a new TextField.

JTextField(String text) : Creates a new TextField initialized with the specified text.

JTextField(String text, int columns) : Creates a new TextField initialized with the specified text and columns.

JTextField(int columns) : Creates a new empty TextField with the specified number of columns.

The Methods of JTextField class are given in the below table 2.1.

Method Name	Description
void setEdittable(Boolean b)	Sets the specified Boolean to indicate whether or not this text field should be editable.
Boolean isEditable()	Return the Boolean indicating whether this text field is editable or not.
String getText()	Return the text contained in this text field.
void setText(String t)	Sets the text of this text field to the specified text.

Table 2.1: Methods of JTextField class.

Program of JTextField is shown below.

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
   {
JFrame f= new JFrame("TextField Example");
JTextField t1,t2;
t1=new JTextField();
t1.setBounds(50,100, 200,30);
     t1.setText(“ Hello”);
     t2=new JTextField("Welcome to Javatpoint.");
t2.setBounds(50,150, 200,30);
f.add(t1);
f.add(t2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
   }
   }
```

Output of program is shown in Figure-2.4.

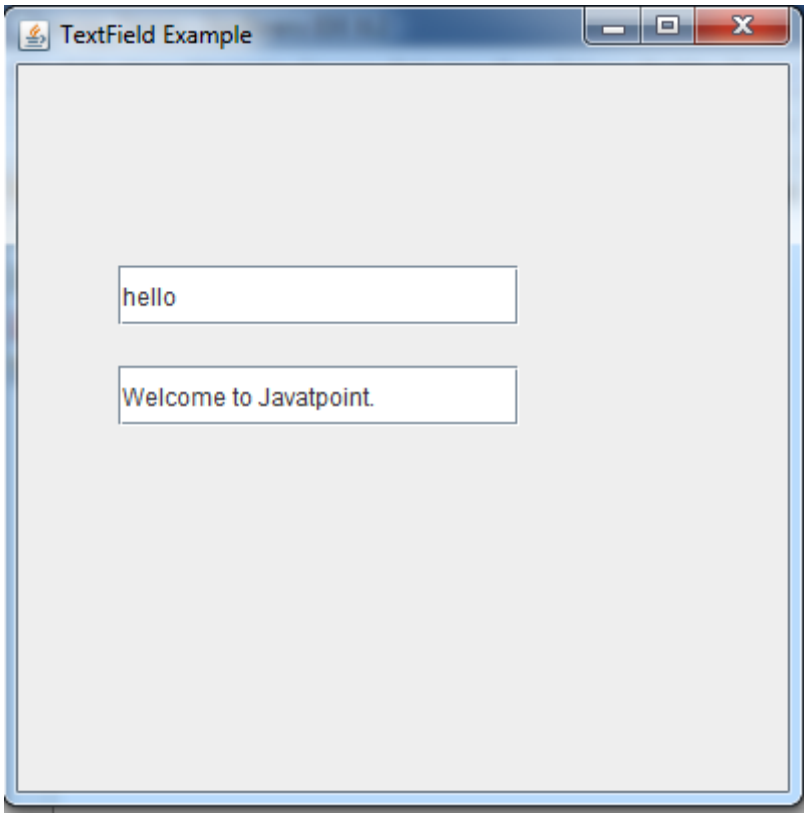


Figure 2.4 : Output of JTextField class

2.5.2 JPasswordField

JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

Constructor of JPasswordField are shown below.

JPasswordField(): Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.

JPasswordField(int columns) : Constructs a new empty JPasswordField with the specified number of columns.

JPasswordField(String text) : Constructs a new JPasswordField initialized with the specified text.

JPasswordField(String text, int columns) : Construct a new JPasswordField initialized with the specified text and columns.

The Methods of JPasswordField class are given in the below table 2.2.

Method Name	Description
char getEchoChar()	Returns the character to be used for echoing.
void setEchoChar(char c)	Sets the echo character for this JPasswordField.
String getText()	Return the text contained in this text field.
void setText(String t)	Sets the text of this text field to the specified text.
String getPassword()	returns the text contained in JPasswordField.

Table 2.2: Methods of JPasswwordField class.

Program of JPasswordField is shown below Figure-2.5.

Figure 2.5 : Output of JTextField class

2.5.3 JButton

The JButton class provides the functionality of a push button. JButton allows an icon, a string, or both to beassociated with the push button.

Constructors

JButton() : It creates a button with no text and icon.

JButton(String s) : It creates a button with the specified text.

JButton(Icon i) : It creates a button with the specified icon object.

JButton(String s , Icon icon) : It creates a button with the specified text and icon object.

The Methods of JButton class are given in the below table 2.3.

Method Name	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Table 2.3: Methods of JButton class.

When the button is pressed, an ActionEvent is generated. The ActionEvent objectpassed to the actionPerformed( ) method which is registered by ActionListener, you can obtainthe action command string associated with the button. By default, this is the string displayedinside the button. However, you can set the action command by calling setActionCommand( )on the button. You can obtain the action command by calling getActionCommand( ) on theevent object.

// Program to create three button and when button press according frame background color will change.

```
import javax.swing.*;

import java.awt.event.*;

import java.awt.*;

public class ButtonDemo extends JFrame implements ActionListener

{

   JLabel l1;

   JButton b1,b2,b3;
```

```
 ButtonDemo()
 {
     setLayout(new FlowLayout());
     setSize(400,700);
     setTitle("Java program Buttons Clicked");
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     setVisible(true);

l1=new JLabel("What is happening");
     add(l1);

     b1=new JButton("Red");
     add(b1);
     b2=new JButton("Green");
     add(b2);
     b3=new JButton("Blue");
     add(b3);

     b1.addActionListener(this);
     b2.addActionListener(this);
     b3.addActionListener(this);

 }
 public void actionPerformed(ActionEvent e)
 {
     if(e.getSource()== b1)
```

```
        {
           getContentPane().setBackground(Color.red);
            l1.setText("Set Color Red");
        }
        else if(e.getSource()== b2)
        {
            getContentPane().setBackground(Color.green);
            l1.setText("Set Color Green");
        }
        else if(e.getSource()== b3)
        {
        getContentPane().setBackground(Color.blue);
            l1.setText("Set Color Blue");
        }
    }
     public static void main(String args[])
     {
        ButtonDemo bd =new ButtonDemo();
     }
}
```

Output of the program is shown in Figure 2.6.

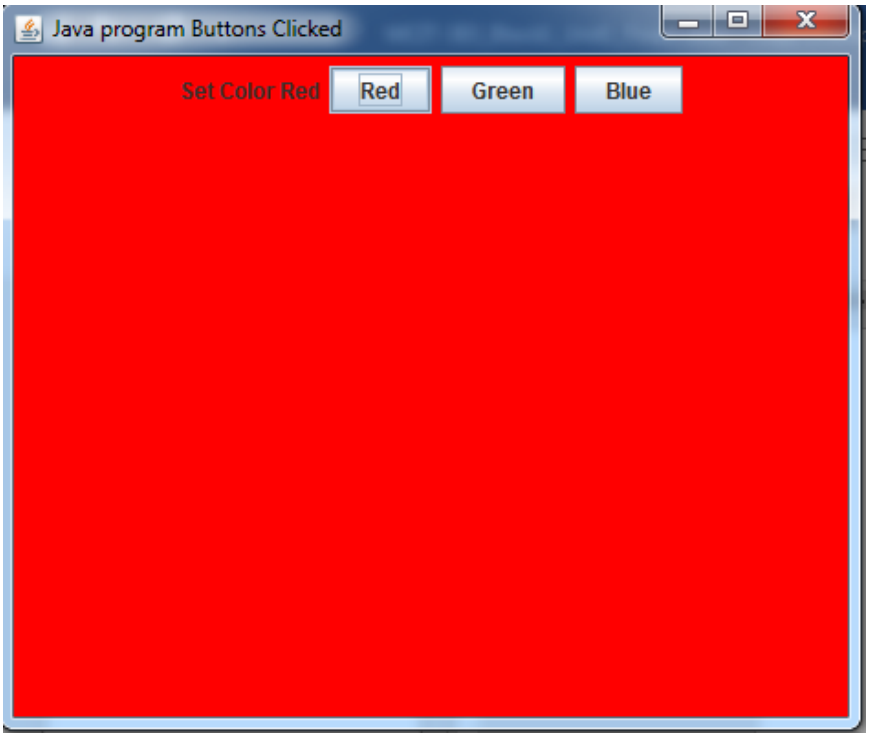


Figure 2.6 : Output of JButton class

2.6. JCheckBox, JRadioButton

2.6.1 JCheckBox

The JCheckBox 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 ".

An ItemEvent is generated when user selects or deselect a check box. If multiple checkbox put in your program then which checkbox is selected, to obtain a reference by calling getItem( ) method of ItemEvent class. The ItemEvent object passed to the itemStateChanged( ) method which is registered by ItemListener.

Constructors

JJCheckBox() : Creates an initially unselected check box button with no text, no icon.

JChechBox(String s) : Creates an initially unselected check box with text.

JCheckBox(String text, boolean selected) : Creates a check box with text and specifies whether or not it is initially selected.

The Methods of JCheckBox class are given in the below table 2.4.

Method Name	Description
protected String paramString()	It returns a string representation of this JCheckBox.
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.

Table 2.4: Methods of JCheckBox class.

```
// Program to create JCheckBox
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class CheckboxExample extends JFrame implements ItemListener
{
   JCheckBox c1,c2;
   JLabel l1;
   JPanel p1;
   CheckboxExample()
   {
       // Frame setting
      setLayout(new FlowLayout());
      setSize(400,700);
      setTitle("Java program for JCheckBox");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);
```

```
    // create checkbox
    c1=new JCheckBox("Apple");
    c2=new JCheckBox("Orange");

    // create JLabel
    l1=new JLabel();

    // create JPanel
    p1=new JPanel();
    p1.add(c1);
    p1.add(c2);
    p1.add(l1);
    add(p1);
    c1.addItemListener(this);
    c2. addItemListener(this);
    p1=new JPanel();

 }
public void itemStateChanged(ItemEvent e)
 {
    if (e.getSource() == c1)
    {
        if (e.getStateChange() == 1)
        {
            l1.setForeground(Color.red);
           l1.setText(c1.getText() + "is  selected");
```

```
            }
            else
            {

                l1.setForeground(Color.red);
               l1.setText(c1.getText()+ "is  not selected");
                }
        }
            else
            {
                if (e.getStateChange() == 1)
                l1.setText(c2.getText()+ "is  selected");
            else
                l1.setText(c2.getText()+ "is not  selected");
            }
        }
           public static void main(String args[])
          {
                    CheckboxExample cb=new CheckboxExample();
                    }
}
```

Output of the program is shown in Figure 2.7.

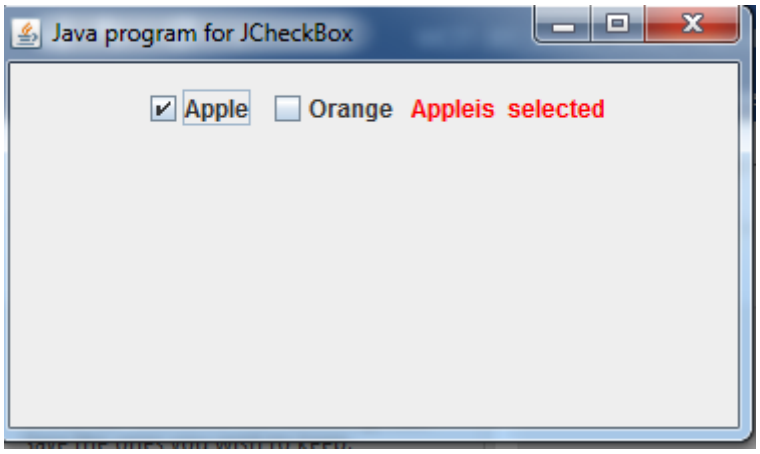


Figure 2.7 : Output of JCheckBox class

2.6.2 JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. Radio buttons must beconfigured into a group. Only one of the buttons in the group can be selected at any time.For example, if a user presses a radio button that is in a group, any previously selectedbutton in that group is automatically deselected.

Constructors

JRadioButton() : Creates an unselected radio button with no text.

JRadioButton(String s) : Creates an unselected radio button with specified text.

JRadioButton(String s, boolean selected) :Creates a radio button with the specified text and selected status.

The Methods of JRadioButton class are given in the below table 2.5.

Method Name	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.

void addActionListener(ActionListener a)	It i s used t o add t he act ion l istener t o t his object.

Table 2.5: Methods of JRadioButton class.

```
// Program to create JRadioButton
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JRadioButtonDemo extends JFrame implements ActionListener
{
   JRadioButton r1,r2,r3;
   JLabel l1;

   JRadioButtonDemo()
   {
      // Frame setting
      setLayout(new FlowLayout());
      setSize(400,400);
      setTitle("Java program for JRadioButton");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);
// Create a RadioButton
      r1=new JRadioButton("A");
      add(r1);
      r2=new JRadioButton("B");
      add(r2);
```

```
        r3=new JRadioButton("C");
        add(r3);
        ButtonGroup bg =new ButtonGroup();
        bg.add(r1);
        bg.add(r2);
        bg.add(r3);

        l1=new JLabel("select one");
        add(l1);

        r1.addActionListener(this);
        r2.addActionListener(this);
        r3.addActionListener(this);

    }
   public void actionPerformed(ActionEvent e)
    {
        l1.setText("you select" +e.getActionCommand());
    }
    public static void main(String args[])
    {
        JRadioButtonDemo rd=new JRadioButtonDemo();
    }

}
```

Output of the program is shown in Figure 2.8.

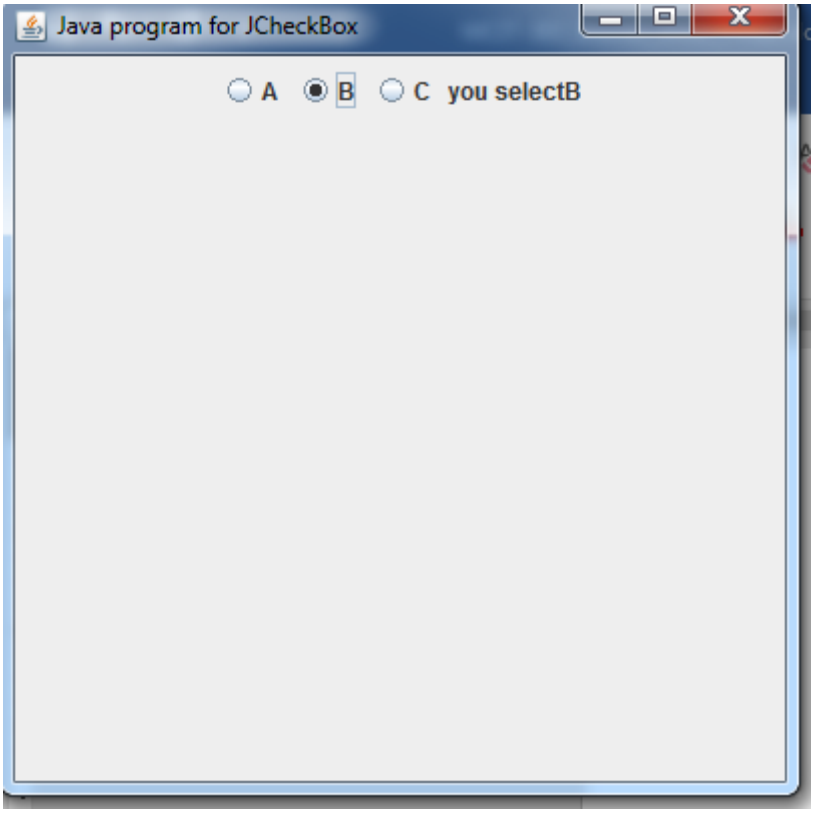


Figure 2.8 : Output of JRadioButton class

2.7. JList, JScrollPane, JComboBox

2.7.1 JList

JList is use for select one or more itemsfrom a list.JList class represents a list of text items.

Constructor

JList() :Creates a JList with an empty, read-only, model.

JList(ary[] listData) : Creates a JList that displays the elements in the specified array.

JList(ListModel<ary> dataModel) : Creates a JList that displays elements from the specified, non-null, model.

The Methods of JList class are given in the below table 2.6.

Method Name	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

Table 2.6: Methods of JList class.

A JList generates a ListSelectionEvent when the user select item from list.This event is also generated when the user de selects an item. It is handled by implementingListSelectionListener.

This listener specifies only one method, called valueChanged( ).

```
// Program to create Jist.
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
public class JListDemo extends JFrame implements ListSelectionListener
{
   JList list;
   JLabel l1;
```

```
String s[] = { "Apple", "Banana", "Orange", "Graps"};
JListDemo()
{
    // Frame setting
    setLayout(new FlowLayout());
    setSize(400,400);
    setTitle("Java program for JRadioButton");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);

    //create List
    list=new JList(s);
    add(list);

    l1=new JLabel("You select");
    add(l1);

    // Set the list selection mode to single selection.
    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
     list.addListSelectionListener(this);

}
public void valueChanged(ListSelectionEvent e)
{
    int idx =list.getSelectedIndex();
    // Display selection, if item was selected.
```

```
        if(idx != -1)
            l1.setText("Current selection: " + s[idx]);
        else
            l1.setText("Choose a Item");

    }

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

The Output of the program is shown in Figure- 2.9.

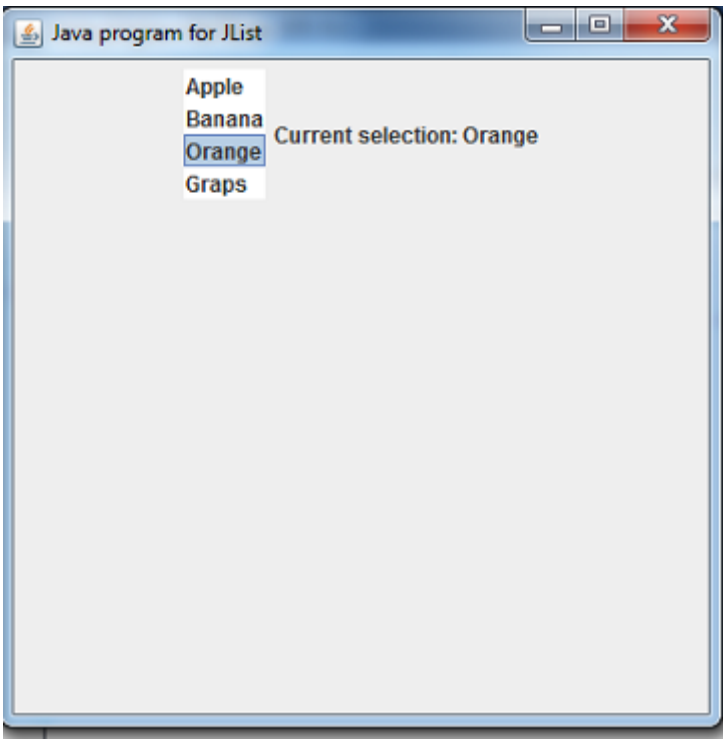


Figure 2.9 Output of JList Class

2.7.2 JScrollPane

A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scr oll pane to display a large component or a component whose size can change dynamically. If component size is larger than viewable area then automatically horizontal or Vertical scroll bar are set.

Constructors

JScrollPane() : It creates a scroll pane.

JScrollPane(Component) : It create a scroll pane on specified Component when you want to present.

JScrollPane(int, int) : sets the scroll pane's with two int parameters, when present, set the vertical and horizontal scroll bar respectively.

JScrollPane(Component, int, int) : Set scroll pane vertical or horizontal scroll bar on component .

```
// Program of JScroll pane
import javax.swing.*;
import java.awt.*;
public class JScrollPaneDemo extends JFrame
{
    JList list;
    JScrollPane js;
   String s[] = { "Apple", "Banana", "Orange", "Grapes","Watermelon","Peach",
                "Pear","Cherr","Strawberry","Nectarine","Blueberry","Pomegranate" };
   JScrollPaneDemo()
   {
      // Frame setting
      setLayout(new FlowLayout());
      setSize(400,400);
```

```
        setTitle("Java program for JScrollpane");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
 //create List
        list=new JList(s);
        add(list);
        // Add the list to a scroll pane.
        js=new JScrollPane(list);
       add(js);
    }
    public static void main(String args[])
    {
        JScrollPaneDemo js=new JScrollPaneDemo();
       }
}
// Output of JScrollPane is shown in Figure 2.10.
```

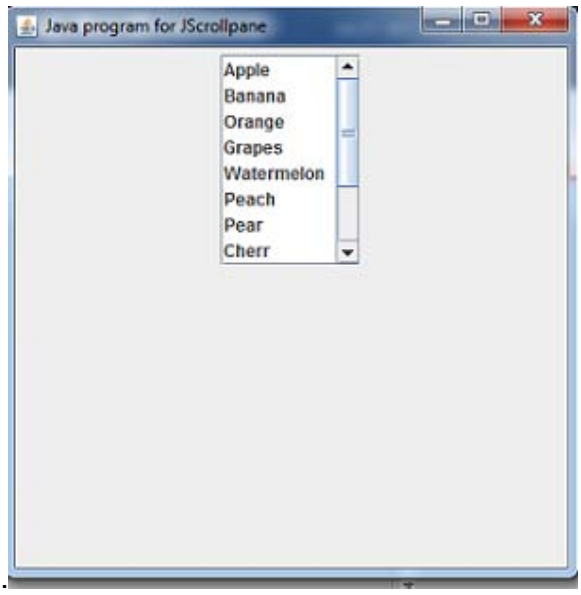


Figure 2.10 Output of JScroll Pane Class

2.7.3 JComboBox

JCombo box is a combination of a text field and a drop-down list. A combo box displays one entry and also display adrop-down list that allows a user to select a different item.

Constructors

JComboBox( ) : Creates a JComboBox with a default data model.

JComboBox(Object[] items) : Creates a JComboBox that contains the elements in the specified array.

JComboBox(Vector<?> items) : Creates a JComboBox that contains the elements in the specified Vector.

The Methods of JComboBox class are given in the below table 2.7.

Method Name	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.
void addItemListener(ItemListener i)	It is used to add the ItemListener.

Table 2.7: Methods of JComboBox class.

// Program to demonstrate the Combo Box and its items like india, japan and Canada. When user select country accordingly flag will display on icon base Label. All Flag images are stored into folder on which your program will save.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JComoBoxDemo extends JFrame implements ActionListener
{
    JLabel l1;
    ImageIcon india,japan,canada;
    JComboBox jcb;
    String flags[] = { "india", "japan", "canada"};
    JComoBoxDemo()
    {
        // Frame setting
        setLayout(new FlowLayout());
        setSize(400,400);
        setTitle("Java program for JComboBox");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);

        // Instantiate a combo box and add it to the content pane.
        jcb = new JComboBox(flags);
        add(jcb);
        // Create a label and add it to the content pane.
        l1= new JLabel(new ImageIcon("india.gif"));
        add(l1);
        jcb.addActionListener(this);
}
```

```
    public void actionPerformed(ActionEvent e)
    {
        String s = (String) jcb.getSelectedItem();
        l1.setIcon(new ImageIcon(s + ".gif"));
    }
    public static void main(String args[])
    {
        JComoBoxDemo jb=new JComoBoxDemo();
    }

}
```

Output of JComboBox is shown in Figure 2.11.

Figure 2.11 Output of JComboBox Class

2.8. EVENT HANDLING

Java Swing, like any other UI library, is an event-driven framework. When a user interactswith a GUI program (such as by clicking a button or pressing a key,Entering a character in Textbox, Clicking or Dragging a mouse,) a Java Swing programreceives an event that can initiate an appropriate reaction.

Event handling is at the core of successful swing programming. Events are supported by the java.awt.event package.

The modern approach to handling events is based on the delegation event model.

Components of Event Handling

Event handling has three main components,

Events : An event is a change in state of an object.

Events Source : Event source is an object that generates an event.

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

How Events are handled ?

A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return.

In the delegation event model, listeners must register with a source in order to receive an event notification.

public void addTypeListener(TypeListener el)

// For Example : addActionListener(this);

This provides an important benefit: notifications are sent only to listeners that want to receive them.

Main Event Class with Description:-

Class	Description

ActionEvent	Generated when a button is pressed, a list item is double clicked, or a menu item is selected.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
TextEvent	Generated when the value of a text area or text field is changed.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
InputEvent	Abstract super class for all component input event classes
KeyEvent	Generated when input is received from the keyboard
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse-enters or exits a component.
FocusEvent	Generated when a component gains or loses keyboard focus.
ContainerEvent	Generated when a component is added to or removed from a container.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 2.8 Event Class

Event Listener Interfaces: -

The delegation event model has two parts: sources and listeners.

Listeners are created by implementing one or more of the interfaces defined by the java.awt.event package.

When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

Event Listener Interfaces are in below Table 2.9.

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
TextListener	Defines one method to recognize when a text value changes.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 2.9 Event Listener Interface

The ActionListener Interface: -

This interface defines the actionPerformed( ) method that is invoked when an action event occurs.

The general forms of these method is

void actionPerformed(ActionEvent ae)

The AdjustmentListener Interface: -

This interface defines the adjustmentValueChanged( ) method that is invoked when anadjustment event occurs.

The general forms of these method is

void adjustmentValueChanged(AdjustmentEvent ae)

The ComponentListener Interface: -

This interface defines four methods that are invoked when a co mponent is resized, moved, shown, or hidden.

The general forms of these methods are

void componentResized(ComponentEvent ce)

void componentMoved(ComponentEvent ce)

void componentShown(ComponentEvent ce)

void componentHidden(ComponentEvent ce)

The ContainerListener Interface: -

This interface contains two methods.

When a component is added to a container, componentAdded( ) is invoked.

When a component is removed from a container, componentRemoved( ) is invoked

The general forms of these methods are

void componentAdded(ContainerEvent ce)

void componentRemoved(ContainerEvent ce)

The FocusListener Interface: -

This interface defines two methods

When a component obtains keyboard focus, focusGained( ) is invoked.

void focusGained(FocusEvent fe)

When a component loses keyboard focus, focusLost( ) is called.

void focusLost(FocusEvent fe)

The ItemListener Interface: -

This interface defines the itemStateChanged( ) method that is invoked when the state of an item changes.

The general forms of these method is

void itemStateChanged(ItemEvent ie)

The KeyListener Interface: -

This interface defines three methods.

The keyPressed( ) and keyReleased( ) methods are invoked when a key is pressed and released, respectively. The keyTyped( ) method is invoked when a character has been entered.

For example, if a user presses and releases the A key, three events are generated in sequence: key pressed, typed, and released.

The general forms of these methods are

void keyPressed(KeyEvent ke)

void keyReleased(KeyEvent ke)

void keyTyped(KeyEvent ke)

The MouseListener Interface: -

This interface defines five methods.

If the mouse is pressed and released at the same point, mouseClicked( ) is invoked.

When the mouse enters a component, the mouseEntered( ) method is called.

When it leaves, mouseExited( ) is called.

The mousePressed( ) and mouseReleased( ) methods are invoked when the mouse is pressed and released, respectively.

The general forms of these methods are:

void mouseClicked(MouseEvent me)

void mouseEntered(MouseEvent me)

void mouseExited(MouseEvent me)

void mousePressed(MouseEvent me)

void mouseReleased(MouseEvent me)

The MouseMotionListener Interface: -

This interface defines two methods.

The mouseDragged( ) method is called multiple times as the mouse is dragged.

The mouseMoved( ) method is called multiple times as the mouse is moved.

The general forms of these methods are

void mouseDragged(MouseEvent me)

void mouseMoved(MouseEvent me)

The TextListener Interface: -

This interface defines the textChanged( ) method that is invoked when a change occurs in a text area or text field.

The general forms of these method is

void textChanged(TextEvent te)

The WindowListener Interface: -

This interface defines seven methods.

The windowActivated( ) and windowDeactivated( ) methods are invoked when a window is activated or deactivated, respectively.

If a window is iconified, the windowIconified( ) method is called. When a window is deiconified, the windowDeiconified( ) method is called.

When a window is opened or closed, the windowOpened( ) or windowClosed() methods are called, respectively.

The windowClosing( ) method is called when a window is being closed.

The general forms of these methods are

```
void windowActivated(WindowEvent we)

void windowClosed(WindowEvent we)

void windowClosing(WindowEvent we)

void windowDeactivated(WindowEvent we)

void windowDeiconified(WindowEvent we)

void windowIconified(WindowEvent we)

void windowOpened(WindowEvent we)
```

2.9LET US SUM UP

- JApplet class is an extended version of java.applet.
- If you add more than one radio button to a container, you must add them to a button group. To do that, you add JRadioButton objects to a ButtonGroup object.
- The user can click on a JCheckBox to check or uncheck a box. Then, the code for the listener can change the processing that's done based on the setting for the check box.
- JCheckBox need to use a listener, you can use either the ActionListener or the ItemListener.
- JTextField is a lightweight component that allows the editing of a single line of text.
- AJPanel objects are containers to other GUI components can be attached. It is pain rectangular area.

- JFrame class has a title, display in the title bar at the top of the window. JFrame contains one or more menu.
- A JCheckBox is a graphical component that can be in either “on” (true) or “off” (false) state. When user clicking on a JCheckBox change its state from “on” to “off”, or from “off” to “on”.
- A JList component present the user with a scrolling list of text items. The list can be set up so that the user can choose either one item or multiple items.
- A JTextField object is a text component that allows for the editing of a single line of text.

2.10CHECK YOUR PROGRESS

1. Which object can be constructed to show any number of choices in the visible window?

a. JCheckBox	**b.** JList	**c.** JLabel	**d.** All of the abov

2. Which of these events is generated when a button is pressed?

   a. WindowEvent **b.** KeyEvent **c.** ActionEvent **d.** ItemEvent

3. Which of these packages contains all the classes and methods required for event handling in java?

   a. java.applet **b.** java.awt **c.** java.event **D.**java.awt.event

4. Which method executes only once ?

   a. start( ) **b.** init( ) **c.** paint( ) **D.**stop( )

5. Which class is used for this Processing Method processActionEvent( )?

a.Button, List,MenuItem	**b.** Button, Checkbox,Choice	**C.**Scrollbar, Component	**D.**None of the above

6. Which method can set or change the text in a Label?.

a. setText( )	**b.**getText( )	**c.** Both a and b	**d.**None of above

7. The swing related classes are contained in

a. javax.swing	**b.** javax.awt	**c.** javax.Swing	**d.**None of above

8. The ActionListener interface is not used for handling action events.

a. True **b.** False

9. The Following steps are required to perform

Implement the Listener interface and overrides its methods

Register the component with the Listener

a. Exception Handling	**b.**String Handling	**c.** Event Handling	**d.**None of the above

10. Which is the container that doesn't contain title bar and MenuBars but it can have other components like button, textfield etc?

a.Window	**b.** JFrame	**c.** JPanel	**d.**Container

11. Class JFrame directly extends class Container.

    a. True **b.** False

12. JApplets can contain menus.

    a. True **b.** False

13. A dedicated drawing area can be declared as a subclass of _____.

14. JTextFields directly extend class _____.

2.11CHECK YOUR PROGRESS: POSSIBLE ANSWERS

As per self-assessment questions asked in Self-Assessment exercises.

1. b.JList
2. c. ActionEvent
3. d. java.awt.event
4. b. init( )
5. a. Button,List,MenuItem
6. a. setText( )
7. a. javax.swing
8. b. False
9. c. Event Handling
10. c. JPanel
11. B. False Reason:JFrame inherits directly from Frame.
12. A.True
13. JPanel
14. JTextComponent

2.12 FURTHER READING

Many courses require students to read some extra material in addition to theirunits. Sometimes a text requires 'readings' which must be obtained by alllearners. Such texts are usually referred to as 'essential texts'. Someinstitutions call them 'set texts'.

On other occasions, students are expected toread widely from a variety of books, but the readings are entirely optional.

These boo ks are r eferred t o a s 'recommended t exts' or backg round r eading.The distinction is important, as books are usually difficult to obtain and theavailability and price of essential books must be checked before they arespecified as compulsory. A course that has no recommended textbooks isknown as a self-contained course.

Following are some examples:

Koul, B. N. and Ghaudhary, Sohanvir (1989). Self-instructional course units - IGNOU Handbook5. New Delhi: Indira Gandhi National Open University.

Thompson, Bruce (2003). Introduction to open learning and instructional design for openlearning. Vancouver: Commonwealth of Learning (COL).

2.13ASSIGNMENTS

1. What is difference between JFrame and JApplet?
2. Write a methods of JButton class ,JList class , JCheckBox class.
3. What is the use of ButtonGroup class?
4. Discuss delegation event model in details.
5. Method setEditable is a JTextComponent method. (Ture/False)
6. JPanel o bjects are containers t o w hich ot her G UI co mponents can be attached. (True/False)

2.14ACTIVITIES

1. Create ap plication t o t ake t wo values f rom t extbox and d o o peration l ike addition, su btraction, m ultiplication a nd d ivision. (take t hree Te xtbox a nd four Button)
2. Write a pr ogram t o take t wo l abel f or use rname and passw ord and t wo textfield and submit that details and display welcome message on Label.
3. Write a pr ogram t o t ake t wo Li st. W hen use r se lect i tem f rom one l ist it moves from second list and remove in the first list.

Unit 3: Swing Menu Component 3

Unit Structure

3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- To understand how to put JMenu on the JFrame.
- Working with JMenu, JMenuBar and JMenuItem.
- How to use JPopupMenu and also work with JPopupMenu.

3.2 INTRODUCTION

The previous chapter containsseveral components of Swing such as JTextField, JPasswordField, JButton, JCheckBox, JRadioButton, JList, JScrollPane, JComboBox. This Chapter presenting overview of swing JMenu Component. A menu bar can be linked to a top-level window. A menu bar shows a list ofMenu selection on the first level. Each selection is associated with a drop-down menu. This concept is implemented in AWT by the following classes: MenuBar, Menu and MenuItem.

3.3 JMenu, JMenuBar, JMenuItem

A Menu is a list of choices. A Menubar displays a list of top-level menu objects. In java, for implementing menu, a number of classes are use like JMenu, JMenuBar and JMenuItem.

A JMenuBar contains a number of object of JMenu and each of JMenu contains a number of object of JMenuItem.

To create a menu bar, first create an object of JMenuBar. This class only defines thedefault constructor. Next, create object of JMenu that will define the selectionsdisplayedon the bar.

Following are the constructors for JMenu:

- JMenu() : Create a new menu with an empty label.
- JMenu(String str) : Create a new menu with the specified label.
- JMenu(String str, boolean off) : Create a menu with the specified label and menu can be torn off.

After a JMenu object has been created then JMenuItem object can be added to JMenu.

Following are the constructors for JMenuItem :

- JMenuItem( ) : Create a new JMenuItem with empty label and no shortcut keyboard key.
- JMenuItem(String str) : Create a new JMenuItem with specified label and no shortcut keyboard key.
- JMenuItem(String str, MenuShortcut s) : Create a new JMenuItem with specified label and specified shortcut keyboard key.

To put JMenu object on the JMenuBar, so first create JMenuBar object. JMenuBar can be created with its default constructor like:

- JMenuBar( )

A JMenuBar is attached with the JFrame window using setMenuBar( ) method.

The Methods of JMenu class are given in the below table 3.1.

Method Name	Description
void setEnabled(Boolean b)	Sets whether or not this menu item can be chosen, it can be enabled or disabled.
boolean isEditable()	Check whether this menu item is enabled.
String getLabel()	Get the label for this menu item to specified label.
void setLabel(String str)	Sets the label for this menu item to the specified label.

Table 3.1: Methods of JMenu class.

//Program to create JMenu is show below.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JMenuDemo extends JFrame implements ActionListener
{
   JLabel l1;
```

```
JMenuBar mb;
JMenu m;
JMenuItem m1,m2,m3,m4;
JMenuDemo()
{
    setLayout(null);
    setSize(400,400);
    setTitle("Java program for Menu Bar");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
    //JMenubar created
    mb=new JMenuBar();
    m=new JMenu("File");
    m1=new JMenuItem("New");
    m2=new JMenuItem("Open");
    m3=new JMenuItem("Save");
    m4=new JMenuItem("Quit");

    m.add(m1);
    m.add(m2);
    m.add(m3);

    m.addSeparator();

    m.add(m4);
    //JMenu add into JMenuBar
```

```
    mb.add(m);
 // JMenuBar attached  to JFrame window
    setJMenuBar(mb);
    m1.addActionListener(this);
    m2.addActionListener(this);
     m3.addActionListener(this);
    m4.addActionListener(this);

    l1=new JLabel("You select");
    add(l1);
}

public void actionPerformed(ActionEvent e)
{
   if(e.getSource()==m1)
       l1.setText("New menu selected");
   else if(e.getSource()==m2)
       l1.setText("Open menu selected");
   else if(e.getSource()==m3)
       l1.setText("Save menu selected");
   else if(e.getSource()==m4)
       l1.setText("Quit menu selected");

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

```
        JMenuDemo md=new JMenuDemo();
    }

}
```

The Output of the program shown in Figure: 3.1

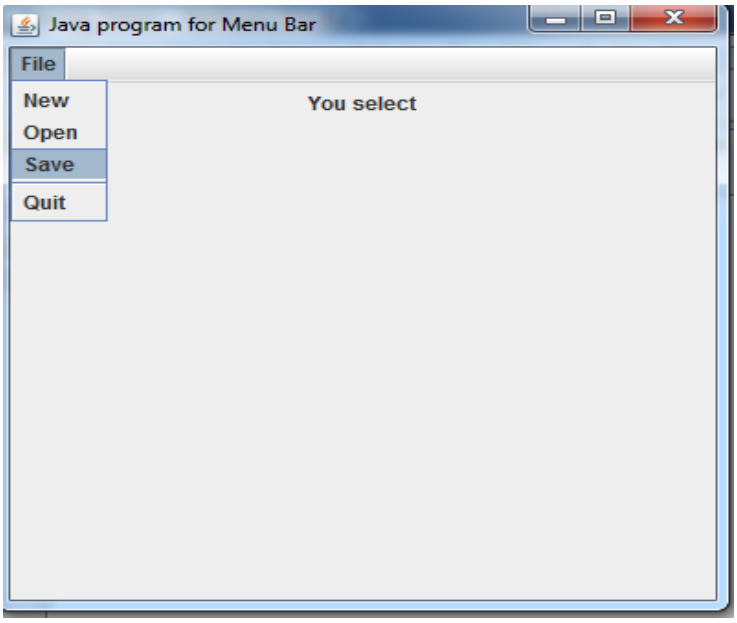


Figure-3.1 Output of JMenu.

3.4 JPopupMenu

A JPopupMenu is a menu which can be dynamically popped up at a specified position within a component. It is implemented by using JPopupMenu. The JPopupMenu is different than other components because JPopupMenu is not components and they are not usually visible. The JPopupMenu is call up by user when user performing some platform-dependent action with the mouse. For Example, User clicking with right mouse button, or clicking the mouse while holding down the control key.

The object of pop-up menu is belonging to the JPopupMenu class. A newly created JPopupMenu is empty. Items can be added to the JPopupMenu with its add(String str) method. User want to add separator line by using addSeparator( )method.

Following are the constructors for JMenuItem

- JPopupMenu() : Constructs a JPopupMenu without an "invoker".
- JPopupMenu(String label) : Constructs a JPopupMenu with the specified title.

The JPopupMenu generate an ActionEvent when user selects items from the menu. Mouse event have to be listened from the component. A MouseEvent object has a boolean value method, isPopupTrigger( ) can call when the user is trying to popup a menu. The JPopupMenu is popup either mousePressed or mouseReleased method. For example, the mousePressed method might look like below code.

```
public void  mousePressed(MouseEvent me)
{
        If(me.isPopupTrigger())
        {
                int x=me.getX();
                Int y=me.getY();
                pmenu.show(this,x,y);
        }
}
```

```
/* Program of JPopupMenu is display on JFrame and JPopupMenu contains item like red, green and blue */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JPopupMenuDemo extends JFrame implementsMouseListener
{
   JPopupMenu pm;
    JMenuItem m1,m2,m3;

  JPopupMenuDemo()
  {
     setLayout(new FlowLayout());
     setSize(400,400);
     setTitle("Java program for JPopupMenu");
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);

        //Create JPopup menu class
        pm=new JPopupMenu();
        m1=new JMenuItem("Red");
        m2=new JMenuItem("Green");
        m3=new JMenuItem("Blue");

         pm.add(m1);
         pm.add(m2);
         pm.add(m3);

         add(pm);
addMouseListener(this);

    }

public static void main(String args[])
    {
        JPopupMenuDemo jpm=new JPopupMenuDemo();
    }

    public void mouseClicked(MouseEvent e)    {     }
public void mouseEntered(MouseEvent e)   {    }
 public void mouseExited(MouseEvent e)     { }

    public void mousePressed(MouseEvent e)
    {
      if(e.isPopupTrigger())
      {
          int x=e.getX();
          int y=e.getY();
```

```
            pm.show(this, x, y);
        }
    }

     public void mouseReleased(MouseEvent e)
    {
      if(e.isPopupTrigger())
      {
          int x=e.getX();
          int y=e.getY();
          pm.show(this, x, y);
      }
    }
}
```

The Output of the program shown in Figure: 3.2

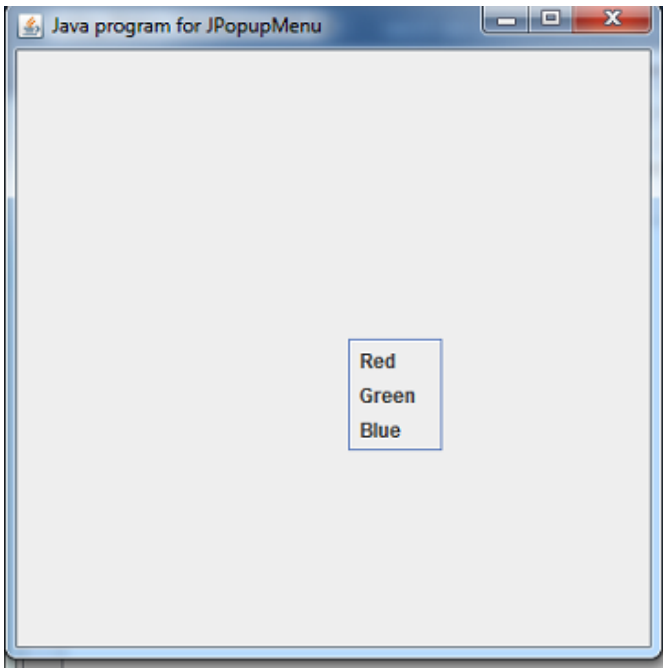


Figure-3.2 Output of JPopupMenu.

3.5LET US SUM UP

- A JMenu is a list of choice. A JMenuBar display a list of top-level menu choice.

- When user want to use JMenu it must be create a JFrame.
- Each JMenuItem is an instance of JMenuItem class attached to the JMenu.
- Shortcut keys to JMenu items can be added using the MenuShortcut class. The MenuShortcut class represents a keyboard accelerator for JMenuItem. JMenu shortcuts are created using virtual keycodes.
- JPopupMenu is a menu which can be dynamically popped up at a specified position within a component. It is implemented in java by class JPopupMenu.

3.6CHECK YOUR PROGRESS

1. A JMenuItem that is a JMenu is called _____.
2. _____Method attaches a JMenuBar to a JFrame.
3. Menus require a JMenuBar object so they can be attached to a JFrame.
   a. True **b.** False
4. Each JMenuItem is an instance of

   a. MenuShortcut class **b.** JPopupMenu class **c.** JMenuItem **D.**None of the above

3.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. submenu
2. setJMenuBar
3. a. true
4. c. JMenuItem

3.8 FURTHER READING

Many courses require students to read some extra material in addition to theirunits. Sometimes a text requires 'readings' which must be obtained by alllearners. Such texts are usually referred to as 'essential texts'. Someinstitutions call them 'set texts'.

On other occasions, students are expected toread widely from a variety of books, but the readings are entirely optional.

These boo ks are r eferred t o a s 'recommended t exts' or backg round r eading.The distinction is important, as books are usually difficult to obtain and theavailability and price of essential books must be checked before they arespecified as compulsory. A course that has no recommended textbooks isknown as a self-contained course.

Following are some examples:

Koul, B. N. and Ghaudhary, Sohanvir (1989). Self-instructional course units - IGNOU Handbook5. New Delhi: Indira Gandhi National Open University.

Thompson, Bruce (2003). Introduction to open learning and instructional design for openlearning. Vancouver: Commonwealth of Learning (COL).

3.9ASSIGNMENTS

1. JMenuBar is attached to the JFrame window using _____ method.
2. A separator line can be added with the _____ method.
3. Write a short note on JMenu.
4. Discuss about JPopupMenu class with example.

3.10ACTIVITIES

1. Create application to make two JMenu one for color and second for shape, color menu contains JMenuItem like red,green and blue. When user click on JMenuItem appropriate background color will change and Second JMenu is shape and its JMenuItem llike Rectangle , circle and oval,when user click on JMenuItem appropriate shape will draw on JFrame.
2. Create a JPopupMenu class, select its item and appropriate background color is change .

Unit 4: Swing Tree and Table Component

4

Unit Structure

4.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- To understand how to make JTable .
- To put data in row and column using JTable.
- Working with JTree.

4.2 INTRODUCTION

This Chapter presenting overview of swing JTree and JTable Component. The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component.JTableUI component enables you to present data in a grid withrows and columns.JTable wasdesigned according to the Model-View-Controller (MVC) design pattern.

4.3 JTree

JTree is a Swing component with which we can display hierarchical data. JTree is quite a complex component. A JTree has a 'root node' which is the top-most parent for all nodes in the tree. A node is an item in a tree. A node can have many children nodes. These children nodes themselves can have further children nodes. If a node doesn't have any children node, it is called a leaf node.The leaf node is displayed with a different visual indicator.it simply provides a view of the data.

A JTree object does not actually contain the data.it simply provides a view of the data. JTree displays its data vertically. Each row displayed by the tree contains exactly one item of data and that is called node.

Following are the constructors for JTree:

- JTree() : Creates a JTree with a sample model.
- JTree(Object[] value) : Creates a JTree with every element of the specified array as the child of a new root node.
- JTree(TreeNode root) : Creates a JTree with the specified TreeNode as its root, which displays the root node.

JTree depend on two models: TreeExpansionEvent, TreeModel and TreeSelectionModel. A JTree generates a variety of events: TreeSelectionEvent, and TreeModelEvent. TreeExpansionEvents occur when a node is expanded or collapsed. A TreeSelectionEvent is generated when the user selects or deselects a node within the tree. A TreeModelEvent is fired when the data or structure of the tree changes.

The listeners for these events are TreeExpansionListener, TreeSelectionListener,

and TreeModelListener, respectively.

The steps to follow to use a tree:

1. Create an instance of JTree.

2. Create a JScrollPane and specify the tree as the object to be scrolled.

3. Add the tree to the scroll pane.

4. Add the scroll pane to the content pane.

A DefaultMutableTreeNode object is created for the top node of the tree hierarchy. To add further tree nodes are then created by calling add( ) method to the tree.

```
//Program to create JTree is show below.
import javax.swing.*;
import java.awt.*;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.DefaultMutableTreeNode;
public class JTreeDemo extends JFrame implements TreeSelectionListener
{
   JTree tree;
   JLabel l1;
```

```
JTreeDemo()
{
    // Frame setting
   setLayout(new FlowLayout());
    setSize(400,400);
    setTitle("Java program for JTree");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);

    // Create top node of tree.
    DefaultMutableTreeNode root=new DefaultMutableTreeNode("Tree Demo");
    // Create subtree of "A".
    DefaultMutableTreeNode a1=new DefaultMutableTreeNode("A");
    root.add(a1);

    DefaultMutableTreeNode a2=new DefaultMutableTreeNode("A1");
    a1.add(a2);

    DefaultMutableTreeNode a3=new DefaultMutableTreeNode("A2");
    a1.add(a3);

    // Create subtree of "B".
    DefaultMutableTreeNode b1=new DefaultMutableTreeNode("B");
    root.add(b1);

    DefaultMutableTreeNode b2=new DefaultMutableTreeNode("B1");
```

```
        b1.add(b2);

        DefaultMutableTreeNode b3=new DefaultMutableTreeNode("B2");
        b1.add(b3);

        // Create the tree.
        tree =new JTree(root);
// Add the tree to a scroll pane.
        JScrollPane js=new JScrollPane(tree);
        add(js);

        l1=new JLabel("You select");
        add(l1);
           tree.addTreeSelectionListener(this);
       }
    public void valueChanged(TreeSelectionEvent e)
    {
      l1.setText("You select :"+e.getPath());

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

The Output of the program shown in Figure: 4.1

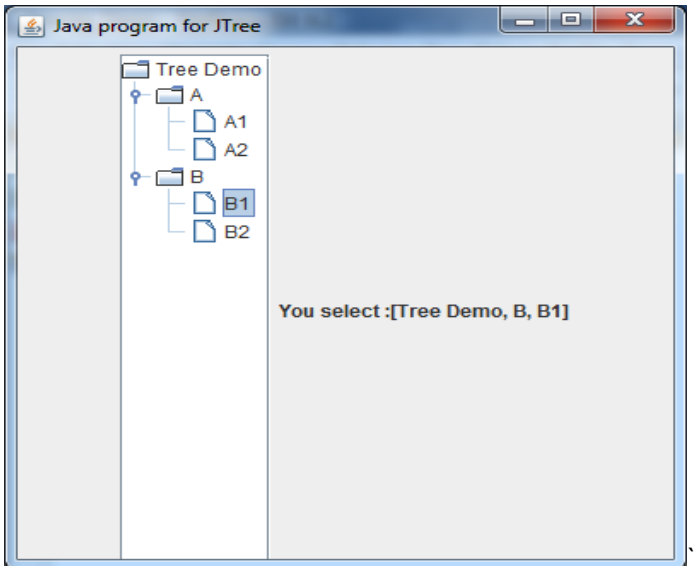


Figure-4.1 Output of JTree.

4.4 JTable

The Swing class JTable is a powerful UI component created for displaying tabular data like aspreadsheet. The data is represented as rows and columns.
Following are the constructors for JTable:

JTable() : Creates a JTable with empty cells.

JTable(int rows, int cols) : Create a JTable with rows and cols of empty cells.

JTable(Object[][] rows, Object[] columns) Creates a table with the specified data.

JTable have a three models. The first is the table model, which is defined by theTableModel interface. This model defines those things related to displaying data in a two-dimensional format. The second is the table column model, which is represented byTableColumnModel. JTable is defined in terms of columns, and it is TableColumnModel thatspecifies the characteristics of a column. The third model determines how items are selected, and it is specified by theListSelectionModel,

A JTable can generate several different events such as ListSelectionEvent and TableModelEvent. A ListSelectionEvent is generatedwhen the user selects

something in the table. By default, JTable allows you to select one ormore complete rows. A TableModelEvent is generated when that table’sdata changes in some way.

The steps to follow to use a JTable:

1. Create an instance of JTable.
2. Create a JScrollPane object, specifying the table as the object to scroll.
3. Add the table to the scroll pane.
4. Add the scroll pane to the content pane.

```
// Program to create table using JTable
import javax.swing.*;
import java.awt.*;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;

public class JTableDemo extends JFrame
{

   JTableDemo()
   {
        // Frame setting
      setLayout(new FlowLayout());
      setSize(700,400);
      setTitle("Java program for JTable");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);


      // Initialize column headings.
      String[] colhead={"NO","NAME"};
      // Initialize data.
      Object[][] rowdata={
                        {"101","Priya"},
```

```
                    {"102","Riya" },
                    {"103","Maan"},
                    {"104","Yashvi"},
                    {"105","Aarvi"}
                };
        // Create the table.
        JTable table = new JTable(rowdata, colhead);
        // Add the table to a scroll pane.
        JScrollPane jsp = new JScrollPane(table);
        // Add the scroll pane to the content pane.
        add(jsp);

    }
 public static void main(String args[])
 {
     JTableDemo jtd=new JTableDemo();
 }

}
```

The Output of the program shown in Figure: 4.2

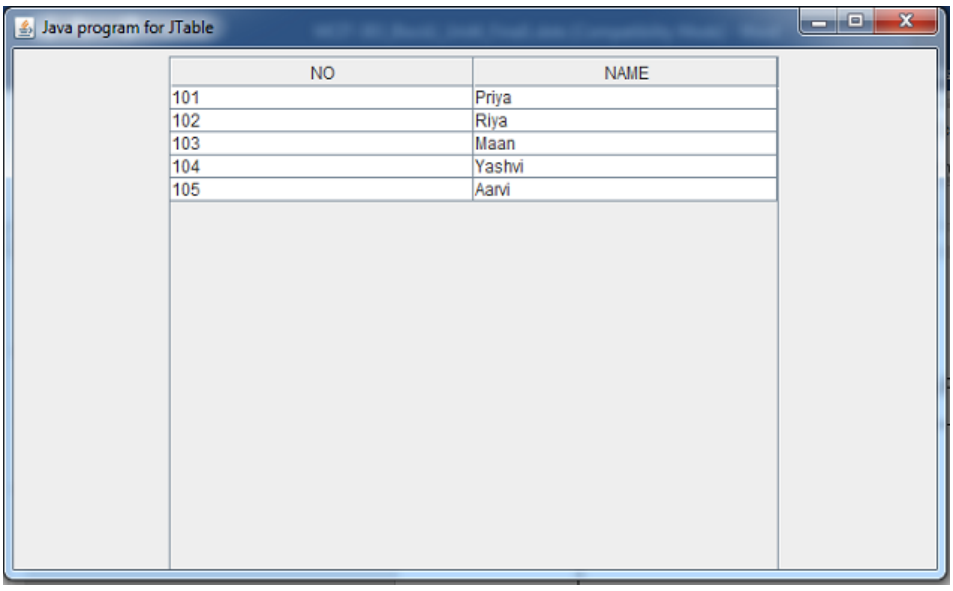


Figure-4.2 Output of JTable.

4.5LET US SUM UP

- JTree is a Swing component that represent hierarchical data.
- A JTree provides a view of the data.
- User can expand individual subtree.
- JTable represent data in rows and columns.
- JTable was designed according to the Model-View-Controller (MVC) design pattern, according to which components responsible for presentation (or the view) are separated from components that store data (or the model) for that presentation.

4.6CHECK YOUR PROGRESS

1. _____ component isrepresenting a hierarchical view of data.
2. JTree is packaged_____.
3. _____ method receives the TreeSelectionEvent.
4. _____component is displays rows and columns of data.

4.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. JTree
2. Javax.swing
3. valueChanged( )
4. JTable

4.8 FURTHER READING

Many courses require students to read some extra material in addition to theirunits. Sometimes a text requires 'readings' which must be obtained by alllearners. Such texts are usually referred to as 'essential texts'. Someinstitutions call them 'set texts'. On other occasions, students are expected toread widely from a variety of books, but the readings are entirely optional.

These boo ks are r eferred t o a s 'recommended t exts' or backg round r eading.The distinction is important, as books are usually difficult to obtain and theavailability and price of essential books must be checked before they arespecified as compulsory. A course that has no recommended textbooks isknown as a self-contained course.

Following are some examples:

Koul, B. N. and Ghaudhary, Sohanvir (1989). Self-instructional course units - IGNOU Handbook5. New Delhi: Indira Gandhi National Open University.

Thompson, Bruce (2003). Introduction to open learning and instructional design for openlearning. Vancouver: Commonwealth of Learning (COL).

4.9ASSIGNMENTS

1. List out event of JTree class.
2. Which model are used in JTree class.
3. Write a step to create JTree.
4. Which model are used in JTable class.
5. List out the events of JTable class.
6. Write a step to create JTable.

4.10ACTIVITIES

1. Write a program to create JTree with two subtrees like vegetable and fruit and add more children in vegetable and fruit tree.

Block-2

JDBC (Java Database Connectivity)

Unit 1: JDBC Introduction 1

Unit Structure

1.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- JDBC connectivity so student can perform CRUD operations in java.
- JDBC Drivers, Statements and ResultSet for data movement.
- The importance of JDBC to access database with Java application
- Architecture of JDBC

1.2 INTRODUCTION

The role of JDBC is very important. It enables Java applications and applets to connect to and access database. Lets take a look into the idea behind this.Different applications have to talk different databases, some standard way is required for this communication. In JDBC, the Java classes are available to provide access to any ANSI SQL-2 compliant database. This block covers the introduction and basics of JDBC. The next sections cover the JDBC driver and practical approaches for Database access.

1.3 JDBC BASICS

The JDBC (Java Database Connectivity) API defines interfaces and classes for writing database applications in Java by making database connections. Using JDBC you can send SQL, PL/SQL statements to almost any relational database. JDBC is a Java API for executing SQL statements and supports basic SQL functionality. It provides RDBMS access by allowing you to embed SQL inside Java code. Because Java can run on a thin client, applets embedded in Web pages can contain downloadable JDBC code to enable remote database access. You will learn how to create a table, insert values into it, query the table, retrieve results, and update the table with the help of a JDBC Program example.

Although JDBC was designed specifically to provide a Java interface to relational databases, you may find that you need to write Java code to access non-relational databases as well.

JDBC Architecture

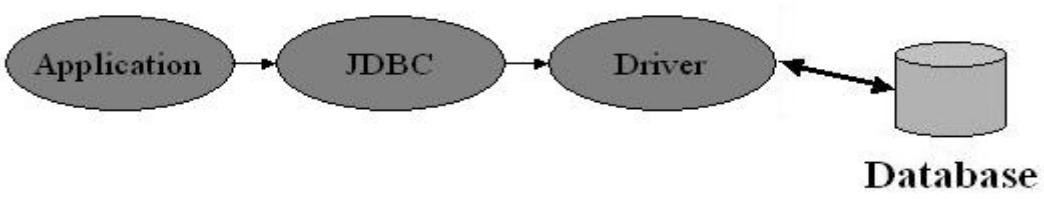
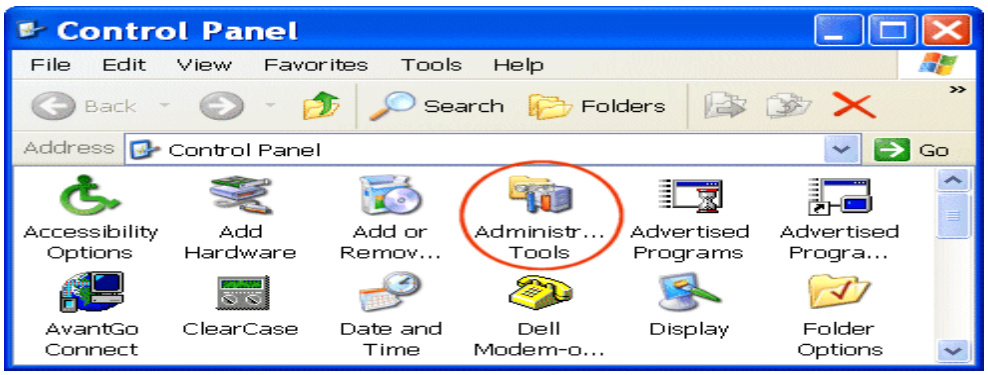


Figure 1: JDBC Architecture

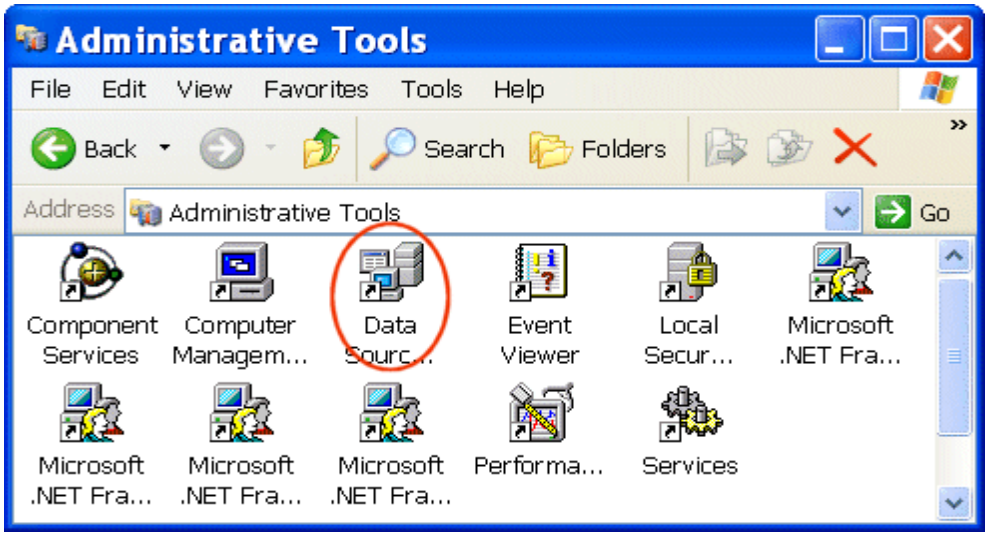
The above diagram represents JDBC architecture. The Java application which is intended to perform database operation needs to call JDBC library. JDBC library comprises Java packages *java.sql.* and javax.sql.** . Both these packages contain interfaces, classes, abstract classes and method to establish and maintain connection with database. Apart from these various methods to manage database transactions are available. JDBC loads a driver which talks to the database. Java application calls the JDBC library. JDBC loads a driver which talks to the database. We can change database engines without changing database code.

1.4Configuring ODBC Data Source

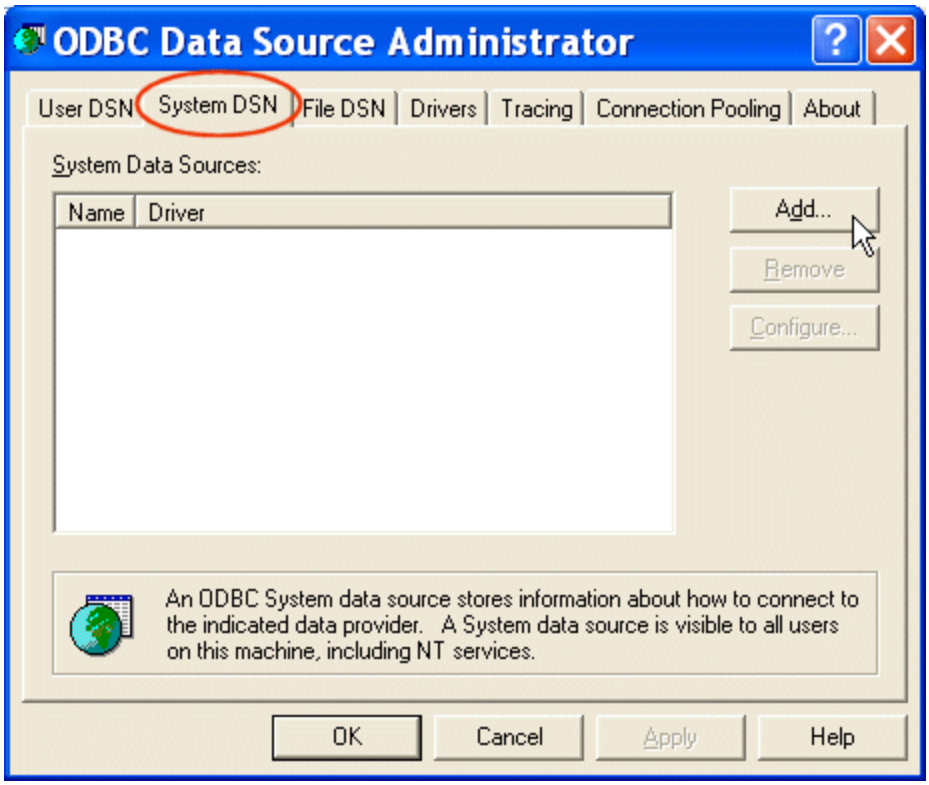
Click **Start > Settings > Control Panel** on the Windows menu. The **Control Panel** window appears.



Double-click **Administrative Tools** on the **Control Panel** window. The **Administrative Tools** window appears.

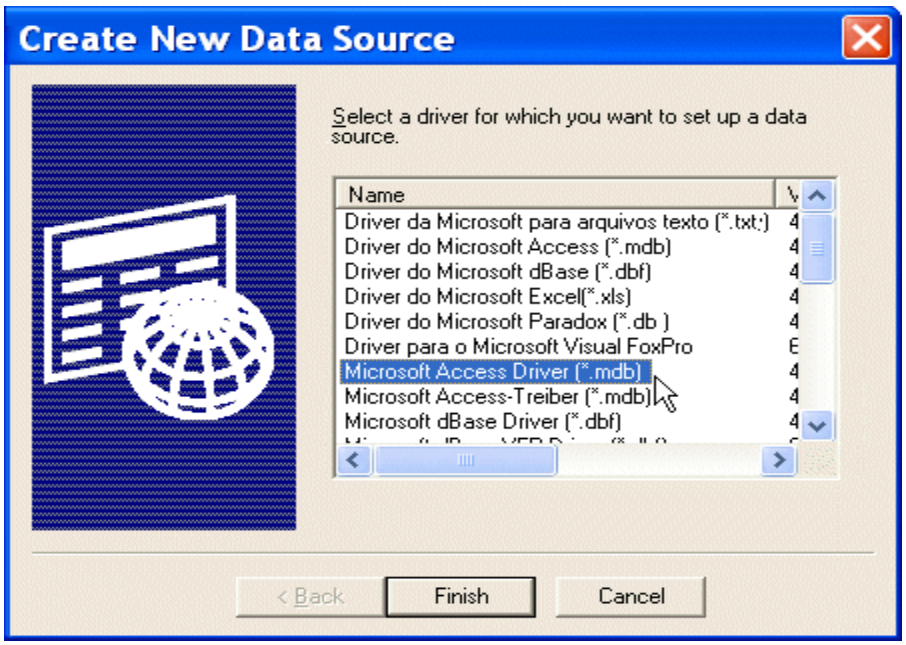


Double-click Data Sources (ODBC) on the Administrative Tools window. The ODBC Data Source Administrator window appears.

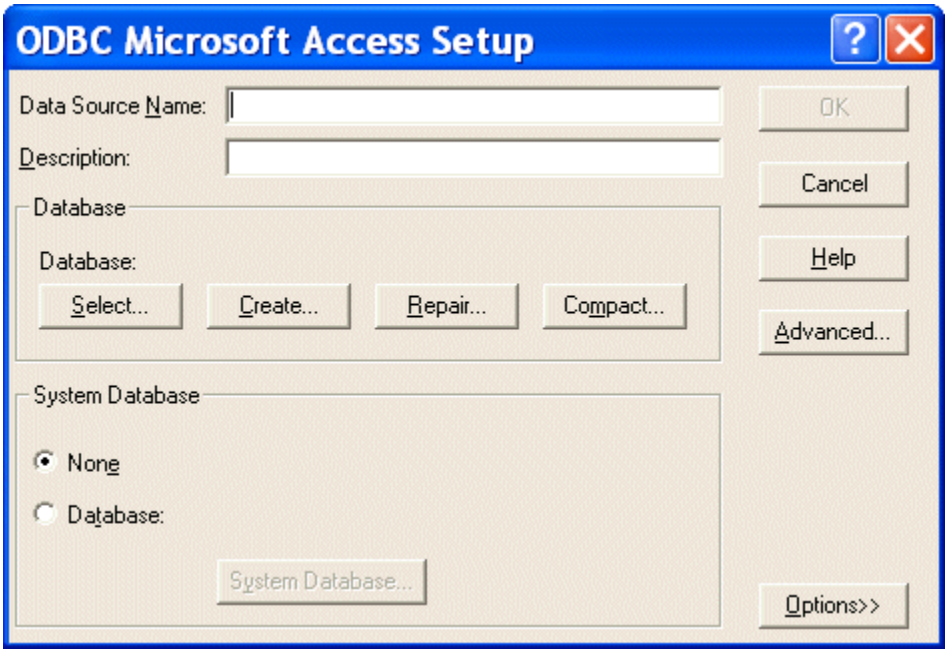


The **Create New Data Source** window appears.

Click the **System DSN** tab and click the **Add button**.

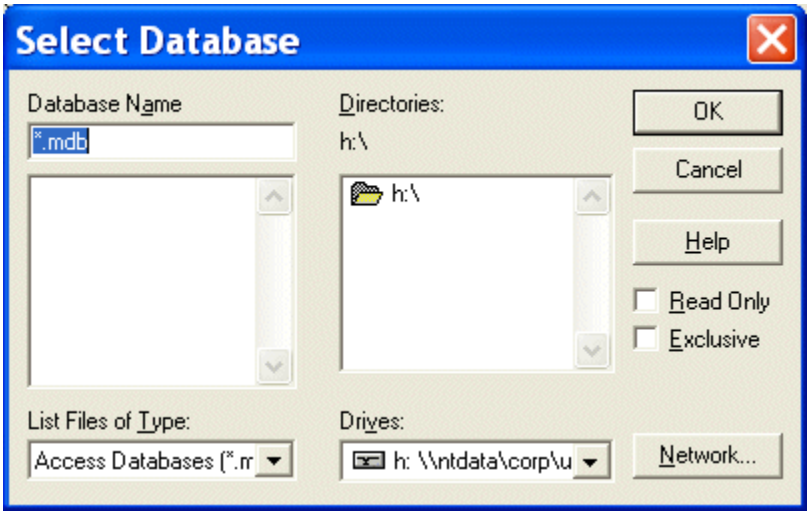


Scroll down the list until you find the driver associated with the database for which you want to create a data source, and then click **Finish**. The **ODBC Setup** dialog box appears.



Note that the information on this dialog box, including the dialog box title, varies based on the database driver you selected in the previous step. Here, we selected a Microsoft Access database driver, so the information displayed in the dialog box is specific to that database.

Enter a name in the **Data Source Name** field (for this example give the name **JdbcBasic**). Click the **Select** button in the **Database** group box. The **Select Database** dialog box appears.



Navigate until you find the database you want to use as the data source and click OK. You are returned to the ODBC Setup dialog box. Click OK on the ODBC Setup dialog box.

1.5 LET US SUM UP

This chapter focus on data base connectivity using JDBC introductory. Using this, student can learn the concept of JDBC and creating steps for ODBC object for DB connectivity.

1.6 CHECK YOUR PROGRESS

1. What are the steps involved in establishing a connection?
2. How can you load the drivers?
3. What Class.forName will do while loading drivers?
4. How can you make the connection?

1.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. This involves two steps (1)loading the driver and (2) making the connection.
2. Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBC Bridge driver, the following code will load it:

   Eg.

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

   Your driver documentation will give you the class name to use. For instance, if the class name is jdbc.DriverXYZ , you would load the driver with the following line of code:

   Eg.:Class.forName("jdbc.DriverXYZ");
3. It is used to create an instance of a driver and register it with the DriverManager. When you have loaded a driver, it is available for making a connection with a DBMS.
4. In establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea:

   Eg.

   String url = "jdbc:odbc:Fred";

   Connection con = DriverManager.getConnection(url, "Fernanda", "J8");

1.8 FURTHER READING

For more focus on JDBC read the book: Database Programming with JDBC and Java by George Reese.

1.9 ASSIGNMENTS

1. What is the use of JDBC?
2. Describe the JDBC Architecture in detail.

1.10 ACTIVITIES

- Try to create ODBC object for Microsoft Access Database which you have create for accessing data in java.

Unit 2: JDBC Queries 2

Unit Structure

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- JDBC Connection to DB
- JDBC Statements

2.2 INTRODUCTION

The JDBC connectivity must require the JDBC connectivity. You should enlist the driver in your program before you use it. Enlisting the driver is the procedure by which the Oracle driver's class record is stacked into the memory, so it tends to be used as a usage of the JDBC interfaces.

You have to do this enlistment just once in your program. You can enroll a driver in one of two different ways.

1. Class.forName()

   The most well-known way to deal with register a driver is to utilize Java's Class.forName() technique, to powerfully stack the driver's class document into memory, which naturally enlists it. This strategy is ideal since it enables you to make the driver enlistment configurable and convenient.

   Example:

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

2. DriverManager.registerDriver()

   The second way to register a driver, is to use the static DriverManager.registerDriver() method.

   You should use the registerDriver() method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

   Example:

   Driver myDriver = new sun.jdbc.odbc.JdbcOdbcDriver();

DriverManager.registerDriver(myDriver);

Database URL Formulation:

you ca n build up connection utilizing by the D riverManager.getConnection() technique. For sim ple r eference, l et m e l ist t he three ove r-burden DriverManager.getConnection() strategies −

getConnection(String url)

getConnection(String url, Properties prop)

getConnection(String url, String client, String secret key)

Here each st ructure requires a database URL. A da tabase U RL i s a l ocation t hat focuses to your database.

Detailing a database URL is the place the majority of the issues related with setting up an association happens.

Example

```
Connection cn=DriverManager.getConnection(String url);
```

When connection is acquired we ca n co operate w ith t he dat abase. Th e JDB C Statement, C allableStatement, a nd P reparedStatement i nterfaces characterize t he techniques and properties that empower you to send SQL or PL/SQL directions and get information from your database.

They additionally characterize techniques that assi stance co nnect i nformation t ype contrasts among Java and SQL information types utilized in a database.

Statement obj ect is used t o ex ecute a S QL st atement and cr eate st atement by the Connection object's createStatement( ) method.

Statement stmt= conn.createStatement( );

Methods

boolean execut e (String S QL): Returns a bool ean va lue of t rue i f a R esultSet object can be r etrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.

int execut eUpdate (String S QL): Returns the num ber of r ows affected by the execution of t he S QL st atement. U se t his m ethod t o e xecute SQL st atements for

which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.

ResultSet executeQuery (String SQL): Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

2.3 PREPARED STATEMENT

The PreparedStatement interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object.

This statement gives you the flexibility of supplying arguments dynamically.

String SQL = "Update stud SET pwd = ? WHERE id = ?";

stmt = conn.prepareStatement(SQL);

All parameters in JDBC are represented by the ? symbol, which is known as the parameter marker. You must supply values for every parameter before executing the SQL statement. The ? symbol represent values respectively.

The setXXX() methods bind values to the parameters, where XXX represents the Java data type of the value you wish to bind to the input parameter. If you forget to supply the values, you will receive an SQLException.

2.3 CALLABLE STATEMENT

After the connection is established,creates the CallableStatement object, which would be used to execute a call to a database stored procedure.

Syntax for Create Procedure in Database:

```
CREATE OR REPLACE PROCEDURE getStudName
  (STUD_ID IN NUMBER, STUD_FIRST OUT VARCHAR) AS
BEGIN
  SELECT first INTO STUD_FIRST
  FROM Employees
```

```
  WHERE ID = STUD_ID;
END;
```

The CallableStatement object can use the three types of parameters: IN, OUT, and INOUT.

IN: A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.

OUT: A parameter whose value is supplied by the SQL statement it returns. You retrieve values from theOUT parameters with the getXXX() methods.

INOUT: A parameter that provides both input and outtput values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

The Connection.prepareCall() method is used to instantiate a CallableStatement object based on the preceding stored procedure –

Syntax:

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getStudName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
```

2.5 LET US SUM UP

This chapter focus on the different types of statements supported by java. The usage of different statement and utilisation of it is discussed in this chapter.

2.6 CHECK YOUR PROGRESS

1. What is the use of PreparedStatement?
2. What is the use of CallableStatement?

2.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Refer 2.3
2. Refer 2.4

2.8 FURTHER READING

For more focus on JDBC read the book: Database Programming with JDBC and Java by George Reese

2.9 ASSIGNMENTS

- Demonstrate use of Statement, Prepared Statement and Callable Statement.

2.10 ACTIVITIES

- Try to create database and use different statement for data manipulation. Use the Procedure also.

Unit 3: Exception Handling in JDBC

3

Unit Structure

3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- JDBC SQLException methods
- The usage of Try...Catch... Finally Block

3.2 INTRODUCTION

Exception handling allows you to handle exceptional conditions such as program-defined errors in a controlled fashion.

When an exception condition occurs, an exception is thrown. The term thrown means that current program execution stops, and the control is redirected to the nearest applicable catch clause. If no applicable catch clause exists, then the program's execution ends.

JDBC Exception handling is very similar to the Java Exception handling but for JDBC, the most common exception you'll deal with is java.sql.SQLException.

3.3SQLEXCEPTION METHODS

An SQLException can occur both in the driver and the database. When such an exception occurs, an object of type SQLException will be passed to the catch clause. The passed SQLException object has the following methods available for retrieving additional information about the exception –

Method	Description
getErrorCode( )	Gets the error number associated with the exception.
getMessage( )	Gets the JDBC driver's error message for an error, handled by the driver or gets the Oracle error number and message for a database error.

getSQLState( )	Gets the X OPEN S QLstate s tring. For a JDBC driver error, no useful information is returned from this method. For a database error, the five-digit XOPEN SQLstate code is returned. This method can return null.
getNextException( )	Gets the next E xception obj ect i n t he exception chain.
printStackTrace( )	Prints the current exception, or throwable, and i t's backtrace t o a st andard er ror stream.
printStackTrace(PrintStream s)	Prints this throwable and its backtrace t o the print stream you specify.
printStackTrace(PrintWriter w)	Prints this throwable a nd i t's backtrace t o the print writer you specify.

3.4Try...Catch...Finally WITH EXAMPLE

By utilizing the information available from the Exception object, you can catch an exception and continue your program appropriately. Here is the general form of a try block –

```
try {
   // Your risky code goes between these curly braces!!!
}
catch(Exception ex) {
   // Your exception handling code goes between these
   // curly braces, similar to the exception clause
   // in a PL/SQL block.
}
```

```
finally {
   // Your must-always-be-executed code goes between these
   // curly braces. Like closing database connection.
}
```

3.5 LET US SUM UP

This chapter focus on the exception handling in JDBC. It focus on exception handling block.

3.6 CHECK YOUR PROGRESS

1. Write a short note on Exception Handling.
2. What is the use of Try...Catch...Finally block?

3.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Refer 3.3
2. Refer 3.4

3.8 FURTHER READING

For more focus on JDBC read the book: Database Programming with JDBC and Java by George Reese

3.9 ASSIGNMENTS

- Demonstrate use of try catch block in JDBC program.

3.10 ACTIVITIES

- Try to create database with also use the Exception handling.

Unit 4: JDBC Driver 4

Unit Structure

4.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- JDBC Drivers and which one is best
- Actual implementation of JDBC application

4.2 INTRODUCTION

JDBC is an API adds the database programming capabilities to Java.java.sql is referred to as JDBC API. JDBC drivers are used by Java applications applets to communicate with database servers.import java.sql.*; The star (*) indicates that all of the classes in the package java.sql are to be imported.

JDBC drivers are used by Java applications applets to communicate with database servers.It accepts the Java call and converts them into database's native language specific calls and vice versa.

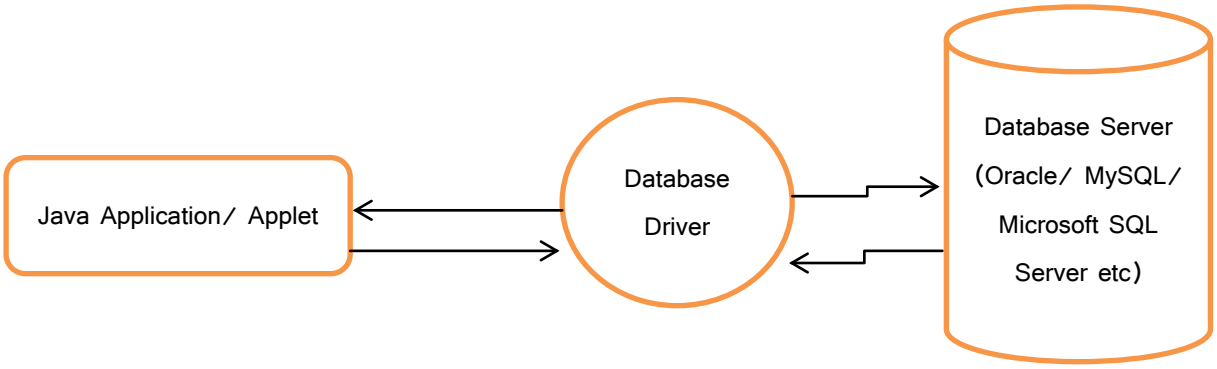


Figure 2: JDBC Driver

Database driver plays a vital role it accepts Java language specific calls from the Java application converts them into database's native language specific call which database engine can understand and converts database's native language specific responses into Java language specific response and delivers to Java application.

4.3JDBC Driver Types

JDBC drivers are classified into four categories.

- Type 1: JDBC-ODBC bridge driver: This is developed by Javasoft. It uses the functionalities of Microsoft's ODBC driver to communicate with database server. It is only as a temporary solution.
- Type 2: Native-API partly Java driver: These drivers use a server's native protocol that talks to database servers.
- Type 3: JDBC-Net pure Java Driver: These are pure Java drivers use standard protocol to communicate with database access server.
- Type 4: Native protocol pure Java driver: These are the pure Java driver uses vender specific protocol to communicate with database servers.

The type four drivers are using as a current industrial standard.

Type 1 JDBC-ODBC Bridge driver

JDBC-ODBC Bridge driver The Type 1 driver translates all JDBC calls into ODBC calls and sends them to the ODBC driver. ODBC is a generic API. The JDBC-ODBC Bridge driver is recommended only for experimental use or when no other alternative is available.

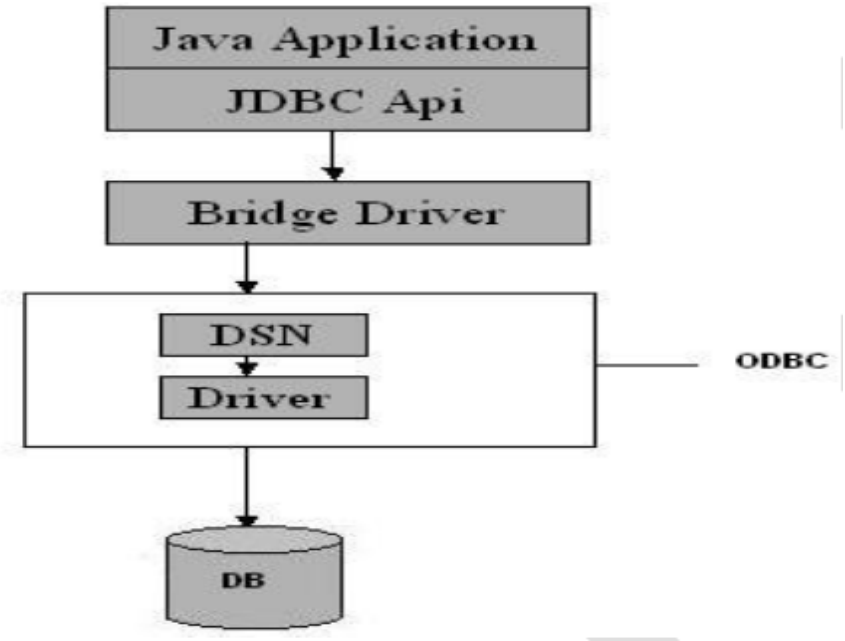


Figure 3: JDBC Type 1 Driver

- **Advantage**
    - The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.
- **Disadvantages**
    - Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable.
    - A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process.
    - They are the slowest of all driver types.
    - The client system requires the ODBC Installation to use the driver.
    - Not good for the Web.

Type 2Native-API/partly Java driver

The distinctive characteristic of type 2 jdbc drivers are that Type 2 drivers convert JDBC calls into database-specific calls i.e. this driver is specific to a particular database. Some distinctive characteristic of type 2 jdbc drivers are shown below. Example: Oracle will have oracle native api.

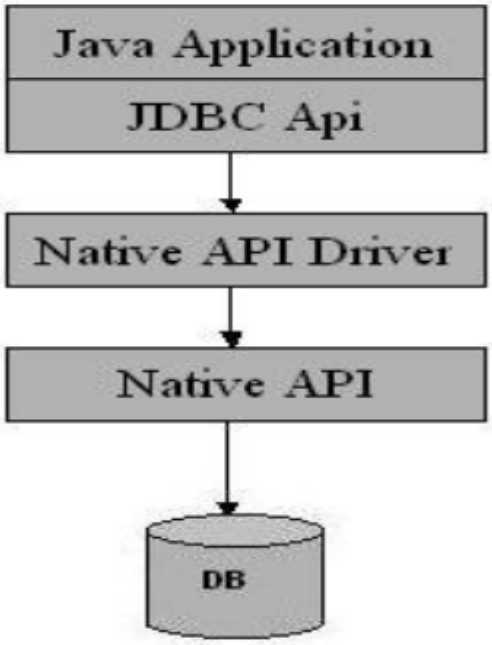


Figure 4: JDBC Type 2 Driver

- **Advantage**
    - The distinctive characteristic of type 2 jdbc drivers are that they are typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type 1 and also it uses Native api which is Database specific.
- **Disadvantages**
    - Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
    - Like Type 1 drivers, it's not written in Java Language which forms a portability issue.
    - If we change the Database we have to change the native api as it is specific to a database 4.
    - Mostly obsolete now
    - Usually not thread safe.

Type 3All Java/Net-protocol driver

Type 3 database requests are passed through the network to the middle-tier server. The middle-tier then translates the request to the database. If the middle-tier server can in turn use Type1, Type 2 or Type 4 drivers.

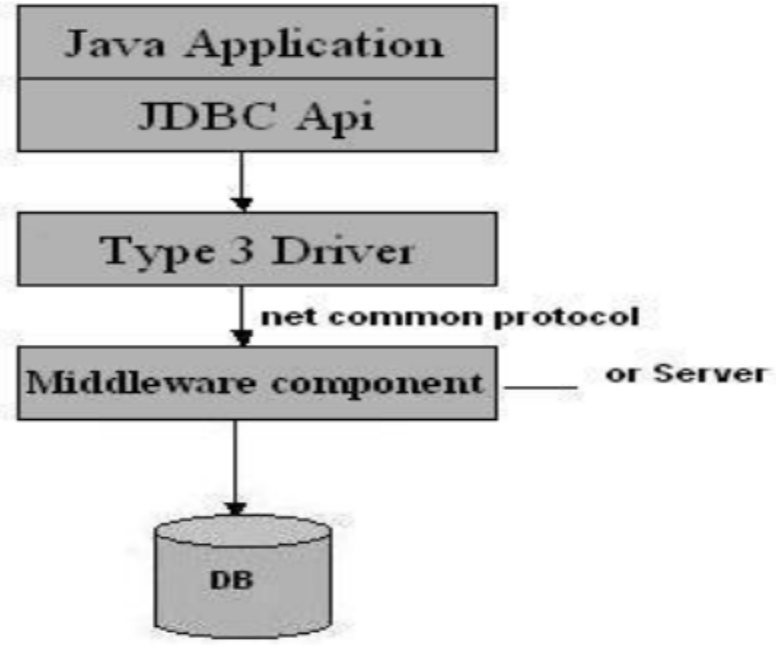


Figure 5: JDBC Type 3 Driver

- **Advantage**
    - This driver is server-based, so there is no need for any vendor database library to be present on client machines.
    - This driver is fully written in Java and hence Portable. It is suitable for the web.
    - There are many opportunities to optimize portability, performance, and scalability.
    - The net protocol can be designed to make the client JDBC driver very small and fast to load.
    - The type 3 driver typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration such as logging and auditing.
    - This driver is very flexible allows access to multiple databases using one driver.
    - They are the most efficient amongst all driver types.
- **Disadvantages**
    - It requires another server application to install and maintain. Traversing the recordset may take longer, since the data comes through the backend server.

Type 4Native-protocol/all-Java driver

The Type 4 uses java networking libraries to communicate directly with the database server.

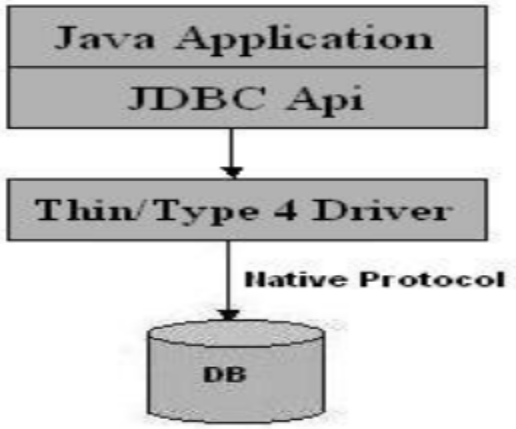


Figure 6: JDBC Type 4 Driver

- **Advantage**
    - The major benefit of using a type 4 jdbc drivers are that they are completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web.
    - Number of translation layers is very less i.e. type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good.
    - You don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically..
- **Disadvantages**
    - With type 4 drivers, the user needs a different driver for each database.

Loading a database driver

The jdbc connection process, we load the driver class by calling Class.forName() with the Driver class name as an argument. Once loaded, the Driver class creates an instance of itself. A client can connect to Database Server through JDBC Driver. Since most of the Database servers support ODBC driver therefore JDBC-ODBC Bridge driver is commonly used.

The return type of the Class.forName (String ClassName) method is "Class". Class is a class in

java.lang package.

```
try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //Or any other driver
}
catch(Exception x){
        System.out.println( "Unable to load the driver class!" );
}
```

Creating a oracle jdbc Connection

The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManager is considered the backbone of JDBC architecture. DriverManager class manages the JDBC drivers that are installed on the system. Its getConnection() method is used to establish a connection to a database. It uses a username, password, and a jdbc url to establish a connection to the database and returns a connection object. A jdbc Connection represents a session/connection with a specific database. Within the context of a Connection, SQL, PL/SQL statements are executed and results are returned. An application can have one or more connections with a single database, or it can have many connections with different databases. A Connection object provides metadata i.e. information about the database, tables, and fields. It also contains methods to deal with transactions.

JDBC URL Syntax:: jdbc: <subprotocol>: <subname>

JDBC URL Example:: jdbc: < subprotocol>: < subname>•Each driver has its own subprotocol

•Each subprotocol has its own syntax for the source. We're using the jdbc odbc subprotocol, so the DriverManager knows to use the sun.jdbc.odbc.JdbcOdbcDriver.

```
try{
 Connection
dbConnection=DriverManager.getConnection(url,"loginName","Password")
}
catch( SQLException x ){
        System.out.println( "Couldn't get connection!" );
}
```

4.4JDBC RESULTSETS & STATEMENTS

Once a connection is obtained we can interact with the database. Connection interface defines methods for interacting with the database via the established

connection. To execute SQL statements, you need to instantiate a Statement object from your connection object by using the createStatement() method.

Statement statement = dbConnection.createStatement();

A statement object is used to send and execute SQL statements to a database.

Three kinds of Statements

- Statement: Execute simple sql queries without parameters.

  Statement createStatement()

  Creates an SQL Statement object.

- Prepared Statement: Execute precompiled sql queries with or without parameters.
  PreparedStatement prepareStatement(String sql)
  returns a new PreparedStatement object. PreparedStatement objects are precompiledSQL statements.
- Callable Statement: Execute a call to a database stored procedure.
  CallableStatement prepareCall(String sql)
  returns a new CallableStatement object. CallableStatement objects are SQL stored procedure call statements.

ResultSet

Statement interface defines methods that are used to interact with database via the execution of SQL statements. The Statement class has three methods for executing statements:

executeQuery(), executeUpdate(), and execute(). For a SELECT statement, the method to use is executeQuery . For statements that create or modify tables, the method to use is executeUpdate. Note: Statements that create a table, alter a table, or drop a table are all examples of DDL

statements and are executed with the method executeUpdate. execute() executes an SQLstatement that is written as String object.

Creating a ResultSet

You create a ResultSet by executing a Statement or PreparedStatement, like this: Statement st atement = co nnection.createStatement(); R esultSet r esult = statement.executeQuery("select * from peopl e"); O r like this: S tring sq l = "select * from people"; PreparedStatement statement = connection.prepareStatement(sql);

ResultSet provides access to a table of data generated by executing a Statement. The table rows are retrieved in sequence. A ResultSet maintains a cursor pointing to its current row of data. The n ext() method is used to successively step through the rows of the tabular results.

ResultSetMetaData I nterface holds information on t he types and properties of the columns in a ResultSet. It is constructed from the Connection object.

ResultSet Types

A ResultSet can be of a certain type. The type determines some characteristics and abilities of the R esultSet. N ot all types are supported by all databases and JDBC drivers. You will have to check your database and JDBC driver to see if it supports the type you want to use. The DatabaseMetaData.supportsResultSetType(int type) method returns true or false depending on w hether the given type is supported or not. The D atabaseMetaData class is covered in a l ater text. At the time of writing there are three ResultSet types:

1. ResultSet.TYPE_FORWARD_ONLY

2. ResultSet.TYPE_SCROLL_INSENSITIVE

3. ResultSet.TYPE_SCROLL_SENSITIVE

The default type is TYPE_FORWARD_ONLY TYPE_FORWARD_ONLY means that the ResultSet can only be navigated forward. That is, you can only move from row 1, to row 2, to row 3 etc. You cannot move backwards in the ResultSet.

TYPE_SCROLL_INSENSITIVE m eans that t he R esultSet ca n be n avigated (scrolled) b oth forward and bac kwards. You can also jump to a posit ion relative to the current position, or jump to an absolute position. The ResultSet is insensitive to changes in theunderlying data source while the ResultSet is open. That is, if a record in the ResultSet is changed in the database by another thread or process, it will not

be reflected in already opened ResulsSet's of this type. TYPE_SCROLL_SENSITIVE means that the ResultSet can be navigated (scrolled) both forward and backwards. You can also jump to a position relative to the current position, or jump to an absolute position. The ResultSet is sensitive to changes in the underlying data source while the ResultSet is open. That is, if a record in the ResultSet is changed in the database by another thread or process, it will be reflected in already opened ResulsSet's of this type.

4.5JDBC APPLICATION

Create Microsoft Access Database and table. Then create odbc object for connectivity purpose. The code always resides between try..catch block.

Example 1: Display database metadata

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection connection=DriverManager.getConnection("jdbc:odbc:JdbcBasic");
  DatabaseMetaData meta=connection.getMetaData();
  System.out.print("Database: "+meta.getDatabaseProductName());
  System.out.println(" version "+meta.getDatabaseProductVersion());
  System.out.println("User name: "+meta.getUserName());
  System.out.println("Driver name:"+ meta.getDriverName());
  System.out.println("URL:"+meta.getURL());
```

Example 2: Display database data

```
  Statement st=connection.createStatement();
  ResultSet rs=st.executeQuery("select * from temp");
  ResultSetMetaData rsm=rs.getMetaData();
   int j=1;
  int i=1;
```

```
  int cocount=rsm.getColumnCount();
  while(j<=cocount)
  {
    System.out.println(rsm.getColumnName(j));
    j++;
  }
  while(rs.next())
  {
        System.out.println(rsm.getColumnName(2)+":"+rs.getString(2));
        //i++;
   }
```

Example 3: CRUD Operations into mysql database

Insert Data:

```
Class.forName("com.mysql.jdbc.Driver");
          cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/baou", "r  oot",
"root");
           String sql= "insert into msc2 values(25,'Pray','Mehsana')";
           Statement st=cn.createStatement();
           st.executeUpdate(sql);
           cn.close();
  connection.close();
```

Delete Data:

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

```
st.executeUpdate("delete from msc2");
```

Display Data:

```
Connection
cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/baou","root", "root");
    Statement st=cn.createStatement();
ResultSet rs=st.executeQuery("select * from msc2");
  while (rs.next())
  {
     System.out.println(rs.getString(2));
     int a= Integer.parseInt(rs.getString(1));
     System.out.println(a);
  }
     cn.close();
```

Update Data:

```
Connection
cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/baou","root","root");
     Statement st=cn.createStatement();
     st.executeUpdate("update msc2 set name=abc where id=25");
```

4.6 LET US SUM UP

This chapter focus on data base connectivity using JDBC.

4.7 CHECK YOUR PROGRESS

1. How can you create JDBC statements?
2. How can you retrieve data from the ResultSet?
3. What are the different types of Statements?
4. How can you use PreparedStatement?
5. What setAutoCommit does?
6. How to call a Strored Procedure from JDBC?
7. How to Retrieve Warnings?
8. How can you Move the Cursor in Scrollable Result Sets?
9. What's the difference between TYPE_SCROLL_INSENSITIVE, and TYPE_SCROLL_SENSITIVE?
10. How to Make Updates to Updatable Result Sets?

4.8 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Create JDBC statements:

   A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it, supplying the appropriate execute method with the SQL statement you want to send. For a SELECT statement, the method to use is executeQuery. For statements that create or modify tables, the method to use is executeUpdate.

   Eg.

   It takes an instance of an active connection to create a Statement object. In the following example, we use our Connection object con to create the Statement object stmt :

   Statement stmt = con.createStatement();

2. Retrieve data from the ResultSet:

   Step 1.

   JDBC returns results in a ResultSet object, so we need to declare an instance of the class ResultSet to hold our results. The following code demonstrates declaring the ResultSet object rs.

Eg.

ResultSet rs = st mt.executeQuery("SELECT C OF_NAME, P RICE F ROM COFFEES");

Step2.

String s = rs.getString("COF_NAME");

The method getString is invoked on the ResultSet object rs , so getString will retrieve (get) the value stored in the column COF_NAME in the current row of rs.

3. Types of Statements:
   1.Statement (use createStatement method)
   2. Prepared Statement (Use prepareStatement method) and
   3. Callable Statement (Use prepareCall)

4. Use PreparedStatement:
   This special type of statement is derived from the more general class, Statement.If you want to execute a Statement object many times, it will normally reduce execution time to use a PreparedStatement object instead.The advantage to this is that in most cases, this SQL statement will be sent to the DBMS right away, where it will be compiled. As a result, the PreparedStatement object contains not just an SQL statement, but an SQL statement that has been precompiled. This means that when the PreparedStatement is executed, the DBMS can just run the PreparedStatement 's SQL statement without having to compile it first.
   Eg.
   PreparedStatement updatSales = con.prepareStatement("UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");

5. When a connection is created, it is in auto-commit mode. This means that each individual SQL statement is treated as a transaction and will be automatically committed right after it is executed. The way to allow two or more statements to be grouped into a transaction is to disable auto-commit mode

Eg.

```
con.setAutoCommit(false);
```

Once auto-commit mode is disabled, no SQL statements will be committed until you call the method commit explicitly.

Eg.

```
con.setAutoCommit(false);
PreparedStatement updateSales = con.prepareStatement(
"UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
updateSales.setInt(1, 50);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate();
PreparedStatement updat eTotal = co n.prepareStatement("UPDATE COFFEES SET TOTAL = TOTAL + ? WHERE COF_NAME LIKE ?");
updateTotal.setInt(1, 50);
updateTotal.setString(2, "Colombian");
updateTotal.executeUpdate();
con.commit();
con.setAutoCommit(true);
```

6. Call a Strored Procedure from JDBC:

   The first step is to create a CallableStatement object. As with Statement an and PreparedStatement objects, this is done with an open Connection object. A CallableStatement object contains a call to a stored procedure;

   Eg.

   ```
   CallableStatement cs = con.prepareCall("{call SHOW_SUPPLIERS}");
   ResultSet rs = cs.executeQuery();
   ```

7. Retrieve Warnings:

   SQLWarning objects are a subclass of SQLException that deal with database access warnings. Warnings do not stop the execution of an application, as exceptions do; they simply alert the user that something did not happen as

planned.A warning ca n be r eported on a Connection obj ect, a S tatement object (including PreparedStatement and C allableStatement obj ects), or a ResultSet object. E ach of t hese classe s has a get Warnings method, w hich you must invoke in order to see the first warning reported on the calling object Eg.

```
SQLWarning warning = stmt.getWarnings();
if (warning != null) {
System.out.println("\n---Warning---\n");
while (warning != null) {
System.out.println("Message: " + warning.getMessage());
System.out.println("SQLState: " + warning.getSQLState());
System.out.print("Vendor error code: ");
System.out.println(warning.getErrorCode());
System.out.println("");
warning = warning.getNextWarning();
}
}
```

8. Move the Cursor in Scrollable Result Sets? :

   One of t he new features in t he JDBC 2.0 API i s the ability to move a r esult set's cursor backward as well as forward. There are also methods that let you move the cursor to a particular row and check the position of the cursor.

   Eg.

```
Statement st                                    mt =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet  srs  = st mt.executeQuery("SELECT C OF_NAME, P RICE F ROM
COFFEES");
```

   The first ar gument i s one of t hree co nstants added t o t he R esultSet A PI t o indicate t he t ype of a R esultSet o bject: TY PE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE , and TYPE_SCROLL_SENSITIVE .

   The se cond ar gument i s one of two R esultSet co nstants for sp ecifying whether a r esult se t i s read-only or updat able: C ONCUR_READ_ONLY an d CONCUR_UPDATABLE . The point to remember here is that if you specify a

type, you must also specify whether it is read-only or updatable. Also, you must specify the type first, and because both parameters are of type int , the compiler will not complain if you switch the order.

Specifying the constant TYPE_FORWARD_ONLY creates a nonscrollable result set, that is, one in which the cursor moves only forward. If you do not specify any constants for the type and updatability of a ResultSet object, you will automatically get one that is TYPE_FORWARD_ONLY and CONCUR_READ_ONLY

9. TYPE_SCROLL_INSENSITIVE v/sTYPE_SCROLL_SENSITIVE.

   You will get a scrollable ResultSet object if you specify one of these ResultSet constants.The difference between the two has to do with whether a result set reflects changes that are made to it while it is open and whether certain methods can be called to detect these changes. Generally speaking, a result set that is TYPE_SCROLL_INSENSITIVE does not reflect changes made while it is still open and one that is TYPE_SCROLL_SENSITIVE does. All three types of result sets will make changes visible if they are closed and then reopened.

   Eg.

```
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet srs = stmt.executeQuery("SELECT COF_NAME, PRICE FROM
COFFEES");
srs.afterLast();
while (srs.previous()) {
String name = srs.getString("COF_NAME");
float price = srs.getFloat("PRICE");
System.out.println(name + " " + price);
}
```

10. Make Updates to Updatable Result Sets:

    Another new feature in the JDBC 2.0 API is the ability to update rows in a result set using methods in the Java programming language rather than having to send an SQL command. But before you can take advantage of this capability, you need to create a ResultSet object that is updatable. In order to do this, you supply the ResultSet constant CONCUR_UPDATABLE to the createStatement method.

    Eg.

    ```
    Connection con = DriverManager.getConnection("jdbc:mySubprotocol:mySubName");
    Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
    ResultSet uprs = stmt.executeQuery("SELECT COF_NAME, PRICE FROM COFFEES");
    ```

4.9 FURTHER READING

- Refer Tutorial Point

4.10 ASSIGNMENTS

- Create an android application for CRUD operations in JAVA

Block-3

Java Network Programming

Unit 1: Networking Basics & Socket Programming

1

Unit Structure

1.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Learn Java networking concepts
- Java client server communication
- Socket Programming

1.2 INTRODUCTION

The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols −

- **TCP** − TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP** − UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

1.3 SOCKET PROGRAMMING

Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket. The following are the constructors.

public Socket(String host, int port) throws UnknownHostException, IOException

public Socket(InetAddress host, int port) throws IOException

public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException

public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException

public Socket()

Important methods

Sr.No.	Method & Description
1	**public void connect (SocketAddress host, int timeout) throws IOException** This method connects the socket to the specified host. This method is needed only when you instantiate the Socket using the no-argument constructor.
2	**public InetAddress getInetAddress()** This method returns the address of the other computer that this socket is connected to.

3	**public int getPort()** Returns the port the socket is bound to on the remote machine.
4	**public int getLocalPort()** Returns the port the socket is bound to on the local machine.
5	**public SocketAddress getRemoteSocketAddress()** Returns the address of the remote socket.
6	**public InputStream getInputStream() throws IOException** Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
7	**public OutputStream getOutputStream() throws IOException** Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.
8	**public void close() throws IOException** Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients. The following are the constructors.

public ServerSocket(int port) throws IOException

public ServerSocket(int port, int backlog) throws IOException

public ServerSocket(int port, int backlog, InetAddress address) throws IOException

public ServerSocket() throws IOException

Important methods

Sr.No.	Method & Description
1	**public int getLocalPort()** Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
2	**public Socket accept() throws IOException** Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.
3	**public void setSoTimeout(int timeout)** Sets the time-out value for how long the server socket waits for a client during the accept().
4	**public void bind(SocketAddress host, int backlog)** Binds the socket to the specified server and port in the SocketAddress object. Use this method if you have instantiated the ServerSocket using the no-argument constructor.

1.4CLIENT SERVER COMMUNICATION USING SOCKET

Example: Java socket programming in which client sends a text and server receives it.

MyServer.java

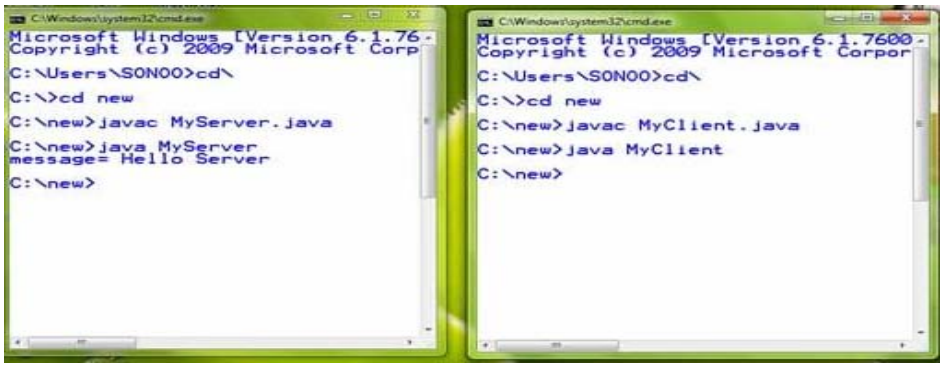
```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
```

```
dout.flush();

dout.close();

s.close();

}catch(Exception e){System.out.println(e);}

}
}
```

Run the program:



1.5 LET US SUM UP

This chapter focus on java networking and socket programming.

1.6CHECK YOUR PROGRESS

1. Explain Java Networking in brief.
2. Explain Socket Class in brief.
3. Explain ServerSocket class in brief.

1.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Refer 1.2
2. Refer 1.3
3. Refer 1.3

1.8 FURTHER READING

- For more detail refer Socket Programming in java book.

1.9 ASSIGNMENTS

1. Explain Java Networking in brief.
2. Explain Socket Class in brief.
3. Explain ServerSocket class in brief.

1.10 ACTIVITIES

- Create Client Server communication using in java.

Unit 2: Introduction of RMI 2

Unit Structure

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- RMI introduction
- RMI Architecture

2.2 INTRODUCTION

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton.

To write an RMI Java application, you would have to follow the steps given below −

- Define the remote interface
- Develop the implementation class (remote object)
- Develop the server program
- Develop the client program
- Compile the application
- Execute the application

2.3 RMI ARCHITECTURE

RMI Feature Gives Java Programmers Ability To Distribute Computing Across The Network. In the RMI model, the server defines objects that the client can use remotely RMI Defines Remote Interface That can Be Used To Create Remote Object. Client can Invoke Method of Remote Object the Same Syntax That is Use to Invoke Method on Local Object. RMI API Provides Classes And Methods That Handles All Communication and Parameter Referencing Requirement. RMI Also Handles Serialization of Object.

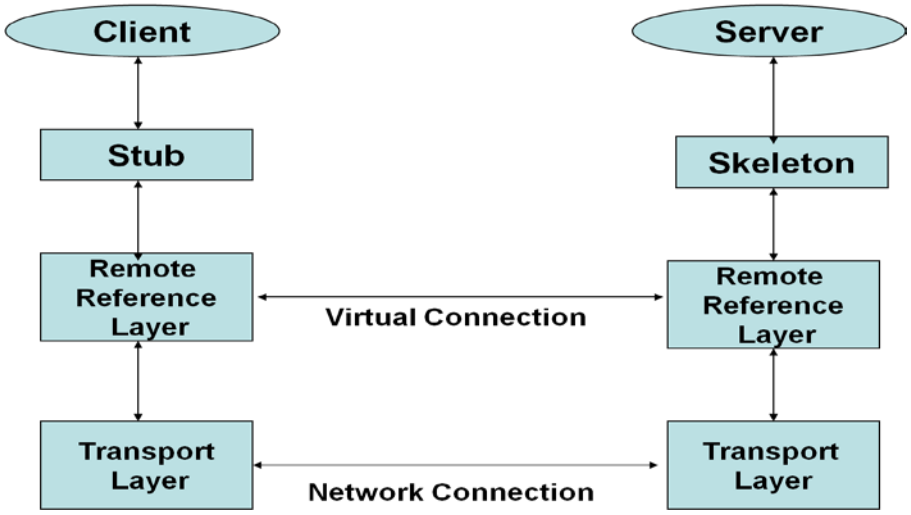


Figure 1: RMI Architecture

Stub:

- Stub basically act as a remote object proxies that are local to client.
- Stub is binding a call to server and find it.
- It also formatting data
- Ex: Marshalling And DMarshalling.
- Marshalling means converted message in proper format.
- The rmic tool will took specified class and generate a stub file for class which exposed all the methods to be used by clients. Stub name and class name is same.

Marshalling:

Example of Marshalling

host 1:------1,1.2,1.3,1.4,.1.5----------

Marshalling this data in binary format

010101010101010------

UnMarshalling this data into original format

Host 2:------1,1.2,1.3,1.4,1.5--------.

Skeleton:

- The stub resides on the client machine and the skeleton resides on the server machine.
- When a client invokes a server method, the JVM looks at the stub to do type checking. The request is then routed to the skeleton on the server, which in turn calls the appropriate method on the server object.
- In other words, the stub acts as a proxy to the skeleton and the skeleton is a proxy to the actual remote method.
- A skeleton is a helper class that is generated for RMI to use. The skeleton understands how to communicate with the stub across the RMI link.
- The skeleton carries on a conversation with the stub; it reads the parameters for the method call from the link, makes the call to the remote service implementation object, accepts the return value, and then writes the return value back to the stub.

Remote Reference Layer:

- The Remote Reference Layers defines and supports the invocation semantics of the RMI connection. This layer provides a RemoteRef object that represents the link to the remote service implementation object.
- The stub objects use the the RemoteRef object understands the invocation semantics for remote services.
- RMI provides only one way for clients to connect to remote service implementations: a multicast, point-to-point connection. Before a client can use a remote service, the remote service must be instantiated on the server and exported to the RMI system. (If it is the primary service, it must also be named and registered in the RMI Registry).

Transport Layer:

- The Transport Layer makes the connection between JVMs. All connections are stream-based network connections that use TCP/IP.
- Even if two JVMs are running on the same physical computer, they connect through their host computer's TCP/IP network protocol stack.
- RMI uses a wire level protocol called Java Remote Method Protocol (JRMP). JRMP is a proprietary, stream-based protocol.

- Sun and IBM have jointly worked on the next version of RMI, called RMI-IIOP, which will be available with Java 2 SDK Version 1.3. The interesting thing about RMI-IIOP is that instead of using JRMP, it will use the Object Management Group (OMG) Internet Inter-ORB Protocol, IIOP , to communicate between clients and servers.

2.4RMI REGISTRY & METHOD

The server object makes methods available for remote invocation by binding it to a name in the RMI Registry. The client object, can thus check for the availability of a certain server object by looking up its name in the registry. The RMI Registry thus acts as a central management point for Java-RMI. The RMI Registry is thus a simple name repository. It does not address the problem of actually invoking remote methods.

Package

Import rmi. *;

Import rmi. server.*;

Exception

RemoteException

Method

Rebind()

Bind()

Number of Steps:

- Define Interface for the remote classes.
- Create and Compile Implementation Classes for The Remote Classes.
- Create Stub and Skeleton Classes using rmic Command.
- Create and Compile Server Application.
- Start The RMI Registry and Server Application
- Create And Compile a Client Program to Access Remote Object
- Test Client

2.5 LET US SUM UP

This chapter focus on the RMI architecture.

2.6 CHECK YOUR PROGRESS

1. What is the use of RMI?
2. Explain the RMI Architecture in brief.

2.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Refer 2.3
2. Refer 2.4

2.8 FURTHER READING

For more focus on RMI read RMI Programming in java.

2.9 ASSIGNMENTS

- Demonstrate use of RMI Architecture.

Unit 3: RMI Implementationand Client-Server Programming

3

Unit Structure

3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- RMI Implementation
- Create RMI client server programming

3.2 INTRODUCTION

A remote interface provides the description of all the methods of a particular remote object. The client communicates with this remote interface.

To create a remote interface −

- Create an interface that extends the predefined interface **Remote** which belongs to the package.
- Declare all the business methods that can be invoked by the client in this interface.
- Since there is a chance of network issues during remote calls, an exception named **RemoteException** may occur; throw it.

3.3DEVELOPING THE IMPLEMENTATION CLASS

We need to implement the remote interface created in the earlier step. (We can write an implementation class separately or we can directly make the server program implement this interface.)

To develop an implementation class −

- Implement the interface created in the previous step.
- Provide implementation to all the abstract methods of the remote interface.

3.4DEVELOPING SERVER - CLIENT PROGRAM

An RMI server program should implement the remote interface or extend the implementation class. Here, we should create a remote object and bind it to the **RMIregistry**.

To develop a server program −

- Create a client class from where you want invoke the remote object.
- **Create a remote object** by instantiating the implementation class as shown below.
- Export the remote object using the method **exportObject()** of the class named **UnicastRemoteObject** which belongs to the package **java.rmi.server**.
- Get the RMI registry using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.
- Bind the remote object created to the registry using the **bind()**method of the class named **Registry**. To this method, pass a string representing the bind name and the object exported, as parameters.

Write a client program in it, fetch the remote object and invoke the required method using this object.

To develop a client program −

- Create a client class from where your intended to invoke the remote object.
- Get the RMI registry using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.
- Fetch the object from the registry using the method **lookup()** of the class **Registry** which belongs to the package **java.rmi.registry**.

  To this method, you need to pass a string value representing the bind name as a parameter. This will return you the remote object.
- The lookup() returns an object of type remote, down cast it to the type Hello.
- Finally invoke the required method using the obtained remote object.

3.5 CLIENT-SERVER PROGRAMMING USING RMI

Create Interface:

```
import java.rmi.*;
public interface RMIInter extends Remote{
        public int sum(int a,int b) throws RemoteException;
}
```

Implement Interface:

```
import java.rmi.*;
import java.rmi.server.*;
public class RMIInterImpl extends UnicastRemoteObject implements RMIInter
{
        public RMIInterImpl() throws RemoteException{
        }
        public int sum(int a,int b) throws RemoteException{
                return(a+b);
        }
}
```

Create Server:

```
import java.rmi.*;
//import java.rmi.regisrty.
public class RMIInterServer{

        public static void main(String args[]){
                RMIInterImpl rii;
                try{
                        rii= new RMIInterImpl();
                        Naming.rebind("RMIInterServer",rii);
                }
                catch(Exception e){
                }
```

```
        }
}
```

Create Client:

```
import java.rmi.*;
//import java.rmi.regisrty.
public class RMIInterClient {
        public static void main(String args[]){
                try{
                        String url="RMIInterServer";
                        RMIInter rmi=(RMIInter)Naming.lookup(url);
                        System.out.println(rmi.sum(10,20));
                }
                catch(Exception e){
                        e.printStackTrace();
                }
        }
}
```

How to RUN?

Within same folder you can see four files

1. Remote Interface : HI.java
2. Remote Interface Definition: HD.java
2. Server : HS.java
3. Client : HC.java

Set all required paths to run java program.

Now carefully these following commands in given order

1. javac HD.java , javac HI.java
2. javac HS.java
3. rmic HS
4. javacHC.java
5. close window.

Open another command window and type.

1. rmiregistry
2. minimize this window (remember dont close this window)

open another command window and type

1. policytool

and you can see java policy setting tool

1.1 click on add policy entry

1.2 another form will be displayed within that form click on add permission

1.3 again another form will be displayed within that click on permission list box and select "AllPermission" then click ok.

1.4 then click "done"

1.5 then select file->save menu and give name any name for ex: mypolicy then close this form (Remember svae this file to the same location at where your RMI files saved)

1.6 close this window

open command window apply the following command

1 java -Djava.security.policy=mypolicy HS

Open another command apply the following command

1 java -Djava.security.policy=mypolicy HC

3.6 LET US SUM UP

This chapter focus on the RMI implementation and RMI programming example.

3.7 CHECK YOUR PROGRESS

1. Write a short note on RMI Client – Server Programming.

3.8 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Refer 3.3 &Refer 3.4

3.9 FURTHER READING

- For more detail refer RMI implementation.
- Refer Tutorial Point RMI Practical.

3.10 ACTIVITIES

- Create an RMI application for client server communication using java RMI.

Block-4

Servlet and JSP

Unit 1: Introduction of Servlet 1

Unit Structure

1.1 LEARNING OBJECTIVE

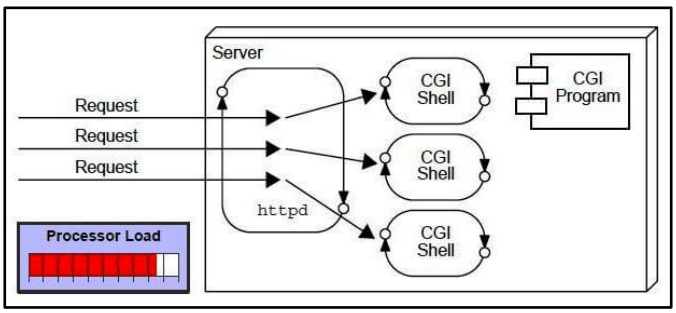
After going through this unit, you should be able to know:

- how to install the Servlet Engine / Web Server;
- basics of Servlet and how it is better than other server extensions;
- how the Servlet engine maintains the Servlet Life Cycle;
- where do we use HttpServletRequest Interface and some of its basic methods;
- where do we use HttpServletResponse Interface and some of its basic methods;
- what is session tracking;
- different ways to achieve Session Tracking like HttpSession & persistent cookies, and
- different ways to achieve inter-servlet communication.

1.2 INTRODUCTION TO SERVLET

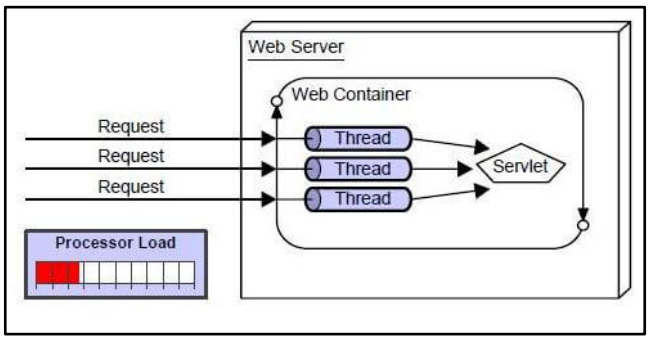
Servlet technology is used to create a dynamic web application, resides at server side and generates a dynamic web page. The technology is robust and scalable as it is based on the Java language. Servlet can be described in many ways, depending on the context, the servlet is a technology used to create a web application, it is mainly used to write a business logic part in an enterprise web application.

Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology.



There are many problems in CGI technology If the number of clients increases, it takes more time to prepare and response the users. For each user request, web server has to starts a new process, and a web server have limited memory space to start a new processes. It uses platform dependent language such as C++, Perl.

Over the CGI, the Servlet has many advantages, the web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the processes such as they share a common memory area, lightweight, cost of communication between the threads are low.



The advantages of Servlet are as follows:

- Better performance: because it creates a thread for each request, not process.
- Portability: because it uses Java language.
- Robust: JVM manages Servlets, so we don't need to worry about the memory leak and garbage collection.
- Secure: because it uses Java language.

There are many interfaces and classes in the Servlet API such as *Servlet*, *GenericServlet*, *HttpServlet*, *ServletRequest*, *ServletResponse*, etc. *GenericServlet* is not specific to any protocol while *HttpServlet* is specific to the HTTP protocol and use to create a Servlet that handles the HTTP requests.

1.3 CREATE YOUR FIRST SERVLET

The *javax.servlet* and *javax.servlet.http* packages represent interfaces and classes for servlet API. The *javax.servlet* package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol. The

javax.servlet.http package contains interfaces and classes that are responsible for requests only.

HelloWorld.java

Write the first servlet program, save it as *HelloWorld.java*

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

publicclassHelloWorldextendsHttpServlet {

publicHelloWorld() {
super();
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("<h1>Hello World!</h1>");
    }

protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Like other java programs, you can compile the servlet program as well through the command line using java compiler.

```
Desktop mantavyagajjar$ javac HelloWorld.java
```

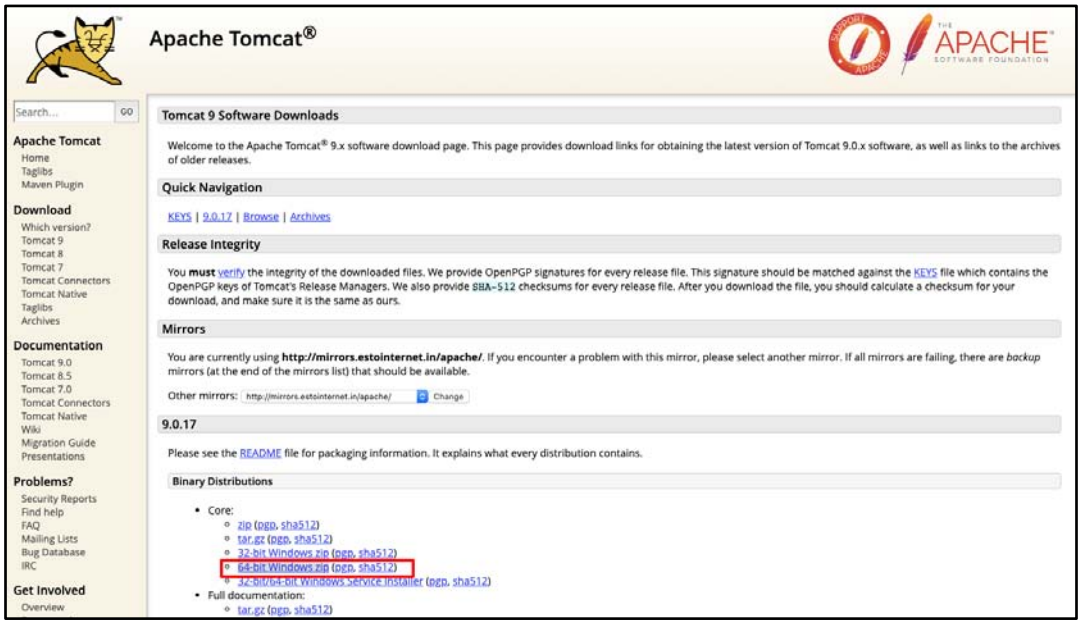
You may get an error, as java servlet is not a normal java program it runs on the web server we need to add the support for java web API, a **servlet-api.jar** library should be added in the CLASSPATH

The **servlet-api.jar** can be found as a part of the web server or web container or can be downloaded from external source too.

To test the output of Servlet, you have to deploy servlet into a web server or web containers such as JBoss or Tomcat. The most popular and lightweight web server and the container is Apache Tomcat.

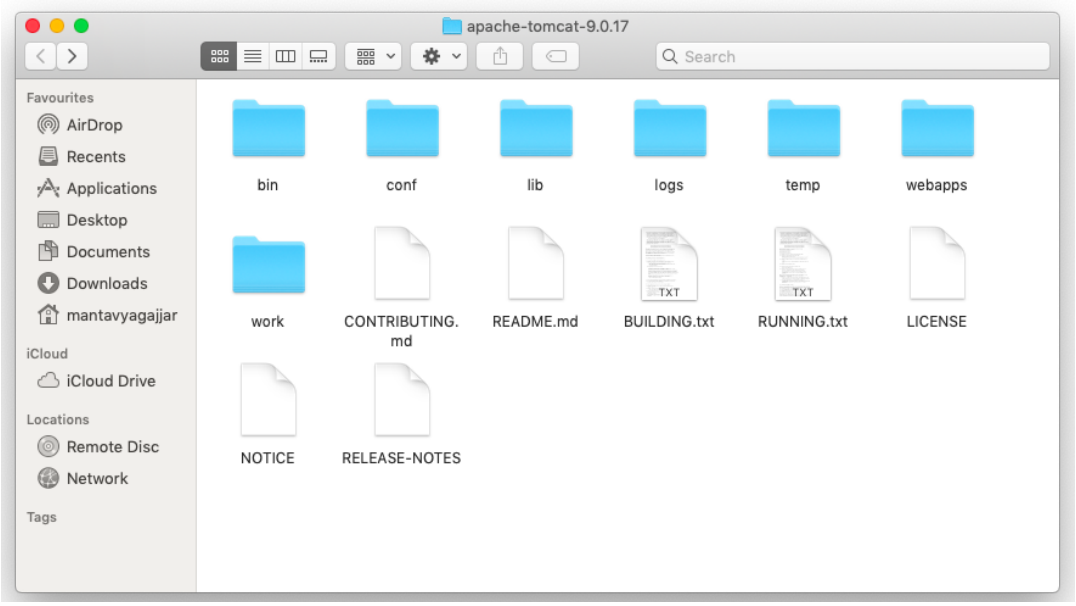
Download Apache Tomcat

Download the Apache Tomcat server from https://tomcat.apache.org/download-90.cgi, the current version is 9.0. If you are using windows platform choose http://mirrors.estointernet.in/apache/tomcat/tomcat-9/v9.0.17/bin/apache-tomcat-9.0.17-windows-x64.zip. If you are working on Linux or MacOS, the best option is to download the source code http://mirrors.estointernet.in/apache/tomcat/tomcat-9/v9.0.17/src/apache-tomcat-9.0.17-src.tar.gz.



Apache Tomcat Website Home Page - http://tomcat.apache.org/

Install the Tomcat server or extract the source depending on the platform you use. You will get the list of directories after the installation of Tomcat Server.



The directory structure after extract of Apache Tomcat

The bin directory contains the list of the commands used to start, stop the server or check the version of Tomcat Server. The lib directory contains the list of libraries required for the web API including **servlet-api.jar**, the webapps directory contains the web applications, we have to add our application into webapps directory.

Add the **servlet-api.jar** to the CLASSPATH. The servlet-api.jar is available under the Tomcat lib directory.

```
export CLASSPATH="/Users/mantavyagajjar/apache-tomcat-9.0.17/lib/servlet-api.jar"
```

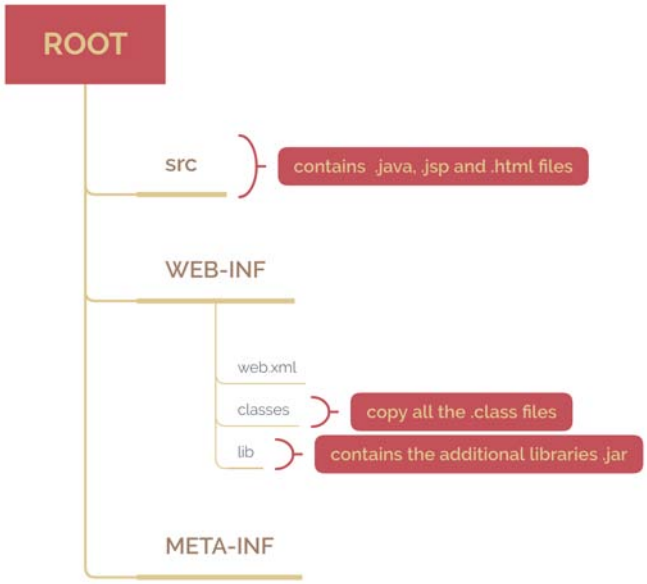
Now, you should be able to compile your Servlet java program using the javac command.

```
Desktop mantavyagajjar$ javac HelloWorld.java
```

Create a web application

Servlet program is not like, writing Java code and execute through command prompt. We need to follow the following steps in order to develop any servlets program. Even for a simple " Hello World" program also one must follow this standard directory structure which is prescribed.

1. Create a root directory with your web app name, create a subdirectory with name ‘src’ and move servlet program in that directory
2. Create sub-directory called WEB-INF in the root directory, this WEB-INF contains the web.xml file.
3. Create a directory called classes under the WEB-INF directory.
4. Compile the servlet *HelloWorld.java* we moved to src directory, you will get the .class file, copy that .class file into classes directory under the WEB-INF directory.



Now, are ready to launch the tomcat server, to start the Tomcat server go to bin directory and run thestartup.sh (If you use windows operating system, you should runthestartup.bat file to start the tomcat server)

```
Desktop mantavyagajjar$ ./startup.sh
```

Open http://localhost:8080/hello/HelloWorld into the browser, you should get the “Hello World!” string as a result.

Web Descriptor, *web.xml* is called a deploymentdescriptor file, for every web app it has to be created under WEB-INF directory, it contains the configuration for the application. Servlet and servlet mapping are one of the parameters used to define on with URL the servlet is accessible.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                  http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
   version="4.0" metadata-complete="true">

<description>Hello World</description>
<display-name>Hello World</display-name>
<servlet>
<servlet-name>HelloWorld</servlet-name>
<servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HelloWorld</servlet-name>
<url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
</web-app>
```

Here, in this example, the servlet will be called when user access /HelloWorldURL from the web-browser.

WebServlet Annotation

WebServlet annotation is an alternative way to define the servlet configuration, all the servlet has to be defined under the web.xml file with their name and URL-

mapping, using WebServlet you can do the same while writing the java file. So, you can ignore the configuration of servlet under the web.xml. Let's see where is the difference when you define the servlet mapping using the WebServlet annotation.

HelloWorld.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "HelloWorld", urlPatterns = {"/HelloWorld"})
publicclassHelloWorldextendsHttpServlet {

publicHelloWorld() {
super();
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("Hello World!");
    }

protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
```

```
    version="4.0" metadata-complete="false">

</web-app>
```

Instead of defining the servlet and servlet-mapping into the XML file, it is defined into java file just above the class using an annotation, also one parameter in web.xml has changed from true to false,metadata-complete="false". Define the servlet configuration using an annotation is super clean and easy to understand.

1.4 SERVLET LIFECYCLE

The entire life cycle of a Servlet is managed by the **Servlet container** which uses the *javax.servlet.Servlet* interface to understand the Servlet object and manage it. So, before creating a Servlet object let's first understand the life cycle of the Servlet object which is actually understanding how the Servlet container manages the Servlet object.

Stages of the Servlet Life Cycle: The Servlet life cycle mainly goes through four stages,

- Loading a Servlet.
- Initializing the Servlet.
- Request handling.
- Destroying the Servlet.

Let's look at each of these stages in details:

Loading a Servlet

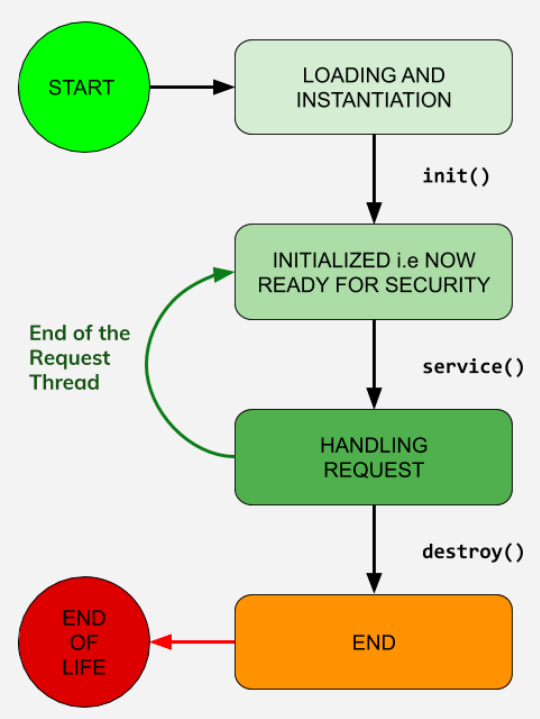
The first stage of the Servlet life cycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages :

- Initializing the context, on configuring the Servlet with a zero or positive integer value.

- If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that the Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

- Loading: Loads the Servlet class.
- Instantiation: Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.



Initializing a Servlet

After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the *Servlet* object by invoking the *Servlet.init(ServletConfig)* method which accepts *ServletConfig* object reference as parameter.

The Servlet container invokes the *Servlet.init(ServletConfig)* method only once, immediately after the *Servlet.init(ServletConfig)* object is instantiated successfully. This method is used to initialize the resources, such as JDBC data source.

Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the *ServletException* or *UnavailableException*.

Handling request

After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request :

- It creates the ServletRequest and ServletResponse objects. In this case, if this is HTTP request then the Web container creates HttpServletRequest and HttpServletResponse objects which are subtypes of the ServletRequest and ServletResponse objects respectively.
- After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

The *service()* method while processing the request may throw the *ServletException* or *UnavailableException* or IOException.

Destroying a Servlet

When a Servlet container decides to destroy the Servlet, it performs the following operations,

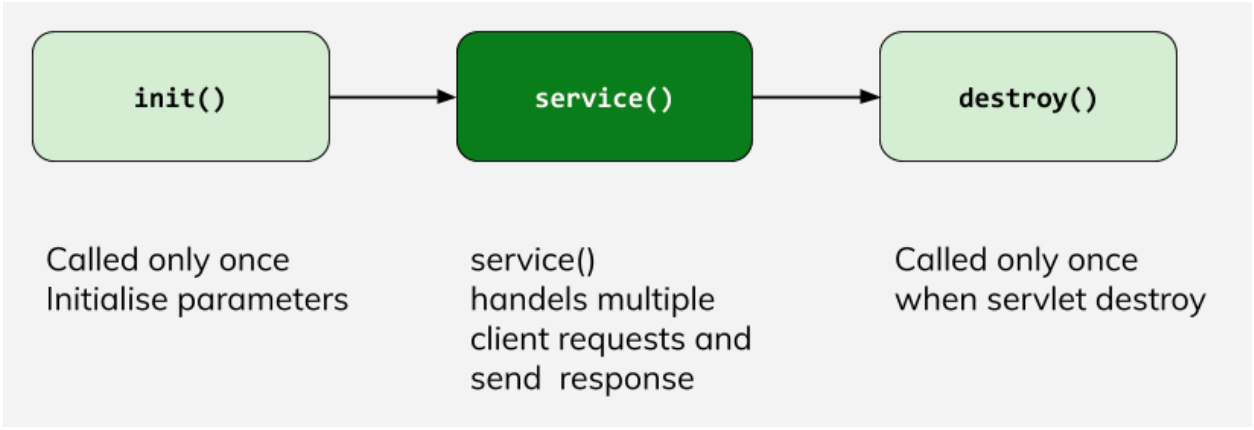
- It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
- After currently running threads have completed their jobs, the Servlet container calls the *destroy()* method on the Servlet instance.

After the *destroy()* method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

1.5SERVLET LIFE CYCLE METHODS

There are three life cycle methods of a Servlet :

- init()
- service()
- destroy()



Let's look at each of these methods in detail:

init() method

The S ervlet.init() method i s called by the S ervlet co ntainer t o i ndicate t hat t his Servlet instance is instantiated successfully and is about to put into service.

```
publicclassMyServletimplementsServlet {
publicvoidinit(ServletConfig config) throws ServletException {
//initialization code
  }
//rest of code
}
```

service() method

The service() method of the Servlet is invoked to inform the Servlet about the client requests.

- This method uses the ServletRequest object to collect the data requested by the client.
- This method uses a ServletResponse object to generate the output content.

```
// service() method

publicclass HelloWorld implements Servlet {
publicvoid service(ServletRequest res, ServletResponse res)
throws ServletException, IOException {
        // request handling code
    }
    // rest of code
}
```

destroy() method

The destroy() method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

```
//destroy() method

publicvoiddestroy() {
}
```

As soon as the destroy() method is activated, the Servlet container releases the Servlet instance.

1.6 TYPES OF SERVLETS

There are two main servlet types, Generic and HTTP:

Generic servlet, extend *javax.servlet.GenericServlet*.They are protocol independent. They contain no inherent HTTP support or any other transport protocol.

HTTP servlet, extend *javax.servlet.HttpServlet*.Have built-in HTTP protocol support and are more useful in a Sun Java System Web Server environment.

For both servlet types, you implement the constructor method *init()* and the destructor method *destroy()* to initialize or deallocate resources.

All servlets must implement a *service()* method, which is responsible for handling servlet requests. For generic servlets, simply override the service method to provide routines for handling requests. HTTP servlets provide a service method that automatically routes the request to another method in the servlet based on which HTTP transfer method is used. So, for HTTP servlets, override *doPost()* to process POST requests, *doGet()* to process GET requests, and so on.

The previous example *HelloWorld.java* we inherit *HttpServlet* and implement *doGet* and *doPost* methods to print "Hello world!", let's write a program to have the same result using *GenericServlet*and try to understand how *GenericServet* and *HttpServlet* are different from each other.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.GenericServlet;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

publicclassHelloWorldextendsGenericServlet {
privatestaticfinallong serialVersionUID = 1L;

publicHelloWorld() {
super();
   }
@Override
publicvoidservice(ServletRequest request, ServletResponse response)
throws IOException, ServletException{

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
```

```
        out.print("Hello World!");
    }
}
```

The *HttpServlet* has *doGet* and *doPost* methods are used to receive the data which are transferred by the HTTP POST and GET methods while *GenericServlet* has the *service* method, which is independent of any protocol. There are a couple of differences listed below.

GenericServlet	HttpServlet
Can be used with any protocol (means, you can create a servlet that can handleFTPrequest, to upload or delete the file). Protocol independent.	Should be used with HTTP protocol only (can handle HTTP specific requests). Protocol dependent.
All methods are concrete except *service()* method. service() method is an abstract method.	All methods are concrete (non-abstract). *service()* is non-abstract method.
service() should be override in the class which implement the *GenericServlet*.	*service()* method need not be overridden.
It is must to use *service()* method as it is a callback method.	Being *service()* is non-abstract, it is replaced by *doGet()* or *doPost()* methods.
Extends Object and implements interface Servlet, *ServletConfig*, and Serializable.	Extends *GenericServlet* and implements interface Serializable
Direct subclass of *Servlet* interface.	Direct subclass of *GenericServlet*.

Defined *javax.servlet* package.	Defined javax.servlet.http package.
All the classes and interfaces belonging to *javax.servlet* package are protocol independent.	All the classes and interfaces present in *javax.servlet.http* package are protocol dependent (specific to HTTP).
Used to handle the protocols other then HTTP.	Used always when handling HTTP request.

Check Your Progress 1

1. State True or False:
    a. Servlet is not a Java Class. T/F
    b. Tomcat 4.0 is an open source and free Servlet Container and JSP Engine. T/F
    c. init() and destroy() methods will be called only once during the lifetime of the Servlet. T/F
2. What are the advantages of servlets over other common server extension mechanisms?
3. Write a Servlet program to display “Welcome to Fifth semester of MCA”
4. Explain different between doGet() and doPost() methods of HttpServlet.
5. Draw a Servlet Life Cycle, to represent the different phases of Servlet Life Cycle.

1.7 SERVLET REQUEST AND RESPONSE

The main job of Servlet is to handle the client’s request, process data on the server, and respond to the client back. Servlet API provides two important interfaces *javax.servlet.ServletRequest* and *javax.servlet.ServletResponse.* The implementation of those interfaces are provided

in*javax.servlet.http.HttpServletRequest* and *javax.servlet.http.HttpServletResponse*to encapsulate client request.

Capture user Input

There are two types of information encapsulated in the requests, system generated and user input data. Let's see an example of how user data can be accessed in the servlet which was entered by the user on HTML web page and create a custom hello message based on the user input.

Index.html

Create anHTML file with the input box, where user can enter the name, on submission of form data entered in the input box passed to the servlet inform of key and value pair.

```
<!DOCTYPE html>
<html>
<head>
<title>User Input Form</title>
</head>
<body>
<form action="/hello/HelloForm" method="get">
<p>Enter your name: <input type="text" name="name"></p>
<input type="submit" value="login">
</form>
</body>
</html>
```

HelloForm.java

Based on the method either GET or POST through which servlet called, is based on the method defined on the form. The form data is prepared in form of a key, value pair and passed to the servlet, a piece of individual key information can be accessed through *getParameter(name)* method, you can iterate on all the keys using *getParameterValues()* method.

```
import java.io.*
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(name = "HelloForm", urlPatterns = {"/HelloForm"})
publicclassHelloFormextendsHttpServlet {

publicHelloForm() {
super();
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("<h1>Hello " + request.getParameter("name") + " !</h1>");
    }

protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doGet(request, response);
    }
}
```

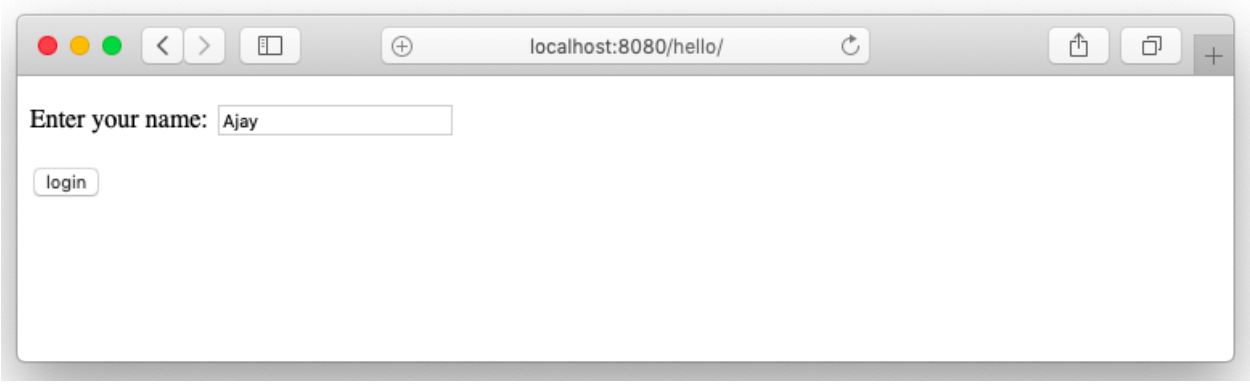
Web.xml

<welcome-file> parameter is used to search for the default file when user access application, you can define multiple default files too.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
    version="4.0" metadata-complete="false">

<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>
```

Access your application /hello and you will see input-box, enter the name and click on submit button.



The name will be passed to a servlet, it creates a new page with a custom message generated by the servlet.

Capture the system parameters

When a user clicks a hyperlink or a submit button, we know that the data entered by a user in the form fields are sent to the server. Along with user input a lot of extra information goes to the server as a request header attached to the request object. Servlet request object can getthose information using *getHeaderNames()* and *getHeader()* methods of *HttpServletRequest* interface.

Apart from user data, other data received in the request header such as client IP address, local port used by browser to initiate a request, browser name and version, user's current language, and many other information attached to the request header.

Let's modify our program to get all this information and print it on the web page along with the output.

HelloForm.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
@WebServlet(name = "HelloForm", urlPatterns = {"/HelloForm"})
publicclassHelloFormextendsHttpServlet {

publicHelloForm() {
super();
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("<h1>Hello " + request.getParameter("name") + " !</h1><br/>");

        Enumeration e = request.getHeaderNames();

while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = request.getHeader(name);
            out.println("<b>" + name + "</b> = " + value + " <br/>");
        }
    }

protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doGet(request, response);
    }}
```

Request Dispatcher

The *RequestDispatcher* interface provides the facility of dispatching the request to another resource it may be HTML, servlet or JSP. This interface can also be used to include the content of another resource also.

There are two methods defined in the RequestDispatcher interface. Forward transfer a request to another resource (Servlet, JSP file, or HTML file) on the server.

```
RequestDispatcher rd=request.getRequestDispatcher("/Login");
rd.forward(request, response);
```

Include the content of a resource (Servlet, JSP page, or HTML file) in the response.

```
RequestDispatcher rd=request.getRequestDispatcher("/Login");
rd.include(request, response);
```

The main difference between *include()* and *forward()* is that include method is used to load the contents of the specified resource, could be a Servlet, JSP, or static resource e.g. HTML files directly into the Servlet's response. On the other hand, forward method is used for server side redirection, where an HTTP request for one servlet is routed to another resource for processing.

1.8COOKIE IN SERVLET

A cookie is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

By default, each request is considered a new request. In cookies technique, we add a cookie with the response from the servlet. So cookie is stored in the cache of the browser. After that, if the request is sent by the user, a cookie is added with a request by default. Thus, we recognize the user as the old user.

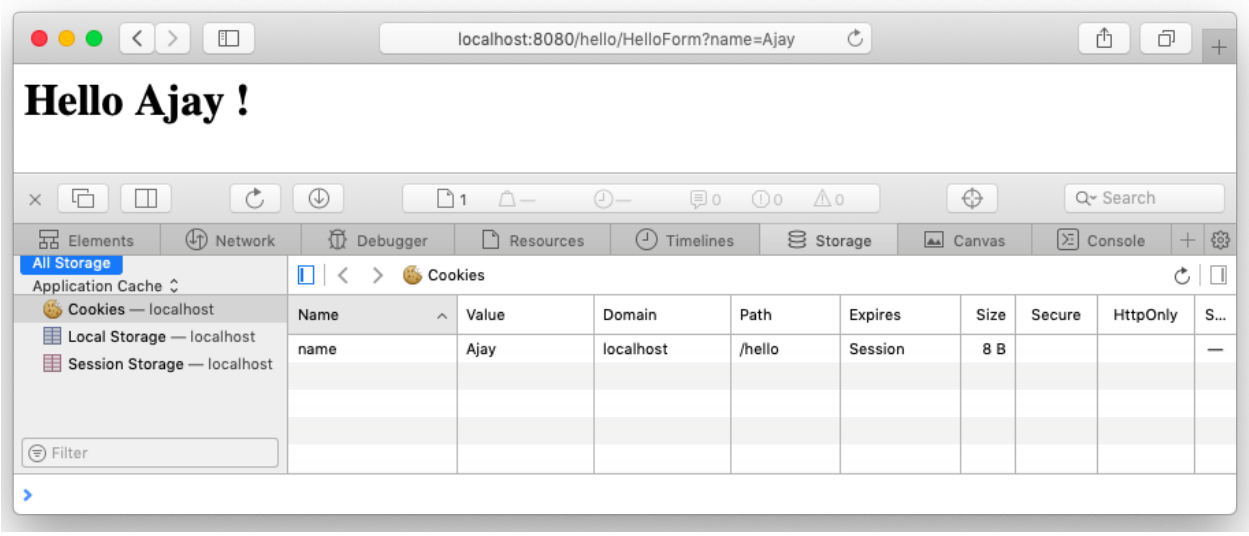
There are 2 types of cookies, Non-persistent cookie and Persistent cookie. Non-persistence is valid for a single session only. It is removed each time when the user closes the browser while Persistent is valid for multiple session. It is not removed each time when a user close the browser. It is removed only when the user logs out or sign out.

It is a simple technique of maintaining the state at the client browser. But, it will not work if the cookie is disabled on the browser. Only textual information can be set in Cookie.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(name = "HelloForm", urlPatterns = {"/HelloForm"})
publicclassHelloFormextendsHttpServlet {

publicHelloForm() {
super();
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        response.setContentType("text/html");
        String name = request.getParameter("name");

//set the cookie in client's browser
        response.addCookie(new Cookie("name",name));

        PrintWriter out = response.getWriter();
        out.println("<h1>Hello " + name + " !</h1><br/>");
    }

protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
```

```
        doGet(request, response);
    }
}
```

The cookie can be accessed through JavaScript or Servlet, have a look at a client browser in below screen.



1.9SESSION MANAGEMENT

The *HttpSession* object is used for session management. A session contains information specific to a particular user across the whole application. When a user enters into a website or an online application for the first time *HttpSession* isobtained via *request.getSession()*, the user request is given a unique ID to identify his session. This unique ID can be stored into a cookie in a request parameter.

The *HttpSession* stays alive until it has not been used for more than the timeout value specified in **web.xml** deployment descriptor file. The default timeout value is 30 minutes, this is used if you don't specify the value in web.xml. This means that when the user doesn't visit web application until 30 minutes, the session is destroyed by the servlet container. The subsequent request will not be served from this session anymore and the servlet container will create a new session.

Let’s create an example that demonstrates how a session can be created and store information in the session.

ProcessRequest.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(name = "ProcessRequest", urlPatterns = {"/ProcessRequest"})
publicclassProcessRequestextendsHttpServlet {

publicProcessRequest() {
super();
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        HttpSession session = request.getSession(false);
if (session == null) {
            response.sendRedirect("/hello/Login.html");
        }

        response.setContentType("text/html");
        String name = request.getParameter("name");

        PrintWriter out = response.getWriter();
        out.println("<h1>Hello " + name + " !</h1><br/>");
    }
```

```
protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        doGet(request, response);
    }
}
```

The method *getSession(false)* return a session, if it was created and alive. If session is not found means that the user is not logged inso, redirect to the login page.If user login successfully a message will be printed (Hello Ajay!) on the screen. Let's write a code for a Login.html page and Login.java servlet.

Login.html

```
<!DOCTYPE html>
<html>
<head>
<title>Login Page</title>
</head>
<body>
<form action="/hello/Login" method="POST">
<p>Username:<br/>
<input type="text" name="name"/>
</p>

<p>Password: <br/>
<input type="password" name="name"/>
</p>

<input type="submit" value="Login"/>
```

```
</form>
</body>
</html>
```

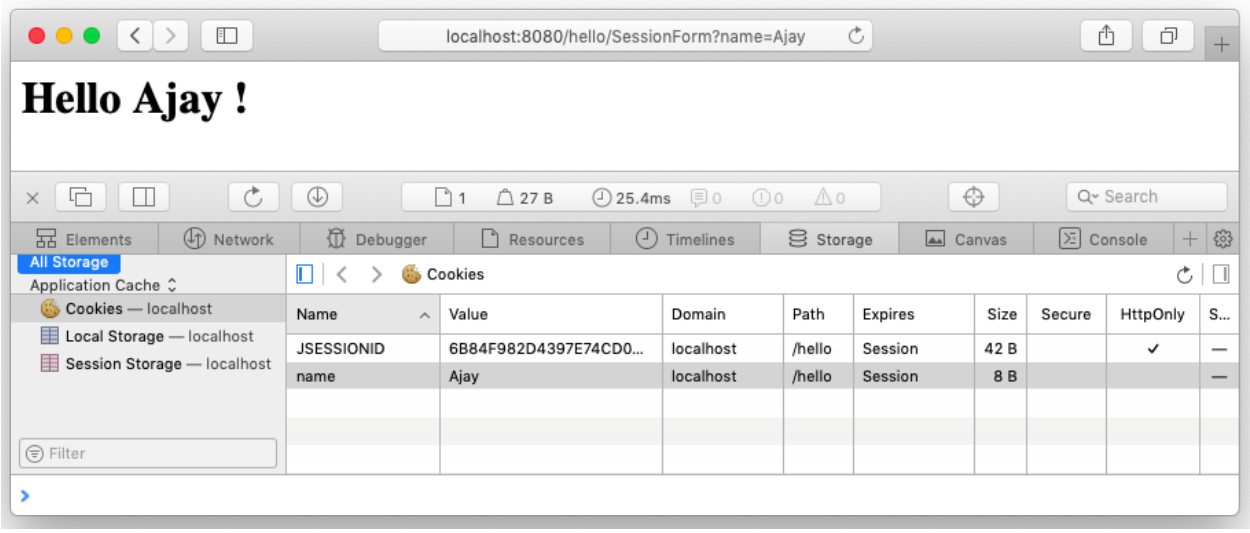
The userentersusername and password, and submit the form to /hello/LoginServlet, Login servlet verify the user, create a new session if the user is valid. Look at the below code of Login.java servlet.

Login.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(name = "Login", urlPatterns = {"/Login"})
publicclassLoginextendsHttpServlet {

publicLogin() {
super();
   }

protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

      String name = request.getParameter("user");
      String password = request.getParameter("password");

      HttpSession session = request.getSession();
      User user = new User(name, password);
```

```
if (user.validUser()) {
            session.setAttribute("userObject", user);
            response.sendRedirect("/hello/ProcessRequest");
        }
    }
}
```

On successful login, a message (Hello Ajay!) will be printed on the client browser, you can also track the session key stored in the cookie.



Check Your Progress 2

1. What are the main functions of the *HTTPServletRequest* Interface? Explain the methods which are used to obtain cookies and query string from the request object.
2. What are the main functions of the *HTTPServletResponse* Interface? Explain the methods which are used to add cookies to response and send an error response.
3. Explain various purposes for which we use Session tracking. Also, Explain in brief the two ways to handle Session Tracking in Servlets.
4. What are the two ways used for Servlet collaboration Servlet Programming

5. How do I call a servlet with parameters in the URL?
6. How do I deserialize an HTTP session?
7. How do I restrict access to servlets and JSPs?
8. What is the difference between JSP and servlets?
9. Difference between GET and POST .
10. Can we use the constructor, instead of init(), to initialize servlet?
11. What are two different types of servlets? Explain the differences between these two.
12. What is the difference between ServletContext and ServletConfig?
13. What are the differences between a session and a cookie?
14. How will you delete a cookie?
15. What is the difference between Context init parameter and Servlet init parameter?
16. What are the different types of Servlet Engines?

1.10LET US SUM UP

Java servlets are small, platform-independent Java programs that run in a web server or application server and provide server-side processing such as enterprise commercial applications. Servlets are widely used for web programming. Servlets dynamically extend the functionality of a web server. A servlet engine can only execute servlet which is contained in the web-servers like, Tomcat or JBoss.

Servlets are basically developed for the server side applications and designed to handle HTTP requests. They are better than other common server extensions like CGI as they are faster, have all the advantages of Java language and supported by many of the browsers.

A Java Servlet has a lifecycle that defines how the servlet is loaded and initialized, how it receives and responds to requests, and how it is taken out of service. Servlets run within a Servlet Container, creation and destruction of servlets is the duty of Servlet Container. There are three principal stages in the life of a Java Servlet, namely: Servlet Initialisation, Servlet Execution, and Servlet Destruction. In first stage, the servlet's constructor is called along with the servlet *init()* method - this is called automatically once during the servlet execution life cycle.

Once your servlet is initialized, a request received by the Servlet Container, will be forwarded to Servlet's service() method. HttpServlet class breaks *service()* method into more useful *doGet()*, *doPost()*, *doDelete()*, *doOptions()*, *doPut()* and *doTrace()* methods depending on the type of HTTP request it received. When the application is stopped or Servlet Container shuts down, your Servlet's *destroy()* method will be called to clean up any resources allocated during initialization and to shutdown gracefully.

There are two important interfaces included in the servlet API. They are HttpServletRequest and HttpServletResponse. HttpServletRequest encapsulates the functionality for a request object that is passed to an HTTP Servlet. It provides access to an input stream and so allows the servlet to read data from the client and it has methods like *getCookies()*, *getQueryString()*& getSession, etc. HttpServletResponse encapsulates the functionality for a response object that is returned to the client from an HTTP Servlet. It provides access to an output stream and so allows the servlet to send data to the client and it has methods like *addCookie()*, *sendError()* and *getWriter(),* etc.

Session tracking is another important feature of the servlet. Every user of a site is associated with a javax.servlet.http.HttpSession object that servlets can use to store or retrieve information about that user.

A servlet uses its request object's *getSession()* method to retrieve the current *HttpSession* object and can add data to an *HttpSession* object with the putValue() method. Another technique to perform session tracking involves persistent cookies. A cookie is a bit of information sent by a web server to a browser and stores it on a client machine that can later be read back from that browser. For each request, a

cookie can automatically provide a client's session ID or perhaps a list of the client's preferences.

Servlets, which are running together on the same server, have several ways to communicate with each other. There are three reasons to use inter-servlet communication. First is Direct Servlet manipulation handling in which servlet can gain access to the other currently loaded servlets and perform some task on each. Second is Servlet Reuse that allows one servlet to reuse the abilities (the public methods) of another servlet. Third is Servlet collaboration that allows servlets to cooperate, usually by sharing some information.

Unit 2: Servlet with JDBC 2

Unit Structure

2.1 LEARNING OBJECTIVE

After going through this unit, you should be able to:

- Understand the different approach to establish the connection and fetch data into Servlet.
- Understand how to insert the record into the database through a Servlet.
- Understand the different approaches to update or delete the records in the database.
- Understand how to configure and use the connection pool in servlet to manage the database connection efficiently.
- Learn how Servlet filter works, let's verify the user and redirect to correct page using servlet filter.

2.2 INTRODUCTION

Accessing data from the database or in any other data sources is a significant operation in web programming. Data access in JSPs and Servlets is done through Java Database Connectivity (JDBC). There are two packages in JDBC 3.0-java.sql and javax.sql. The java.sql package is often referred to as the JDBC core application programming interface (API) and is sufficient to do basic data manipulations. The javax.sql package is the JDBC Optional Package API which provides additional features, including connection pooling, which will be discussed at the end of the chapter. Let's see the different days, you can do the database connection, reading data from and writing to the database.

2.3 CONNECTION TO DATABASE

You have already gone through the database connection and reading data from the database in java program, there is no change in reading data from the database when you are writing a Java Servlet program.

Let's take an example of a contact book application, we will connect to the database, read the contacts and display those contacts on the screen. Let's write a program that connects to the contact book database.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.sql.* ;

@WebServlet(name = "Contact", urlPatterns = {"/Contact"})
publicclassContactextendsHttpServlet {

private Connection conn = null;
private Statement stmt = null;
private ResultSet rset = null;

private String databaseUrl = "jdbc:postgresql://localhost:5432/contactbook";
private String username = "mantavyagajjar";
private String password = "********";

publicContact() {

   }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

       response.setContentType("text/html");
       PrintWriter out = response.getWriter();

try {
           Class.forName("org.postgresql.Driver");
this.conn = DriverManager.getConnection(databaseUrl, username, password);
       } catch (Exception e) {
```

```
            out.println("<h4>Connection to database unsuccessful</h4>");
return;
        }

        out.println("<h4>Connection to database successfully</h4>");
    }
}
```

Above example print the string in browser "**Connection to database successfully**" when your connection to the database is successful, else you will see the message "**Connection to database unsuccessful**".

When you open a connection on each user request make sure that it has to be closed properly at the end of the request. The connection has to be close d in the finally block to release all the resource acquired by the servlet to serve the request.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.sql.* ;

@WebServlet(name = "Contact", urlPatterns = {"/Contact"})
publicclassContactextendsHttpServlet {

private Connection conn = null;
private Statement stmt = null;
private ResultSet rset = null;

private String databaseUrl = "jdbc:postgresql://localhost:5432/contactbook";
```

```
private String username = "mantavyagajjar";
private String password = "********";

publicContact() {

   }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

       response.setContentType("text/html");
       PrintWriter out = response.getWriter();

try {
           Class.forName("org.postgresql.Driver");
this.conn = DriverManager.getConnection(databaseUrl, username, password);
       } catch (Exception e) {
           out.println("<h4>Connection to database unsuccessful</h4>");
       }

if (this.conn == null) {
return;
       }

       out.println("<h4>Connection to database successfully</h4>");

try {
           stmt = this.conn.createStatement();
           rset = stmt.executeQuery("SELECT * FROM dummy");
       } catch (SQLException e) {

       } finally {
```

```
try {
                out.close();
                stmt.close();
                conn.close();
            } catch (SQLException e) {

            }
        }

    }
}
```

Connection Parameters

In the above example, we have seen how the database connection is being opened and closed in the servlet program, it is not advisable to write the database connection parameters (*databaseURL*, *username*, and *password*) in a servlet program, as an enterprise application may have many servlets, and changing connection parameters leads to modify all those servlet programs who access the database.

It is advisable to write the database connection parameters (*databaseURL*, *username*, and *password*) into deployment descriptor file web.xml so that the servlet read those parameters during the initialized phase. i.e. *init(ServletConfig config)* method.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
    version="4.0" metadata-complete="false">
```

```
<context-param>
<param-name>databaseURL</param-name>
<param-value>jdbc:postgresql://localhost:5432/contactbook</param-value>
</context-param>
<context-param>
<param-name>username</param-name>
<param-value>mantavyagajjar</param-value>
</context-param>
<context-param>
<param-name>password</param-name>
<param-value>********</param-value>
</context-param>

<welcome-file-list>
<welcome-file>login.html</welcome-file>
<welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>
```

The se rvlet gets the par ameters in *init(ServletConfig config)* method t hrough *ServletConfig* when servlet get initialized by the servlet container.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.sql.* ;
```

```
@WebServlet(name = "Contact", urlPatterns = {"/Contact"})
publicclassContactextendsHttpServlet {

private Connection conn = null;
private Statement stmt = null;
private ResultSet rset = null;

private String databaseUrl = null;
private String username = null;
private String password = null;

@Override
publicvoidinit(ServletConfig config) throws ServletException {
super.init(config);

        ServletContext context = config.getServletContext();
        databaseURL = context.getInitParameter("databaseURL");
        username = context.getInitParameter("username");
        password = context.getInitParameter("password");
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    }
}
```

Now, the database connection parameters can be changed easily in deployment descriptor web.xml file.

Database Connection Approaches

Writing a single user program that connect to the database and read data from database is not a challenging compared to writing a multi-user enterprise application, you need to choose the right approach to connect to and reading data form the database. Let's understand the different approach available to wiring an enterprise application.

First Approach

Create JDBC *connection*object in init() method, use JDBC connection object to create statement JDBC object and write JDBC persistence logic in *service(request, response)*, *doGet(request, response)* or *doPost(request, response)* method. Close JDBC *connection* object in *destroy()* method.

In this approach, the JDBC *connection* object must be taken as an instance variable of the servlet program. So *connection* object is shared between multiple user requests and therefore it is not threaded safe.

Advantage is, all requests coming to the servlet program will use a single connection to interact with database. This improves the performance of web applications.

Disadvantage is, multiple threads may use a single connection object simultaneously or concurrently, which means programmer should take care of multithreading issues by using synchronization concept.

Second Approach

Create JDBC *connection* object in *service(request, response)*, *doGet(request, response)* or *doPost(request, response)* method. Use JDBC *connection* object to create *statement* object and develop JDBC Persistence logic in *service(request, response)*, *doGet(request, response)* or *doPost(request, response)* methods. Close JDBC *connection* object at the end of the*service(request, response)*, *doGet(request, response)* or *doPost(request, response)* method.

Advantage is, a JDBC connection object is a local variable of *service(request, response)*, *doGet(request, response)* or *doPost(request, response)* methods so no need to take care for the multithreading synchronization.

Disadvantage is, for every request one separate JDBC connection object will be created. So this approach degrades the performance.

Third Approach

Get JDBC *connection* object from JDBC *connection* pool in*service(request, response)*, *doGet(request, response)* or *doPost(request, response)* method, use JDBC connection object to create *statement* object and develop JDBC persistence logic in *service(request, response)*, *doGet(request, response)* or *doPost(request, response)* method. We do not have to close the connection object explicitly, as the connection will be return back to connection pool automatically at the end of *service(request, response)*, *doGet(request, response)* or *doPost(request, response)* methods.

Advantages are:

- JDBC connection object should be taken as a local variable of *service(request, response)*, *doGet(request, response)* or *doPost(request, response)* method. So there is no need to worry about multithreading issues.
- While working with JDBC connection pool, servlet programs are not responsible to create, manage and destroy JDBC connection object.
- We can use a minimum number of JDBC connection objects to handle more clients and requests interact with database.
- Connection Pooling approach perform better than Approach 2.
- Connection pool can be defined specific to a single web application or connection pool can be defined as shared between multiple web applications.

We will see how to use the connection pooling system in java web application at the end of this chapter in detail.

2.4 INSERT RECORD INTO THE DATABASE

The SQL Insert query should be executed in order to insert records in the database, open the connection, create a statement and execute an Insert SQL query through a statement.

The user inputs the values on the HTML form, those values are transferred to the servlet through GET or POST method, the servlet process the data and inserts into the database. Let's take an example of contact book, user input name, email and phone number fieldson the html form *(Contact.html)* and passed to the servlet to store those fields into the database.

Create a table into the database.

```
CREATETABLE contact(
        nameVARCHAR (50),
        email VARCHAR (50) UNIQUE,
        phone VARCHAR (50)
);
```

The connection information is set up in the application descriptor file web.xml, let's create a servlet that takes an input from the user (HTML form) and create a record into the database.

Contact.html

Takes input from the user and transferred to the servlet through GET method.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>Create Contact</title>
</head>
<body>
<form method="get" action="/contactbook/SaveContact">
<p>
    Name: <input type="text" name="name"><br>
    Email: <input type="text" name="email"><br>
    Phone: <input type="text" name="phone"><br>
</p>
<input type="submit" value="Create Contact">
</form>
</body>
</html>
```

Contact.java

Read t he va lues of f ields (name, em ail and phone) f rom request object, use *getParameter(name)* method to read an values captured and transferred by the html form *(Contact.html)*.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.sql.* ;

@WebServlet(name = "Contact", urlPatterns = {"/SaveContact"})
publicclassContactextendsHttpServlet {

private Connection conn = null;
private PreparedStatement preparedStmt = null;
```

```
private String databaseURL = null;
private String username = null;
private String password = null;

@Override
publicvoidinit(ServletConfig config) throws ServletException {
super.init(config);

        ServletContext context = config.getServletContext();
        databaseURL = context.getInitParameter("databaseURL");
        username = context.getInitParameter("username");
        password = context.getInitParameter("password");
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

try {
            Class.forName("org.postgresql.Driver");
this.conn = DriverManager.getConnection(databaseURL, username, password);
        } catch (Exception e) {

        }

if (this.conn == null) {
return;
        }

        String insertSQL = "INSERT INTO contact (name, phone, email) VALUES (?, ?,
```

```
?)";

try {
            preparedStmt = conn.prepareStatement(insertSQL);
            preparedStmt.setString(1, request.getParameter("name"));
            preparedStmt.setString(2, request.getParameter("phone"));
            preparedStmt.setString(3, request.getParameter("email"));
            preparedStmt.execute();

            out.println("Record created successfully");
            preparedStmt.close();
            conn.close();
        } catch (SQLException e) {
            out.println("Error Occurred : " + e);
        }
        out.close();
    }
}
```

The *java.sql.PreparedStatement* is an ideal way to execute the insert or update query as it verify the data according to the type before inserting into the database table, on the successful execution of the above servlet you can see the record is inserted into the contact table.

```
contactbook=# select * from contact;
name         |        email                  |   phone
-------------+--------------------+---------------+-------------
 Ajay Kumar  | ajay@gmail.com          | 9898098981
```

```
 Nikunj Jani   | nikunjjani@gmail.com  | 9898798985
(2 rows)
```

You may get an error on screen if duplicate record found, we have created *contact* table where email field is defined as unique.

```
Error Occurred :  org.postgresql.util.PSQLException: ERROR: duplicate key value violates unique constraint "contact_email_key" Detail: Key (email)=(ajay@gmail.com) already exists.
```

2.5 READING FROM DATABASE

Java web application has Servlet as a base technology, the servlet is a tool to write the controllers in MVC application model. Servlet can also help to secure the business process in web based enterprise applications, we can write the business logic part in servlet such as:

- Validate the use input as per the business need
- Populate the result by applying the business logic
- Insert or update the record into the table

Let's write a program to fetch the records form contact table and and display all thecontact records on web page.

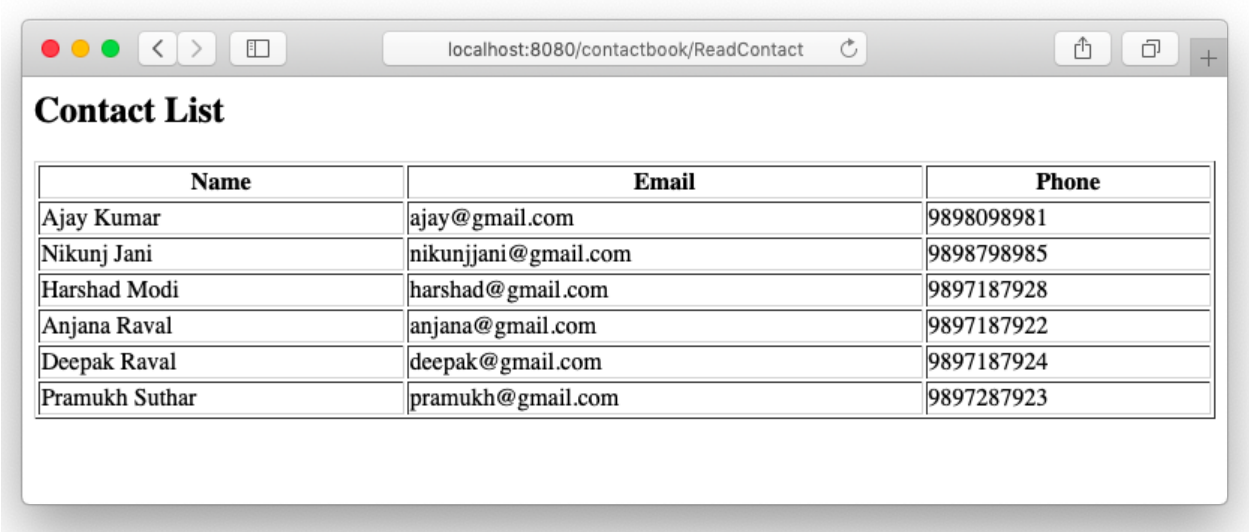
ReadContact.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.sql.* ;
```

```
@WebServlet(name = "Contact", urlPatterns = {"/ReadContact"})
publicclassReadContactextendsHttpServlet {

private Connection conn = null;
private PreparedStatement preparedStmt = null;

private String databaseURL = null;
private String username = null;
private String password = null;

@Override
publicvoidinit(ServletConfig config) throws ServletException {
super.init(config);
        ServletContext context = config.getServletContext();
        databaseURL = context.getInitParameter("databaseURL");
        username = context.getInitParameter("username");
        password = context.getInitParameter("password");
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

try {
            Class.forName("org.postgresql.Driver");
this.conn = DriverManager.getConnection(databaseURL, username, password);
        } catch (Exception e) {
return;
        }
```

```
        String insertSQL = "SELECT * FROM contact WHERE 1=1";

try {
            preparedStmt = conn.prepareStatement(insertSQL);
            ResultSet rs = preparedStmt.executeQuery();


            out.print("<table border=\"1\"><tr>");
            out.print("<th>Name</th><th>Email</th>");
            out.print("<th>Phone</th></tr>");
while(rs.next()) {
                out.print("<tr><td>"+rs.getString("name")+"</td>");
                out.print("<td>"+rs.getString("email")+" </td>");
                out.print("<td>"+rs.getString("phone")+"</td></tr>");
            }
            out.println("</table>");

            preparedStmt.close();
            conn.close();
        } catch (SQLException e) {
            out.println("Error Occured" + e);
        }
        out.close();
    }
}
```

Access t he URLhttp://localhost:8080/contactbook/ReadContact you wi ll se e all t he contacts you have created in the database.



Contacts available in the database

```
contactbook=# select * from contact;
name          |       email                |   phone
---------------+--------------------+-----------
 Ajay Kumar       | ajay@gmail.com         | 9898098981
 Nikunj Jani      | nikunjjani@gmail.com   | 9898798985
 Harshad Modi     | harshad@gmail.com      | 9897187928
 Anjana Raval     | anjana@gmail.com       | 9897187922
 Deepak Raval     | deepak@gmail.com       | 9897187924
 Pramukh Suthar   | pramukh@gmail.com      | 9897287923
(6 rows)
```

Reading all the data from the database table may slow down the application performance when you have millions of records stored into the database table. The performance can be improved when we fetch and display only the required data. Let's modify our program *(ReadContact.java)* to display only requested data by the user, take an input from the user and search and display the contacts based on the user's input.

```
protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

try {
       Class.forName("org.postgresql.Driver");
this.conn = DriverManager.getConnection(databaseURL, username, password);
   } catch (Exception e) {
return;
   }

   String insertSQL = "SELECT * FROM contact WHERE name ilike ? ESCAPE '!'";

   out.print("<h2>Contact List</h2>");
   out.print("<form action='/contactbook/ReadContact' method='get'>");
   out.print("Search Contact: <input type='text' name='q'>");
   out.print("<input type='submit' value='Search Contact'></form>");

try {
       preparedStmt = conn.prepareStatement(insertSQL);
       String query = "%" + request.getParameter("q") + "%";

if(query != null) {
           preparedStmt.setString(1, query);
           ResultSet rs = preparedStmt.executeQuery();

           out.print("<table border='1' style='width:100%'><tr>");
           out.print("<th>Name</th><th>Email</th>");
           out.print("<th>Phone</th></tr>");
```

```
while(rs.next()) {
                out.print("<tr><td>"+rs.getString("name")+"</td>");
                out.print("<td>"+rs.getString("email")+" </td>");
                out.print("<td>"+rs.getString("phone")+"</td></tr>");
            }
            out.print("</table>");

            preparedStmt.close();
            conn.close();
        }
    } catch (SQLException e) {
        out.println("Error Occured" + e);
    }
    out.close();
}
```

Just change the *doGet* method to allow a user to filter on the name field.

2.6 UPDATE OR DELETE RECORDS

The delete or update operation needs an identification to the record on which the operation is being executed, usually developer choose the primary key as an auto

increment number field which is use to identify unique record. The id of record can be guessed easily and the user can perform the update or delete operation just by accessing an URL as below.

```
http://localhost:8080/contactbook/DeleteContact?id=29
```

You should secure those sensitive servlets, so only valid user can access such servlets. There are three ways to make it secure, it is advisable to implement the best suitable approach in your java web application.

First Approach

The first approach to secure sensitive urls, check for the user's validity on access of such restricted urls. This approach is commonly implemented by all the web developers, we should check for the current session, if valid user found in session allow access to such urls else redirect user to login page. So, each time we can check the session for a valid user before executing the critical operation.

```
HttpSession session = request.getSession(false);
if(session.getAttribute("userObj") == null) {
    RequestDispatcher rd = request.getRequestDispatcher("/Login.html");
    rd.forward(request, response);
}
```

Second Approach

The second approach is to create a urlsafe key for each record based on a unique key field. Add new column in the table and then change the code to generate the values for urlsafe column.

```
ALTER TABLE contact ADD COLUMN urlsafe VARCHAR(100);
```

Let's modify our *SaveContact.java* servlet to create the urlsafe key based on the unique field email.

```
protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

try {
      Class.forName("org.postgresql.Driver");
this.conn = DriverManager.getConnection(databaseURL, username, password);
   } catch (Exception e) {
return;
   }

   String insertSQL = "INSERT INTO contact (name, phone, email, urlsafe) VALUES
(?, ?, ?, md5(?))";
try {
      preparedStmt = conn.prepareStatement(insertSQL);
      preparedStmt.setString(1, request.getParameter("name"));
      preparedStmt.setString(2, request.getParameter("phone"));
      preparedStmt.setString(3, request.getParameter("email"));
      preparedStmt.setString(4, request.getParameter("email"));

      preparedStmt.execute();

      out.println("Record created successfully");
      preparedStmt.close();
      conn.close();
   } catch (SQLException e) {
      out.println("Error Occured" + e);
```

```
    }
    out.close();
}
```

We have added a new field named urlsafe which can be generated by PostgreSQL based on the unique value, so now we can identify each record uniquely in the database. Your data will be looking as below.

```
contactbook=# select * from contact;
 name       |         email          |   phone    |              urlsafe
----------------+------------------------+------------+----------------------------------
 Ajay Kumar     | ajay@gmail.com         | 9898098981 | 3ba1708d4d427814c9fa1b5a56675bee
 Nikunj Jani    | nikunjjani@gmail.com   | 9898798985 | b6a6c1a62a09c42a1325ffda1f8c91bc
 Harshad Modi   | harshad@gmail.com      | 9897187928 | f5e2b761c60508a8d9ff30eadf272879
 Anjana Raval   | anjana@gmail.com       | 9897187922 | 377935861f33a7c1d296ddf15713c0f2
 Deepak Raval   | deepak@gmail.com       | 9897187924 | 5ae4927580af7bac3c6adf451158e0e5
 Pramukh Suthar | pramukh@gmail.com      | 9897287923 | dfdc36f348b15558a1bc912deeca26cb
 Mantavya Gajjar | mantavyagajjar@gmail.com | 9898798982 | 11a2db4be94e348f34ecdb906cee25d2
(7 rows)
```

Now, it will be difficult for the userto make a guess for any record to delete when you use the urlsafe key as a record key in the URL parameter.

```
http://localhost:8080/contactbook/DeleteContact?id=5ae4927580af7bac3c6adf451158e0e5
```

Third Approach

The third approach does not delete any record in the database, instead of adding a new field named active, by default when a record is being created in the system set active to true if you want to delete any record set active to false. So by default when you perform read or search operation add the default condition such as `WHERE active='t'`.

```
stable=# select name, email, urlsafe, active from contact;
name       |          email          |              urlsafe             | active
---------------+-----------------------+----------------------------------+--------
Ajay Kumar      | ajay@gmail.com          | 3ba1708d4d427814c9fa1b5a56675bee | t
 Nikunj Jani    | nikunjjani@gmail.com    | b6a6c1a62a09c42a1325ffda1f8c91bc | t
 Harshad Modi   | harshad@gmail.com       | f5e2b761c60508a8d9ff30eadf272879 | t
 Deepak Raval   | deepak@gmail.com        | 5ae4927580af7bac3c6adf451158e0e5 | t
 Mantavya Gajjar | mantavyagajjar@gmail.com | 11a2db4be94e348f34ecdb906cee25d2 | t
 Pramukh Suthar  | pramukh@gmail.com       | dfdc36f348b15558a1bc912deeca26cb | f
 Anjana Raval   | anjana@gmail.com        | 377935861f33a7c1d296ddf15713c0f2 | f
(7 rows)
```

2.7 DATABASE CONNECTION POOLING

Database Connection Pooling is a great technique used by a lot of application servers to optimize performance. Database Connection creation is a costly task thus it impacts the performance of the application. Hence a application server creates a database connection pool which are pre-initiated database connections that can be leveraged to increase performance.

Connection pool is a set of opened connection to the same database, those are created when application server start, so we can save the time to load the JDBC database driver into memory and established the connection to the database, when user need a connection it can be assigned from the pool and when database operation completed the connection can be taken back and add to the pool, this is the biggest advantages of using connection pool in an enterprise web application.

Apache Tomcat also provides a way of creating database Connection Pool. Let us see an example to implement database Connection Pooling in the Apache Tomcat server. We will improve our contact book web application to use the connection pool to get the database connection from database connection pool and fetch the data using a query.

Apache Tomcat allows an application to define the resource used by the web application in *context.xml* (from Tomcat 5.x version onwards). We have to create a file *context.xml* under META-INF directory.

Additional Libraries

You may need to add some additional libraries to the CLASSPATH to compile the servlet. In my example I have added below listed libraries to the CLASSPATH, they are available in the Tomcat lib directory.

```
apache-tomcat-9.0.17/lib/tomcat-jni.jar
apache-tomcat-9.0.17/lib/tomcat-jdbc.jar
```

META-INF/context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>

<!-- Specify a JDBC datasource -->
<Resource name="jdbc/contactbook" auth="Container"
    type="javax.sql.DataSource" username="mantavyagajjar" password="******"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://localhost:5432/contactbook" maxIdle="4" maxTotal="8"/>

</Context>
```

In the above code snippet, we have specified a database connection pool. The name of the resource is `jdbc/contactbook`. We will use this name in our application to get the data connection.

Modify the Servlet Program

Let's modify the servlet to use the connection from the connection pool instead of open and close connection on each user request. Now connection related activities will be managed by the connection pool.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.sql.* ;
import javax.sql.*;
import javax.naming.*;

@WebServlet(name = "Contact", urlPatterns = {"/ReadContact"})
publicclassReadContactextendsHttpServlet {

private DataSource dataSource;
private Connection connection;
private PreparedStatement statement;

@Override
publicvoidinit(ServletConfig config) throws ServletException {
super.init(config);
try{
        Context initContext = new InitialContext();
        Context envContext = (Context) initContext.lookup("java:/comp/env");
        dataSource = (DataSource) envContext.lookup("jdbc/contactbook");
    } catch (NamingException e) {

    }
  }
```

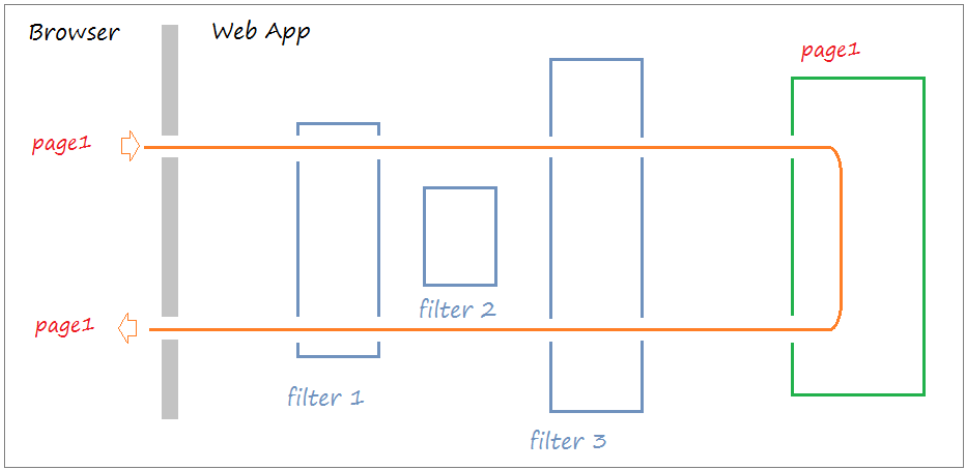
```
protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String insertSQL = "SELECT * FROM contact WHERE name ilike ? ESCAPE '!'";

        out.print("<h2>Contact List</h2>");
        out.print("<form action='/contactbook/ReadContact' method='get'>");
        out.print("Search Contact: <input type='text' name='q'>");
        out.print("<input type='submit' value='Search Contact'></form>");

try {
            connection = dataSource.getConnection();
            statement = connection.prepareStatement(insertSQL);

            String query = "%" + request.getParameter("q") + "%";

if(query != null) {
                statement.setString(1, query);
                ResultSet rs = statement.executeQuery();

                out.print("<table border='1' style='width:100%'><tr>");
                out.print("<th>Name</th><th>Email</th>");
                out.print("<th>Phone</th></tr>");

while(rs.next()) {
                    out.print("<tr><td>"+rs.getString("name")+"</td>");
                    out.print("<td>"+rs.getString("email")+" </td>");
                    out.print("<td>"+rs.getString("phone")+"</td></tr>");
```

```
            }
            out.print("</table>");
        }
    } catch (SQLException e) {
        out.println("Error Occured" + e);
    }
  }
}
```

The output remains the same, there is no change in the execution of the queries, if you compare the code,reduced a lot as connection is managed by the application server.

2.8 RESTRICT USER-ACCESS TO SERVLET

Normally, when a user requests a servlet or web page, a request is sent to the application server, the application server allows access to that requested servlet or web page if exist on the server, we have to change that mechanism so it will have to pass through the filter before reaching the servlet or web page required, like the illustration below:

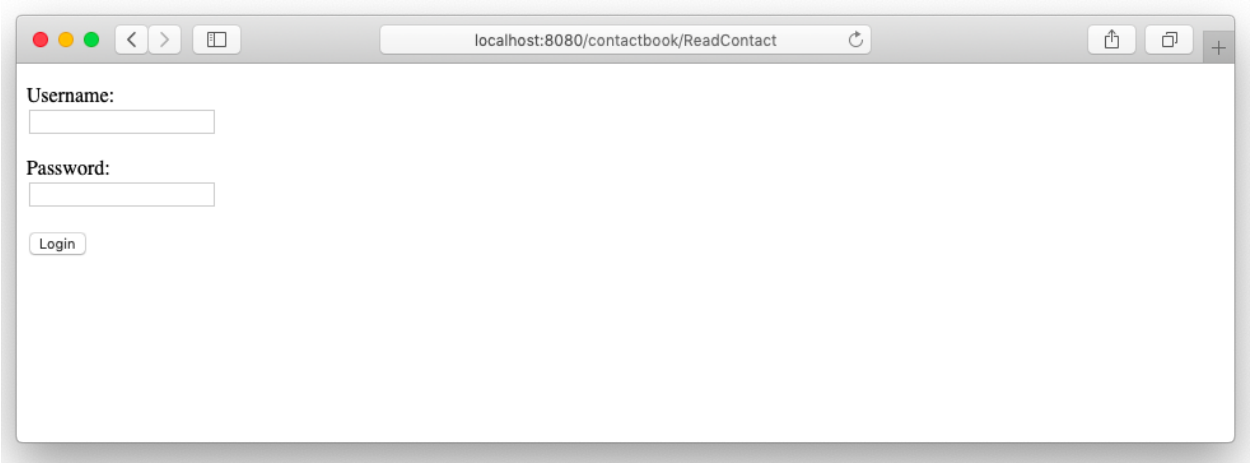


However, there are situations where the user's request does not pass all Filters, as a user does not have enough access to such resource and due to that filter redirects users to another page.

Let's implement the filter that verifies the current user, if the user is not valid then redirect to a login page or allow access on the page requested for valid users.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebFilter("/ReadContact")
publicclassReadContactFilterimplementsFilter {

publicvoiddoFilter(ServletRequest request, ServletResponse response, FilterChain
chain)
throws IOException, ServletException {

        HttpSession session = null;
        HttpServletRequest httpRrequest = (HttpServletRequest) request;

        session = httpRrequest.getSession(false);

if(session.getAttribute("user") == null) {
            session.setAttribute("returnURL", httpRrequest.getServletPath());
            RequestDispatcher rd = request.getRequestDispatcher("/login.jsp");
            rd.forward(request, response);
        } else {
            chain.doFilter(request, response);
        }
    }
}
```

We have implemented the Filter which will be called when user request for /ReadContact url from the browser, a doFilter method will be called with *ServletRequest, ServletResponse, FilterChain* objects. The filter will check the session for a valid user object, if valid user object found then allow the user to access the /ReadContact servlet else redirect to the *Login* page.



Enter the user and password, and click on the Login button we will be redirected to *Login* Servlet to verify the user, the session will be created if user is valid. Filter will check for the session again and grant access on the requested resource.

Until a valid user found in session, the user will be able to access /ReadContact servlet.

Unit 3: Basics of Java Server Pages

3

Unit Structure

3.1 LEARNING OBJECTIVE

After going through this unit, you should be able to:

- understand the need for JSP;
- understand the functioning of JSP;
- understand the relation of applets and servlets with JSP;
- know about various elements of JSP;
- explain various scripting elements of JSP;
- explain various implicit objects of JSP, and
- understand the concept of custom tags and the process of creating custom tag libraries in JSP.

3.2 INTRODUCTION TO JSP

Java Server Pages is a technology used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL. A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development.

JSP is a technology based on the Servlet, Servlet Container or Application Server convert all the JSP pages to Servlet, the Servlet will be executed by the servlet container finally. Java Server Pages are mostly used to prepare an application user interface than Servlet that generates the user interface. We can use all the objects such as *HttpServletRequest* or *HttpServletResponse* which are available to Servlet.

Java Server Pages executes much faster compared to other dynamic languages. It is much better than the Common Gateway Interface (CGI). Java server pages are built over Java Servlets API. Hence, JSP Page has access to all Java Servlet APIs, even it has access to JNDI (Java Naming Directory Interface), JDBC and other java libraries. JSP is used in MVC architecture as a view layer. The MVC application architecture can be achieved using JSP and Servlet technologies, Java Beans are use to create a model, Servlet used to create a controllers and JSP pages are used to create a view layer.

There are various advantages of using Java Server Pages, some of them are listed below:

- As it is built on Java technology, hence it is platform independent and not dependent on any specific operating system.
- JSP page converted to Java Servlet, hence you can access all the Java objects in JSP page which can be used in Servlet.
- JSP Scripting elements enables you to mix the Java and HTML code together in JSP file.
- Using JSP Custom Tag Library feature code can be simplified and readable format, The JSP Taglib Directive executed by the Servlet Container or Web server and converted into the equivalent HTML code.

3.3 JSP SCRIPTING ELEMENTS

All the JSP files converted to Servlet before it executed by the Servlet Container or Web Server, The code written inside the JSP scripting elements will be added to the Servlet. Using scripting elements we will be able to write the Java and HTML code in a single file.

There are three forms of writing the elements in JSP file:

- JSP Declaration
- JSP Scriptlet
- JSP Expression

Let's see the usage of those elements.

JSP Declaration

A declaration tag is a piece of Java code for declaring variables, methods, and classes. If we declare a variable or method inside declaration tag it means that the declaration is made inside the servlet class but outside the service method.

We can declare any variables inside the declaration block such as static member, an instance variable, an integer or a string variable, we can also declared any other Java object inside the declaration tag.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Declaration Tag Example</title>
</head>
<body>
<%! int count =10; %>
<% out.println("The Number is " + count); %>
</body>
</html>
```

The variable which is declared in the declaration tag is initialized and printed as output.

JSP Scriptlet

Scriptlet tag allows writing Java code into a JSP file. The JSP file converted into Servlet by Servlet Container, all the statements written within Screplet tags are encapsulated in *_jspservice()* method of Servlet, finally Servlet will be compiled and executed by the Servlet Container. For each request of the client, service method of the JSP gets invoked hence the code inside the Scriptlet executes for every request. A Scriptlet contains java code that is executed every time JSP is invoked.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Scriplet Example</title>
</head>
<body>
<% int number_ond=10;
 int number_two=40;
 int numbers = number_ond + number_two;
 out.println("Scriplet Number is " +numbers);
 %>
</body>
</html>
```

In the Scriptlet tags, we have declared two variables number_one and number_two. Third va riable numberswill b e declared andinitialized with t he su mmation of number_one and number_two.

JSP Expression

Expression tag eva luates the expression pl aced inside the block. It can access the data stored i n any variables. It a llows for creating expressions l ike ar ithmetic and logical, the final result will be en capsulated into the println statement, hens the final result will be displayed on the webpage.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Expression Example</title>
```

```
</head>
<body>
<% int number_one=10; int number_two=10; int number_three=20; %>
<% out.println("The expression number is "); %>
<%= number_one * number_two * number_three %>
</body>
</html>
```

We have used anexpression tag, where we have written an arithmetic expression to multiply three numbers i.e. number_one and number_two and number_three.

JSP Comments

The JSP comments are the statement or block of statements, converted to the Java comments by the JSP container during the conversion form JSP file to Java Servlet. The HTML comments are encapsulated into the println function and pushed to the browser as HTML file.

Comment in JSP	Comment in HTML
<% -- JSP Comments %>	<!-- HTML Comment -->

Comments are used to write a docu mentation within the code or we can ignore a part of the code by adding comment.

3.4 JSP DIRECTIVES

JSP directives are the messages to JSP container. They provide global information about an entire JSP page. JSP directives are used to give special instruction to a container for translation of JSP to Servlet code. During the translation phase of JSP Lifecycle, a JSP file is converted into the Java Servlet, will complied to Java Class file. JSP Directives give instructions to the Servlet Container on how to transfer the

code i nto the S ervlet dur ing t he t ranslation phas e. Directives ca n have m any attributes separated by a sp ace i nform of key-value pairs. JSP Directive ca n b e described written as<%@ attribute="" %>.

There are three types of directives:

- Page directive
- Include directive
- Taglib directive

Let's see each one of them in detail with an example:

JSP Page directive

It pr ovides attributes that ar e applied t o t he ent ire JSP p age. I t def ines page-dependent attributes, su ch as scr ipting l anguage, er ror page, and buffering requirements. It is used to provide instructions to a Servlet Container that creates the Servlet related to the current JSP page.

Following are the list of attributes associated with page directive:

1. Language
2. Extends
3. Import
4. contentType
5. info
6. session
7. isThreadSafe
8. autoflush
9. buffer
10. IsErrorPage
11. pageEncoding
12. errorPage
13. isELIgonored

Language

At the beginning of JSP file, a page directives should be declared as below.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
```

Import

To perform a specific operation if you need support from external libraries, those libraries can can be import in JSP page using an import attribute.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   import="java.util.Date" pageEncoding="ISO-8859-1"%>
```

Extends

As every JSP page is converted to Servlet java class before execution, you can inherit another java class using extends attribute.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
          pageEncoding="ISO-8859-1"%>
<%@ page extends="hello.LoginPage" %>
```

In the above example, JSP page extends an existing servlet LoginPage which is declared in hello package.

Info

It defines a string which can be accessed by *getServletInfo()* method. This attribute is used to set the servlet's description.

```
<%@ page info="HelloWorld Example" pageEncoding="ISO-8859-1"%>
```

Session

JSP page creates a session automatically for all pages by default. Sometimes we don't need a session to be created automatically in JSP page, we can set Session attribute to false. The default value of the session attribute is true, so the session is created automatically. When it is set to false, then we can indicate the compiler to not create the session by default.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    session="false"%>
```

isThreadSafe

When isThreadSafe is set to true, Servlet Container creates multiple objects for the same JSP file when requested by multiple clients. Each client is served with a separate *_jspService()* method. When isThreadSafe is set to false, indicates the container to create one Servlet object for each client requesting the same JSP. Multiple clients will have multiple Servlet objects created by the container to honor all the clients.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    isThreadSafe="true"%>
```

AutoFlush

This attribute specifies that the buffered output should be flush automatically or not, the default value of that attribute is true. If the value is set to false the buffer will not be flush automatically, when the buffer gets full we may get an exception.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    autoFlush="false"%>
```

Buffer

Using this attribute the output response object may be buffered. We can define the size of the buffer to be done using this attribute, the default buffer size is 8KB. The bufferindicatesa size of the buffer used by the servlet to write the output to the buffer before writing to the response object.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    buffer="16KB"%>
```

ErrorPage

This attribute is used to set an error page for the JSP page. When anexception occurs during the executing of JSP page, Servlet Container automatically redirects a request to the error page.

```
<%@ page language="java" contentType="text/html;" pageEncoding="ISO-8859-1"
    errorPage="errorHandler.jsp"%>
```

isErrorPage

It indicates that JSP Page have the capability to receive an exception from other JSP pages. The default value is false.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    isErrorPage="true"%>
```

isELIgnored

The default value is set to true, means you can evaluate an expression such as ${2 * 4 + 3 * 4} in JSP page. You can deactivate by setting values to false for any specific JSP file.

```
<%@ page language="java" contentType="text/html;" pageEncoding="ISO-8859-1"
    isELIgnored="true"%>
```

JSP Include directive

JSP include directive is used to include one file into another file. This included file can be HTML, JSP, text files. It is very good features that used to break the user interface into a header, footer and content part. The filer will be included during the translation phase. Let's divide the whole page into header, footer and use them into index page using the JSP include.

header.jsp

Define the menu bar.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
<div class="container">
<a class="navbar-brand" href="#">Navbar</a>
<div class="collapse navbar-collapse" id="navbarNavAltMarkup">
<div class="navbar-nav">
<a class="nav-item nav-link active" href="#">
        Home<span class="sr-only">(current)</span>
</a>
<a class="nav-item nav-link" href="#">Features</a>
<a class="nav-item nav-link" href="#">Pricing</a>
</div>
</div>
</div>
</nav>
```

Footer.jsp

Define the sticky footer which stays bottom of the page

```
<style>
.footer {
background-color: #f5f5f5;
```

```
}
</style>

<footer class="footer mt-auto py-3">
<div class="container">
<span class="text-muted">Place sticky footer content here.</span>
</div>
</footer>
```

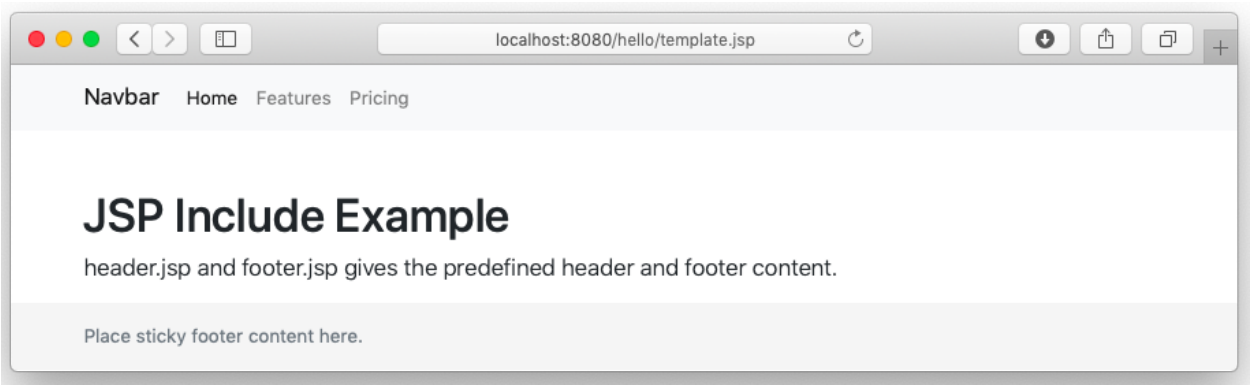
Index.jsp

Create an index page with the content and reuse the header and footer by including them into the page.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Header with Menu</title>
<link rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
    crossorigin="anonymous"/>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
    crossorigin="anonymous"/>
</head>
<body class="d-flex flex-column h-100">

<%@ include file="header.jsp" %>

<main role="main" class="flex-shrink-0">
```

```
<div class="container">
<h1 class="mt-5">JSP Include Example</h1>
<p class="lead">
        header.jsp and footer.jsp gives the predefined header and footer content.
</p>
</div>
</main>

<%@ include file="footer.jsp" %>
</body>
</html>
```

Now, open the index.jsp file in the browser you will see a beautiful page with menu bar and footer.



JSP Taglib Directive

JSP taglib directive is used to import the tag library with "taglib" as a prefix. The tag library is a set of custom tags which executed by the Servlet Container to generate the HTML output. It uses a set of custom tags, identifies the location of the library and provides means of identifying custom tags in JSP page.

Let's take an example to understand how custom tag library can help a developer to simplify the JSP code.

- Create a new web app called*hello*underwebapps directory

- Create a required directory structure, i.e. WEB-INF and WEB-INF/lib directory
- Copy taglibs-standard-impl-1.2.5.jar and taglibs-standard-spec-1.2.5.jar libraries into webapps/hello/WEB-INF/lib form webapps/examples/WEB-INF/lib directory.
- Create an index.jsp in webapps/hello/index.jsp and use the below code to test the custom tag-library.

```
<html>
<head>
<title>Tag Plugin Examples: forEach</title>
</head>
<body>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page import="java.util.Vector" %>

<h3>Iterating over a range</h3>
<c:forEach var="item" begin="1" end="10">
    ${item}
</c:forEach>

<% Vector v = new Vector();
    v.add("One"); v.add("Two"); v.add("Three"); v.add("Four");
    pageContext.setAttribute("vector", v);
  %>

<h3>Iterating over a Vector</h3>
<c:forEach items="${vector}" var="item">
    ${item}
</c:forEach>
</body>
</html>
```

The taglib is a tool used to define custom tags that are processed by the Servlet Container and translated into the HTML code as per the definition of custom tag and its method. Look at the output of the above code, the HTML page is generated.

3.5 JSP IMPLICIT OBJECTS

JSP implicit objects are created during the translation phase automatically added to the Servlet. When writing a JSP page we do not have to create those objects explicitly as they are created by the Servlet Container. There are 9 implicit objects can be accessed directly without explicit declaration in any JSP file:

1. out
2. request
3. response
4. config
5. application
6. session
7. pageContext
8. page
9. exception

Let’s see the usage of each object in detail

out

Out is one of the implicit objects used to write data to buffer and send output to the client in response. *Out* object allows us to access the servlet output stream, *out* is an instance of *javax.servlet.jsp.jspWriter class*.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Implicit Objects - out Example</title>
</head>
<body>
<% int number_one=10; int number_two=20;
      out.println("number_one is " +number_one);
      out.println("number_two is "+number_two);
    %>
</body>
</html>
```

Request

The request object is an instance of *java.servlet.http.HttpServlet* class. The request is one of the arguments of service method, for every user request Servlet Container create an instance of *java.servlet.http.HttpServlet* class and passed to *_jspservice(request, response)* method. It will be used to get information like user inputs and request header values. We can get the list of parameters using *getParameter()* method to access the user inputs pass to the server.

Index.html - the HTML form takes username and password from user and passes to hello.jsp f ile. The he llo.jsp that get the r eads the us ername an d passw ord f rom request object and display a value on the hello.jsp page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>User Input Form</title>
</head>
<body>
<form action="/hello/ProcessRequest" method="post">
<p>Enter your name: <input type="text" name="username"></p>
<input type="submit" value="login">
</form>
</body>
</html>
```

Hello.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Request Object - Example</title>
</head>
<body>
<%
    String username = request.getParameters('username');
```

```
    out.println("Welcome " + username);
   %>
</body>
</html>
```

Response

The r esponse is an i nstance of type *HttpServletResponse* interface. The containercreates a request object and pa ss it to *_jspservice(request, response)* method as a parameter. It represents the response that is given to the client.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Response Object - Example</title>
</head>
<body>
<%
    String username = request.getParameters('username');
    response.addCookie(new Cookie("username",username));
   %>
</body>
</html>
```

Config

The c onfig i s of t he type *java.servlet.servletConfig* interface, It is created by the Servlet C ontainer f or each JSP page, I t r eads the initialization par ameter f rom web.xml and passes to Servlet or JSP page.

Application

The application object is an instance of *javax.servlet.ServletContext* interface, the instance is created by the Servlet Container, loads the attributes defined in the web.xml deployment descriptor file. The *javax.servlet.ServletContext* object contains a set of methods which are used to get and set the attributes which are loaded in Servlet Container.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Application Object - Example</title>
</head>
<body>
<% out.println(application.getContextPath()); %>
</body>
</html>
```

This code will print the application root path, i.e. /hello

Session

The session object is holding "httpsession" object. The session object is used to get, set and remove attributes to session scope and also used to get session information.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Session Object - Example</title>
</head>
<body>
<% session.setAttribute("user","ajay@gmail.com"); %>
<a href="/help/current-session.jsp">Click to see current login user</a>
</body>
</html>
```

The above program will set the attribute "*user*" to the session, the below program will read the same "*user*" attribute form the session.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Session Object - Example</title>
</head>
<body>
<%
      String name = (String)session.getAttribute("user");
      out.println("User Name is " + name);
    %>
</body>
</html>
```

pageContext

In JSP, *pageContext* is an implicit object of type *javax.servlet.jsp.PageContext* class. The *pageContext* object can be used to set, get or remove the attribute from one of the following scopes:

- page
- request
- session
- application

In JSP, the page is the default scope, if you do not pass the scope.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Session Object - Example</title>
</head>
<body>
<%
     pageContext.setAttribute("student", "Vijay Patel", pageContext.PAGE_SCOPE);
     String name = (String) pageContext.getAttribute("student");
     out.println("student name is " +name);
   %>
</body>
</html>
```

Student attribute will not be accessible to another page in this example.

Page

The page is an implicit object holds the currently executed servlet object for the corresponding JSP. Acts as this object for current JSP page.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Session Object - Example</title>
</head>
<body>
<%
     String pageName = page.toString();
     out.println("Page Name is " +pageName);
   %>
</body>
</html>
```

Print the string representation of the current jsp page.

Exception

The exception object represents all errors and exceptions. The exception implicit object is of type *java.langThrowable*. You can access the exception object on a page that you declare to be an error page using the isErrorPage attribute of the page directive.

The exception object is created only if the JSP uses the page directive to set *isErrorPage* set to true. When a JSP generates an error and forwards that error to

the error page, the container sets the JSP exception object of the error page to the generated error.

3.6 JSP EXPRESSION LANGUAGE

Expression Language (EL) is a mechanism that simplifies the accessibility of data stored in the Java bean component or any other objects like request, session, and application. There are several implicit objects, operators and reserved words in Expression Language. The JSP Expression Language supports operators and control-flow statements, There are many operators supported in JSP such as arithmetic and logical operators to perform an expression. The Expression Language was introduced in JSP 2.0.

JSP Syntax of Expression Language (EL)

The expression written within the curly braceswill be evaluated at runtime and sent to the output stream.The expression should be a valid expression and it can be mixed with a html text and can be combined with other expressions to form larger expression. To get a better idea, on how expression works in JSP, let's go through below example.

In this example, we will write an arithmetic expression using plus (+) operator to add two numbers i.e. (1+2) and get the output.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression Language - Example</title>
</head>
<body>
```

```
<a>Expression is:</a>
   ${1+2};
</body>
</html>
```

You will see “Expression is 3” line in your browser as an output.

JSP Flow Control Statements

JSP is based on the Java Language, hens we can use all the flow and control statements which are used in Java such as if - else, switch, for or while. We can use all the APIs and building blocks of Java programming language in JSP programming including control flow statements. There are two types of flow control statements described below;

Decision-Making Statements: Decision-making statement in JSP is based on whether the resultfor a condition is true or false. The statement will behave according to the result of a condition. There are two types of decision-making statements described below:

- If – else
- Switch

JSP If-else

"If-else" statement is basic of control flow statement, and it tells the program to execute the certain section of code only if the particular conditions evaluates to true. The if-else statement can evaluate multiple conditions, based on the result the next set of statements will be executed, If the first condition is true then " if block" is executed and if the conditions is false then "else block" is executed.

```
if (test condition) {
   //Block of statements
}
```

```
else {
   //Block of statements
}
```

In JSP page if-else can be written as below.

```
<body>
<%! int month=5; %>
<% if(month==2){ %>
<p>Its February</p>
<% }else{ %>
<p>Any month other than February</p>
<%} %>
</body>
```

JSP Switch

The body of the switch statement is called a "switch block". The switch case is used to check the number of possible execution paths. A switch can be used with byte, short, char, and int primitive data types. The switch statement contain more than one cases, we can also include a default case as it is optional. Consider the below JSP program, it declares an int named weekday whose value represents a day of week(1-7). The code displays the name of the day, based on the value of day, using the switch statement.

```
<body>
<%! int weekday=2; String weekday="Saturday" %>
<%
 switch(weekday) {
   case 0:
     weekday="Sunday";
     break;
```

```
   case 1:
     weekday="Monday";
     break;
   case 2:
     weekday="Tuesday";
     break;
   case 3:
     weekday="wednesday";
     break;
   case 4:
     weekday="Thursday";
     break;
   case 5:
     weekday="Friday";
     break;
  }
  out.println(weekday);
%>
</body>
```

JSP For loop

It is used for iterating over the list of elements for a certain condition, and it has three parameters.

- Variable counter is initialized
- Condition till the loop has to be executed
- Counter has to be incremented

Go through the below program, i is the counter variable, the loop will be executes 5 times based on the conditions, and counter will be increased by 1 on each iteration.

```
<body>
```

```
<%! int num=5; %>
<%
  out.println("Numbers are:");
  for(int i=0;i<num;i++){
    out.println(i);
  }
%>
</body>
```

We have for loop which iterates till counter (i.e. int i is counter)is less than 5, the output will be “Numbers are: 0 1 2 3 4”.

JSP While loop

It is used to executes the code block based on the conditions, while loop has only one parameter (i.e. condition), the loop will be executed until the condition is true.

```
<body>
<%! int day=2; int i=1; %>
<%
  while(day>=i){
    if(day==i){
        out.println("Its Monday");
        break;
    }
    i++;
  }
%>
</body>
```

JSP Operators

JSP O perators supports most of ar ithmetic and l ogical oper ators which ar e supported by java within expression language (Expression Language) tags.

Frequently used operators are mentioned below:

.	Access a bean property or Map entry
[]	Access an array or List element
( )	Group, a subexpression to change the evaluation order
+	Addition
-	Subtraction or negation of a value
*	Multiplication
/ or div	Division
% or mod	Modulo (remainder)
== or eq	Test for equality
!= or ne	Test for inequality
< or lt	Test for less than
> or gt	Test for greater than
<= or le	Test for less than or equal
>= or ge	Test for greater than or equal

&& or and	Test for logical AND
|| or or	Test for logical OR
! or not	Unary Boolean complement
Empty	Test for empty variable values

JSP Expression Language (EL) makes it easy to access the application for the data stored in the JavaBeans components. It also allows creating expressions which are both ar ithmetic and l ogical. Within E L t ags we can use i ntegers, f loating poi nt numbers, strings, and Boolean values. In JSP we can also use loops and decision-making statements using Expression Language tags

3.7 JSP ACTION TAGS

Actions are used to controlling behavior of Servlet Engine. JSP actions are written in XMLlanguage. JSP pr ovides a bunch of st andard Action Tags that we can use for specific tasks such as working w ith j ava bean obj ects, i ncluding ot her r esources, forward the request to another resource.

There are 11 types of action names as following:

1. jsp:useBean	5. jsp:forward	9. jsp:text
2. jsp:setProperty	6. jsp:plugin	10. jsp:param
3. jsp:getProperty	7. jsp:attribute	11. jsp:attribute
4. jsp:include	8. jsp:body	12. jsp:output

Jsp:useBean

jsp:useBean action name is used when we want to set or get the multiple values of object in the JSP page. With this tag, we can easily invoke a bean , get and se t the attributes of that bean.

Let's take an example to understand how user input values from HTML form will be set in Java Bean using *jsp:useBean, jsp:setProperty and jsp:getProperty.* We will create below a list of files in our example.

- Contact.java - Java Bean, declare a Contact class
- Index.html - HTML form which takes input from a user and passes to createContact.jsp when a user submits the form.
- createContact.jsp - A JSP file create an instance of Contact Bean, set to the values received form Index.html.

The name of the object variables declared in Java bean (i.e. name, email, and phone) and name of the fields declared in HTML form are same. Servlet Container automatically maps the received parameters with the properties of Java Beans using set methods (i.e. setName, setEmail, setPhone) in Java Beans.

Contact.java

```
package com.company;

publicclassContact {

private String name;
private String email;
private String phone;

public String getName() {
return name;
   }

publicvoidsetName(String name) {
this.name = name;
   }
```

```
public String getEmail() {
return email;
   }

publicvoidsetEmail(String email) {
this.email = email;
   }

public String getPhone() {
return phone;
   }

publicvoidsetPhone(String phone) {
this.phone = phone;
   }
}
```

Index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Java Beans Example - Create Contact</title>
</head>
<body>
<form method="get" action="/bean-example/createContact.jsp">
     Name: <input type="text" name="name"><br>
     Email: <input type="text" name="email"><br>
     Phone: <input type="text" name="phone"><br>
<input type="submit">
</form>
```

```
</body>
</html>
```

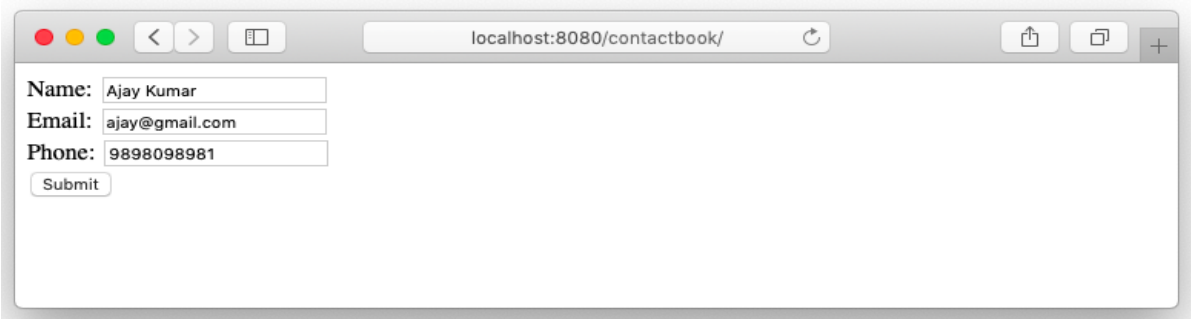
createContact.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<%@ page import="com.company.Contact" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Java Bean Example</title>
</head>

<body>
<jsp:useBean id="employee" class="com.company.Contact" scope="session">
<jsp:setProperty name="employee" property="*"/>

<p>Employee Name: <jsp:getProperty name="employee" property="name"/></p>
<p>Email: <jsp:getProperty name="employee" property = "email"/></p>
<p>Email: <jsp:getProperty name="employee" property = "phone"/></p>
</jsp:useBean>
</body>
</html>
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
    version="4.0" metadata-complete="false">

<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>
```

Now, open URLhttp://localhost:8080/contactbook you w ill se e a form to cr eate a contact.



Click on Submit but ton, nam e, em ail, and phon e w ill be passed t o a JSP page, Contact Bean will be invoked and all the attributes set using setProperty methods.

Jsp:include

It is used to insert output of oneJSP file into another JSP file, just like include directive. It is added during the request processing phase.

Index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
</head>
<body>
<jsp:include page="index.html" flush="true" />
</body>
</html>
```

It will display the HTML form to create a contact form as below.

Jsp:forward

It is used to forward the implicit request object to another JSP or any static page. Here the request can be forwarded with parameters or without parameters.

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
</head>
<body>
<jsp:forward page="index.html" />
</body>
</html>
```

When we access the index.jsp, it will be redirected to index.html.

Jsp:plugin

It is used to add Java components into JSP, Java components can be either an applet or bean. It detects the browser and adds <object> or <embed> tags to the response.

```
<jsp:plugin type="applet/bean" code="objectcode" codebase="objectcodebase">
```

The type attribute specifies either an object or a bean value, code attribute specifies class name of applet or bean, the codebase contains the package name for the Java Bean or URL that contains Applet.

Jsp:param

This is a child object of the jsp:plugin object described above, jsp:paramis used to pass additional values to the Java Bean or Applet.

```
<jsp:plugin type="bean" code="Student.class" codebase="com.book">
<jsp:params>
<jsp:param name="name" value="Ajay Kumar" />
<jsp:param name="email" value="ajay@gmail.com" />
<jsp:param name="email" value="9898098981" />
</jsp:params>
</jsp:plugin>
```

Jsp:text

It is used to template text in JSP pages. Its body does not contain any other elements, and it contains only text and EL expressions.

```
<jsp:text>Template text</jsp:text>
```

Template text refers to only text which can be any generic text which needs to be printed on JSP or an EL expression.

Jsp:output

The jsp:output element specifies the XML declaration or the document type declaration in the request output of the JSP document.

The XML declaration and document type declaration that are declared by the jsp:output element are not interpreted by the JSP container. Instead, the container simply directs them to the request output.

To illustrate this, let's take below example:

```
<jsp:output doctype-root-element="books" doctype-system="books.dtd" />
```

The resulting output is:

```
<!DOCTYPE books SYSTEM "books.dtd">
```

3.8 COOKIES IN JSP

Cookies are text data stored on the client computer and are used to store information. A JSP can access to the cookies through the request method *request.getCookies()* which returns an array of Cookie objects and set the cookie through *response.addCookie(cookie)* method.

Adding Cookie to Response

If the browser is configured to store cookies, it will keep those cookies until the expiry date, Itcan be set-up using the following steps:

- Creating the cookie object
- Setting the maximum age
- Sending the cookies in HTTP response headers

Please refer the below code, it is used to add *name* and *email* fields in the cookie.

```
<%
  Cookie name = new Cookie("name", request.getParameter("name"));
  Cookie email = new Cookie("email", request.getParameter("email"));

  name.setMaxAge(60*60*10);
  email.setMaxAge(60*60*10);

  response.addCookie(name);
  response.addCookie(email);
%>
```

3.9 MVC ARCHITECTURE IN JSP

MVC is an application architecture that separates business logic, presentation and data. In MVC,M stands for Model, V stands for View, C stands for the controller.

MVC is a systematic way to use the application where the flow starts from the view layer, where the request is raised and processed in controller layer and sent to model layer to insert data and get back the success or failure message.

Model Layer:

This is the data layer which consists of the business logic of the system. It contains all the data of an application, It also represents the state of an application. It consists of classes which fetches the data from the database on users request. The controller connects with model and fetches the data and sends to the view layer. The model connects with the database as well and stores the data into a database.

View Layer:

This is a presentation layer. It consists of HTML, JSP, etc. into it. It normally presents the UI of the application. It is used to display the data which is fetched from the controller which in turn fetching data from model layer classes. This view layer shows the data on the user interface of the application.

Controller Layer:

It acts as an interface between View and Model. It intercepts all the requests which are coming from the view layer. It receives the requests from the view layer and processes the requests and does the necessary validation for the request. This request is further sent to the model layer for data processing, and once the request is processed, it sends back to the controller with the required information and displayed accordingly by the view.

Example

Let’s take an example to understand how mode, view, and the controller can be developed using HTML, servlet and JSP page. Develop a login form which takes user and password as input and to a servlet, servlet verifies the user and password and depending on the result choose which JSP page (welcome or error) to display on the user's browser.

User.java - a model class which defines the data and method to process the data

```
package com.book;

publicclassUser {

private String username;
private String password;

publicUser(String username, String password) {
this.username = username;
this.password = password;
    }

public String getUsername() {
return username;
    }

publicvoidsetUsername(String username) {
this.username = username;
    }

public String getPassword() {
return password;
    }

publicvoidsetPassword(String password) {
this.password = password;
    }

public Boolean login() {
//check in the database
//verify the validity of the user and password
```

```
returntrue;
    }
}
```

Login.java - servlet act as a controller, which actually takes the input from the user (login.html), initiate the model and verify the login if login valid redirect to index.jsp else error.jsp.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import com.book.*;

@WebServlet(name = "Login", urlPatterns = {"/Index"})
publicclassLoginextendsHttpServlet {

publicLogin() {
super();
    }

protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        String name = request.getParameter("username");
        String password = request.getParameter("password");

        User user = new User(name, password);

        HttpSession session = null;
```

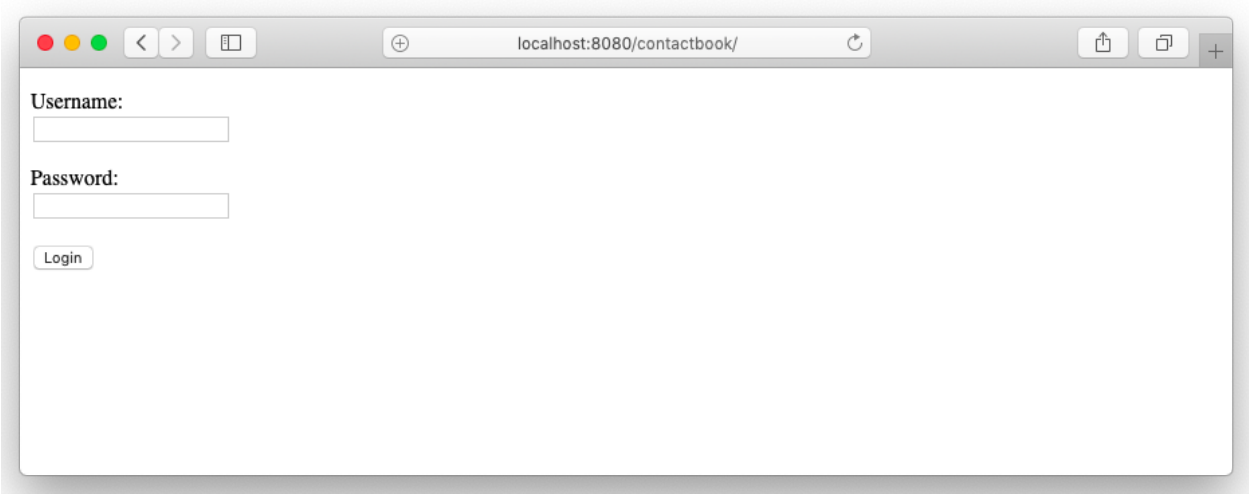
```
        RequestDispatcher rd = null;

if (user.login()) {
            session = request.getSession();
            session.setAttribute("user", user);

            rd = request.getRequestDispatcher("/index.jsp");
            rd.forward(request, response);
        } else {
            rd = request.getRequestDispatcher("/error.jsp");
            rd.forward(request, response);
        }
    }
}
```

Index.jsp - A view which is called from the controller and displayed home page after the login.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ page import="com.book.*"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
</head>
<body>
<%
```

```
    User user = (User) session.getAttribute("user");
    out.println("<h2>Hello " + user.getUsername() +"</h2>");
  %>
<jsp:include page="index.html" />
</body>
</html>
```

Output



Enter a username and password and click on the Login button, the value will be transferred to the Login servlet and redirect to Index.jsp if the user is valid.

Unit 4: JDBC with JSP 4

Unit Structure

4.1 LEARNING OBJECTIVE

After going through this unit, you should be able to know:

- Understand how to establish connection to database in JSP
- Understand how to fetch the data from database and display it on the JSP page
- Understand how you can get powered the java standard tag libraries

4.2 INTRODUCTION TO JSP

We have gone through Chapter 3: Introduction JSP and JSP Basics , we learn the basics of JSP Elements and JSP Directives, usage of JSP Implicit Objects and JSP Expression Language , JSP Action Tags, JSP Cookies. JSP technology is used based on the servlet, as every JSP page is converted to servlet by the servlet container. Servlet is used to define the controllers in the MVC application whereas JSP pages take care for the presentation part.

The JSP is the presentation layer in the MVC model, it is most important how securely we can fetch the data from database and display it on the web page. The current trend in web applications is to fetch the data through javascript RPC call, the browser renders the data in the view . JSP is rendered at server side as first is converted into Servlet and served by the servlet container . So, what we get on the browser is HTML page including the data.

In this chapter, we will go through the database connection, fetch the data from database and display it on the JSP page. We will use the different built-in JSTL libraries to perform some basics utility functions such as iteration on the dataset or fetch the data set form the database

4.3 CONNECTING TO DATABASE

We will follow the best approach to do the database connection, the connection pool is the right approach when you are working in the java web application. Opening and

closing the connection will be taken care of by the connection pool which is managed by the web server.

Let's go through the database connection example and fetch the data into the JSP page. Create a new project contact book.

Import Libraries

Import libraries used to make the database connection and java standard tag libraries to manage the core template activities and database utility to fetch the data.

```
/WEB-INF/lib/postgresql-42.2.5.jar
/WEB-INF/lib/taglibs-standard-impl-1.2.5.jar
/WebContent/WEB-INF/lib/taglibs-standard-spec-1.2.5.jar
```

Database Connection

As explained above we will follow the best approach to make the connection with the database using the connection pool, let's create a context.xml file under the directory /META-INF/context.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>

<Resource name="jdbc/contactbook" auth="Container"
    type="javax.sql.DataSource" username="mantavyagajjar" password="shreeji"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://localhost:5432/stable" maxIdle="4" maxTotal="8"/>

</Context>
```

Index.jsp, to fetch the data we have used the sql taglib and to iterate and fetch the values we use the core JSTL library, which provides the.

```
<%@ page import="java.sql.*, javax.sql.*, javax.naming.*"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>Contact List</h2>

<sql:query var="result" dataSource="jdbc/contactbook">
    SELECT * FROM contact
</sql:query>

<table border="1" style="width: 100%">
<tr>
<th>Name</th>
<th>Email</th>
<th>Phone</th>
</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
<td><c:out value="${row.name}"/></td>
<td><c:out value="${row.phone}"/></td>
<td><c:out value="${row.email}"/></td>
</tr>
</c:forEach>
```

```
</table>
</body>
</html>
```

This code will produce the below output. If you look at the output closely, we get the same output which was generated by the servlet in Block-4 *Chapter 2: Servlet with JDBC*, under the *Database Connection Pooling* topic. The huge amount of code is reduced only because we place the piece of code in the right place. Servlet is not used to generate the user interface, the presentation layer has to be produced by the JSP page.

4.4 JAVA STANDARD TAG LIBRARIES

We have to see the basics of Java Standard Tag Libraries in chapter Chapter 3: Basics of Java Server Pages under the topic *JSP Taglib Directive*. During the previous example we have used two java standard tag libraries http://java.sun.com/jsp/jstl/core and http://java.sun.com/jsp/jstl/sql, which provides a great set of features to build the user interface.

Core Tags

All the JSP Expression Language statements can be replaced with the tags available in http://java.sun.com/jsp/jstl/core standard tag library.

```
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
```

Following table lists out the Formatting JSTL Tags

S.No.	Tag & Description
1	<c:out> Like <%= ... >, but for expressions.
2	<c:set > Sets the result of expression evaluation in a 'scope'
3	<c:remove > Removes a scoped variable (from a particular scope, if specified).
4	<c:catch> Catches any Throwable that occurs in its body and optionally exposes it.
5	<c:if> Simple conditional tag which evaluates its body if the supplied condition is true.
6	<c:choose> Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>.
7	<c:when> Subtag of <choose> that includes its body if its condition evaluates to 'true'.

8	<c:otherwise > Subtag of <choose> that follows the <when> tags and runs only if all of the prior conditions evaluated to 'false'.
9	<c:import> Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.
10	<c:forEach > The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality.
11	<c:forTokens> Iterates over tokens, separated by the supplied delimiters.
12	<c:param> Adds a parameter to a containing 'import' tag's URL.
13	<c:redirect > Redirects to a new URL.
14	<c:url> Creates a URL with optional query parameters

SQL Tags

The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as PostgreSQL, Oracle, MySQL, or Microsoft SQL Server.

```
<%@ taglib prefix = "sql" uri = "http://java.sun.com/jsp/jstl/sql" %>
```

Following is the syntax to include JSTL SQL library in your JSP

S.No.	Tag & Description
1	<sql:setDataSource> Creates a simple DataSource suitable only for prototyping
2	<sql:query> Executes the SQL query defined in its body or through the SQL attribute.
3	<sql:update> Executes the SQL update defined in its body or through the SQL attribute.
4	<sql:param> Sets a parameter in an SQL statement to the specified value.
5	<sql:dateParam> Sets a parameter in an SQL statement to the specified java.util.Date value.
6	<sql:transaction > Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

JSTL Functions

JSTL includes a number of standard functions, most of which are common string manipulation functions. Following is the syntax to include JSTL Functions library in your JSP −

```
<%@ taglib prefix = "fn" uri = "http://java.sun.com/jsp/jstl/functions" %>
```

Following table lists out the various JSTL Functions

S.No.	Function & Description
1	fn:contains() Tests if an input string contains the specified substring.
2	fn:containsIgnoreCase() Tests if an input string contains the specified substring in a case insensitive way.
3	fn:endsWith() Tests if an input string ends with the specified suffix.
4	fn:escapeXml() Escapes characters that can be interpreted as XML markup.
5	fn:indexOf() Returns the index within a string of the first occurrence of a specified substring.
6	fn:join() Joins all elements of an array into a string.
7	fn:length() Returns the number of items in a collection, or the number of characters in a string.
8	fn:replace() Returns a string resulting from replacing in an input string all occurrences

	with a given string.
9	fn:split() Splits a string into an array of substrings.
10	fn:startsWith() Tests if an input string starts with the specified prefix.
11	fn:substring() Returns a subset of a string.
12	fn:substringAfter() Returns a subset of a string following a specific substring.
13	fn:substringBefore() Returns a subset of a string before a specific substring.
14	fn:toLowerCase() Converts all of the characters of a string to lower case.
15	fn:toUpperCase() Converts all of the characters of a string to upper case.
16	fn:trim() Removes white spaces from both ends of a string.

4.5 EXAMPLE: CONTACT BOOK

We have gone through Servlet, Database and JSP topics in Unit 4, we have studied the different approaches of writing the Servlet, Servlet filters, Database Connection, Reading data from database and display those data onto the JSP page.

Let's see the full example with of address book, where we will create a new contact, read or search the contacts, edit the contact and delete the contacts using JSP, Servlet best practices.

Create Database

Let's first create the PostgreSQL database and create a contact table. Execute the below command to create the database and table.

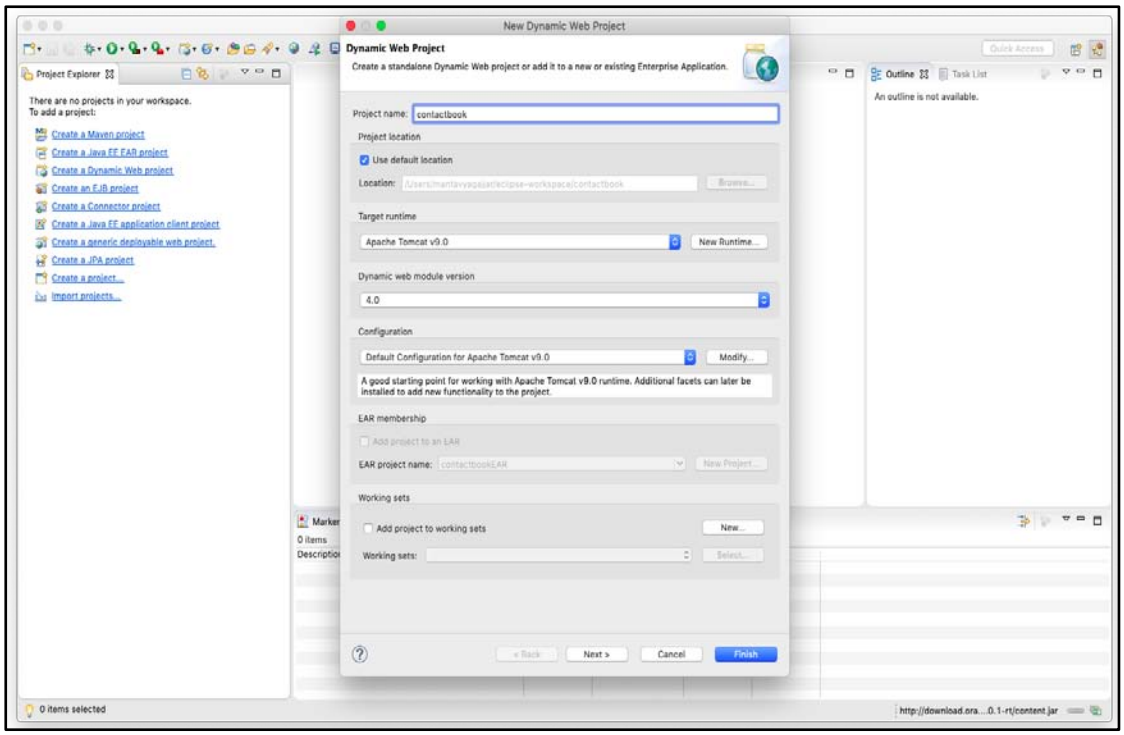
```
$ createdb contactbook --encoding=UNICODE

$ psql contactbook
```

Connect to the database and create a contact table.

```
# CREATE TABLE contact (
name VARCHAR (50),
   email VARCHAR (50) UNIQUE,
   phone VARCHAR (50),
   urlsafe VARCHAR(100)
);
```

Create a Project

This example we are going to create with Eclipse Studio, let's create the Dynamic Web Project in Eclipse and name it ***contactbook***. The blank project will be created with the default web configuration.



Setup the connection

First things first, set up the connection details and connection pool in the context.xml under the META-INF folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>

<Resource name="jdbc/contactbook" auth="Container"
    type="javax.sql.DataSource" username="mantavyagajjar" password="********"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://localhost:5432/contactbook" maxIdle="4" maxTotal="8"/>

</Context>
```

Show Contact List

The default page will display the list of contacts, when user access the /contactbook application, the contacts will be fetched from database and display on the index

page. We will create an index.jsp that show the list of contacts in the database and allow the user to perform the edit or delete operations on it.

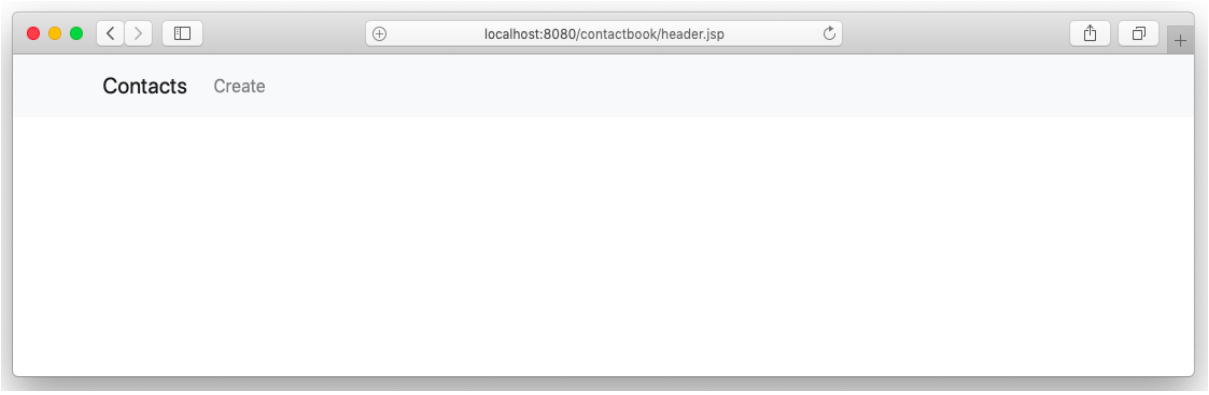
Header.jsp

A common header that creates a menubar, so every page has the same menu bar which includes the header.jsp page. We have also included the bootstrap CSS and font awesome icons, so other JSP pages do not have to import any CSS libraries.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
rel="stylesheet"/>
<link href="https://use.fontawesome.com/releases/v5.8.1/css/all.css"
rel="stylesheet"/>
</head>

<body>
<nav class="navbar navbar-expand-lg navbar-light bg-light mb-4">
<div class="container">
<a class="navbar-brand" href="/contactbook">Contacts</a>
<div class="collapse navbar-collapse" id="navbarNavAltMarkup">
<div class="navbar-nav">
<a class="nav-item nav-link" href="/contactbook/create.jsp">Create</a>
</div>
</div>
</div>
</nav>
</body>
```

```
</html>
```

So, if you access the header.jsp you can see the only menu bar on the page as below.



Index.jsp

The index.jsp page can be called with query string or without the query string, based on the parameters received it shows the data.

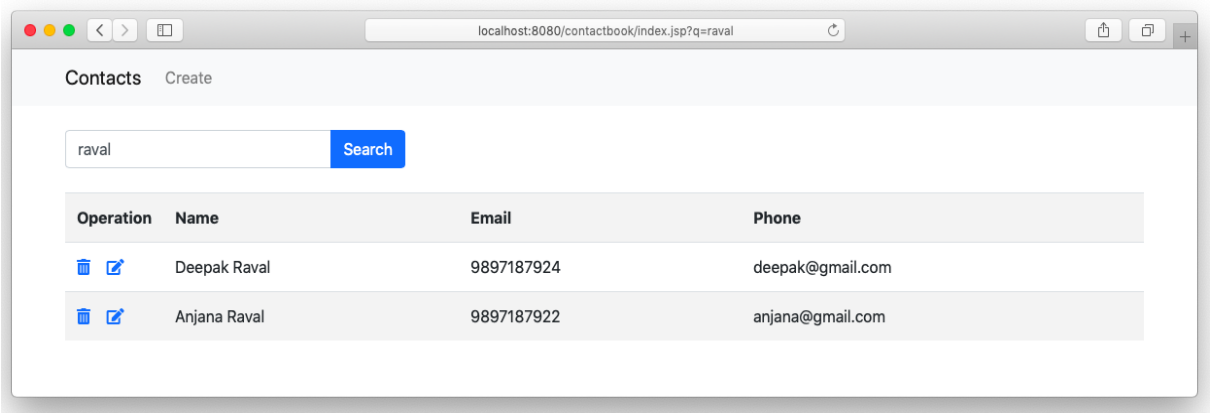
Show all contacts	http://localhost:8080/contactbook/
Apply the filter for name field	http://localhost:8080/contactbook/index.jsp?q=ajay

```
<%@ page import="java.sql.*, javax.sql.*, javax.naming.*"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<!DOCTYPE html>
<html>

<head>
```

```
<meta charset="UTF-8">
<title>Contact Book</title>
</head>

<body>
<%@ include file="header.jsp" %>
<div class="container">
<div class="row">
<div class="col-4">

<c:set var="searchName" value='<%=request.getParameter("q")%>'/>

<sql:query var="result" dataSource="jdbc/contactbook">
        SELECT * FROM contact WHERE name ilike ?
<sql:param value="%${searchName}%" />
</sql:query>

<form action="/contactbook/index.jsp" method="get">
<div class="input-group mb-4">
<input type="text" name="q" id="q" class="form-control" placeholder="Search"/>
<div class="input-group-append">
<input type="submit" value="Search" class="btn btn-primary"/>
</div>
</div>
</form>


</div>
</div>

<div class="row">
<div class="col">
<table class="table table-striped">
```

```
<tr>
<th width="10">Operation</th>
<th>Name</th>
<th>Email</th>
<th>Phone</th>
</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
<td>
<a href="/contactbook/Delete?record=${row.urlsafe}">
<i class="fas fa-trash-alt"></i>
</a>  
<a href="/contactbook/create.jsp?record=${row.urlsafe}">
<i class="fas fa-edit"></i>
</a>
</td>
<td><c:out value="${row.name}" /></td>
<td><c:out value="${row.phone}" /></td>
<td><c:out value="${row.email}" /></td>
</tr>
</c:forEach>
</table>
</div>
</div>
</div>
</body>
</html>
```

The index page shows all the data on the first load as the query string is not passed, when user search for the contact, the same page receives the query which applies to the SQL to filter on the data.



Create or Update Contact

The database operation such as create, exit or update records has to be done through the Servlet, we will write a servlet that will either create or update the record based on the request received from the user. Let's create the create form when a user enters the contact data and submit to the Create.java servlet. The same create.jsp page is used to edit the contact when the user clicks on the Edit icon beside the name on the contact list.

Create.jsp

Create page may receive a recordID parameter if received then the form will be edit mode or the default will be in create mode.

Create Model	http://localhost:8080/contactbook/create.jsp
Edit Mode	http://localhost:8080/contactbook/create.jsp?record=3ba1708d4d427814c9fa1b5a56675bee

```
<%@ page import="java.sql.*, javax.sql.*, javax.naming.*"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Create Contact</title>
</head>

<body>
<%@ include file="header.jsp" %>

<div class="container">
<div class="row">
<div class="col-6">
<h2>Contact Form</h2>

<c:set var="recordID" value='<%=request.getParameter("record")%>'/>

<sql:query var="result" dataSource="jdbc/contactbook">
        SELECT * FROM contact where urlsafe=?
<sql:param value="${recordID}" />
</sql:query>

<c:set var="row" value="${result.rows[0]}"/>

<form action="/contactbook/Create" method="post">
<div class="form-group">
<input hidden type="text" name="recordID" id="recordID"
            class="form-control" value="${row.urlsafe}"/>
</div>
<div class="form-group">
<label for="name">Name</label>
```

```
<input type="text" name="name" id="name"
            class="form-control" value="${row.name}"/>
</div>
<div class="form-group">
<label for="name">Email</label>
<input type="text" name="email" id="email"
            class="form-control" value="${row.email}"/>
</div>
<div class="form-group">
<label for="name">Phone</label>
<input type="text" name="phone" id="phone"
            class="form-control" value="${row.phone}"/>
</div>
<input type="submit" value="Save Contact" class="btn btn-primary"/>
</form>
</div>
<div class="col-6">
</div>
</div>
</div>
</body>
</html>
```

The cr eate.jsp page f etch t he record f rom t he dat abase w hen i t r eceived t he recordID, t he page will r etrieve t he dat a using t he ur lsafe ke y and se t i n t he respective fields, a hi dden field on t he form will be filled with the value of recordID when received. When user submit the form all the data submitted to the Create.java servlet

Create.java

Create Servlet received data from create.jsp page, if the form is in edit mode servlet receive the recordID in addition to the other fields.

```
import java.sql.*;
import javax.sql.*;

import java.io.*;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet("/Create")
publicclassCreateextendsHttpServlet {
privatestaticfinallong serialVersionUID = 1L;

private DataSource dataSource;
private Connection connection;
private PreparedStatement statement;

publicCreate() {
super();
    }

@Override
publicvoidinit(ServletConfig config)
throws ServletException {

super.init(config);
try{
            Context initContext = new InitialContext();
            Context envContext = (Context) initContext.lookup("java:/comp/env");
            dataSource = (DataSource) envContext.lookup("jdbc/contactbook");
        } catch (NamingException e) {
```

```
        }
    }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        String SQL = "INSERT INTO contact (name, phone, email, urlsafe) VALUES (?,
?, ?, md5(?))";
        String recordID = request.getParameter("recordID");
if(recordID.length() >0) {
          SQL = "UPDATE contact SET name=?, phone=?, email=?, urlsafe=md5(?)
WHERE urlsafe=?";
        }
try {
           connection = dataSource.getConnection();
           statement = connection.prepareStatement(SQL);
           statement.setString(1, request.getParameter("name"));
           statement.setString(2, request.getParameter("phone"));
           statement.setString(3, request.getParameter("email"));
           statement.setString(4, request.getParameter("email"));

if(recordID.length() >0) {
          statement.setString(5, recordID);
           }
           statement.execute();
        } catch (SQLException e) {

        }
        response.sendRedirect("/contactbook/index.jsp");
    }

protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {

        doGet(request, response);
    }

}
```

Servlet takes care of creating a new record or updating the existing record in the database and redirect to the index.jsp page.

Delete Contact

The Delete Servlet takes urlsafe key from the index.jsp page and delete the record. If receive record parameter then executes the delete query else returns back to the index.jsp page.

```
import java.sql.*;
import javax.sql.*;
import java.io.*;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet("/Delete")
publicclassDeleteextendsHttpServlet {
privatestaticfinallong serialVersionUID = 1L;

private DataSource dataSource;
private Connection connection;
private PreparedStatement statement;
```

```
publicDelete() {
super();
   }

@Override
publicvoidinit(ServletConfig config)
throws ServletException {

super.init(config);
try{
          Context initContext = new InitialContext();
          Context envContext = (Context) initContext.lookup("java:/comp/env");
          dataSource = (DataSource) envContext.lookup("jdbc/contactbook");
       } catch (NamingException e) {

       }
   }

protectedvoiddoGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

if(request.getParameter("record").length() <= 0) {
          response.sendRedirect("/contactbook/index.jsp");
       }

       String SQL = "DELETE FROM contact WHERE urlsafe=?";
try {
          connection = dataSource.getConnection();
          statement = connection.prepareStatement(SQL);
          statement.setString(1, request.getParameter("record"));

          statement.execute();
```

```
        } catch (SQLException e) {

        }
        response.sendRedirect("/contactbook/index.jsp");
    }

protectedvoiddoPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        doGet(request, response);
    }
}
```

Download Example

Download a copy of the full example, it is an Eclipse Dynamic Web Project https://drive.google.com/file/d/1VgASNRsQ-iFH8s8LEr-3w6vpLoKG3JcJ/view?usp=sharing