B.S.C. (IT)

SEMESTER-V PAPER-IV (CBCS)

TYIT ADVANCED JAVA

Prof. Suhas Pednekar
Vice Chancellor
University of Mumbai, Mumbai.

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai.

Prof. Prakash Mahanwar
Director
IDOL, University of Mumbai.

Programe Co-ordinator : **Mandar L. Bhanushe**
Head, Faculty of Science and Technology,
IDOL, University of Mumbai – 400098.

Course Co-ordinator : **Sumedh Pandit Shejole,**
Assistant Professor,
B.Sc.(Information Technology),
Institute of Distance & Open Learning,
University of Mumbai- 400098.

Course Writers : **Dr. Rakhee Yadav**
Assistant Professor
S. K. Somaiya College of Arts, Science and Commerce

: **Mr. Milind Thorat**
Assistant Professor
K. J. Somaiya Institute of Engineering and Information Technology

: **Ms. Rohini Desai**
Assistant Professor
Vidyalankar School of Information Technology

: **Ms. Aarti Sahitya**
Assistant Professor
K. J. Somaiya Institute of Engineering and Information Technology

September 2021, Print I

Published by : **Director**
Institute of Distance and Open Learning ,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP Composed : **Univeristy Press**
Institute of Distance and Open Learning

Printed by :

CONTENTS

*****

SYLLABUS

CLASS: B. Sc (Information technology)		**Semester–V**	
Paper IV; SUBJECT: Advanced Java			
Periods per week **1 Period is 50 minutes**	**Lecture**	**5**	
	TW/Tutorial/Practical	**3**	
		Hours	**Marks**
Evaluation System	**Theory Examination**	2	60
	TW/Tutorial/Practical	-	40

Unit-I	**Swing:** Event Handling, JFrames, Lists, Tables, Trees, Text Components, Progress Indicators, Component Organizers
Unit-II	**Introduction to servlets:** Need for dynamic content, java servlet technology, why servlets? **Servlet API and Lifecycle:** servlet API, servletConfig interface, ServletRequest and ServletResponse Interfaces, GenericServlet Class. ServletInputStream And ServletOutputStreamClasses,RequestDispatcher Interface,HttpServlet Class, HttpServletRequest and HttpServletResponse Interfaces, HttpSession Interface, Servlet Lifecycle. **Working with servlets:** organization of a web application, creating a web application(using netbeans), creating a servlet, compiling and building the web application
Unit-III	**JDBC:** Design of JDBC, JDBC configuration, Executing SQL statement, Query Execution, Scrollable and updatable result sets, row sets, metadata, Transaction. **JSP:** Introduction, disadvantages, JSP v/s Servlets, Lifecycle of JSP, Comments, JSP documents, JSP elements, Action elements, implicit objects, scope, characterquoting conventions, unified expression language.
Unit-IV	**Java server Faces :** Need of MVC , what is JSF?, components of JSF, JSF as an application, JSF lifecycle, JSF configuration, JSF web applications (login form, JSF pages) **EJB:** Enterprise bean architecture, Benefits of enterprise bean, types of beans, Accessing beans, packaging beans, creating web applications, creating enterprise bean, creating web client, creating JSP file, building and running web application.
Unit-V	**HIBERNATE:** Introduction, Writing the application, application development approach, creating database and tables in MySQL, creating a web application, Adding the required library files, creating a java bean class, creating hibernate configuration and mapping file, adding a mapping resource, creating JSPs. **STRUTS:** Introduction, Struts framework core components, installing and setting up struts, getting started with struts.

Unit-VI	**WEB Services:** SOAP, Building a web services using JAX-WS, Building web service. **JAVAMAIL:** Mail Protocols, Components of the Javamail API, JAVAMAIL API, Starting with API. **JNDI:** NAMING Service, Directory service, JNDI, Resources and JNDI,

Books:

Java EE 6 for Beginners, Sharanam Shah, Vaishali Shah, SPD (Unit II to VI) Core Java Vol. II – Advanced Features, Cay S. Horstmans, Gary Coronell, Eight Edition, Pearson (Unit I and III) Java Complete Reference, Herbert Schildt, Seventh Edition,TMH. (Unit I)

References:

Java EE Project using EJB 3, JPA and struts 2 for beginners, Shah, SPD
Java Programming A practical Approach, C Xavier, McGraw Hill Java Server Faces A practical Approach for beginners, B M Harwani, Eastern Economy Edition (PHI). Advanced Java Technology, Savaliya, Dreamtech.

Term Work:

Assignments: Should contain at least 6 assignments (one per unit) covering the Syllabus.

Practicals
1. Write a java program to present a set of choices for a user to select Stationary products and display the price of Product after Selection from the list. 2. Write a java program to demonstrate typical Editable Table, describing employee details for a software company. 3. Write a java program using Split pane to demonstrate a screen divided in two parts, one part contains the names of Planets and another Displays the image of planet. When user selects the planet name form Left screen, appropriate image of planet displayed in right screen.
4. Develop Simple Servlet Question Answer Application to demonstrate use of HttpServletRequest and HttpServletResponse interfaces. 5. Develop Servlet Application of Basic Calculator (+,-,*, /, %) using ServletInputStream and ServletOutputStream.
6. Develop a JSP Application to accept Registration Details form user and Store it into the database table. 7. Develop a JSP Application to Authenticate User Login as per the registration details. If login success the forward user to Index Page otherwise show login failure Message.
8. Develop a web application to add items in the inventory using JSF. 9. Develop a Room Reservation System Application Using Enterprise Java Beans.
10. Develop a Hibernate application to store Feedback of Website Visitor in MySQL Database.

11. a. Develop a simple Struts Application to Demonstrate 3 page Website of Teaching Classes which passes values from every page to another.

b. Develop a simple Struts Application to Demonstrate E-mail Validator.

*****

UNIT I

1

SWINGS

Unit Structure

1.0 EVENT HANDLING

Event handling is fundamental to Java programming because it is integral to the creation of applets and other types of GUI-based programs. any program that uses a graphical user interface, such as a Java application written for Windows, is event driven. Thus, you cannot write these types of programs without a solid command of event handling. Events are supported by a number of packages, including **java.util, java.awt**, and **java.awt.event**. When a user interacts with a GUI-based software, the majority of events to which your programme will reply are created. There are several types of events, including those generated by the mouse, the keyboard, and various GUI controls, such as a push button, scroll bar, or check box. It then examines the main event classes and interfaces used by the AWT and develops several examples that demonstrate the fundamentals of event processing. The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a source generates an event and sends it to one or more listeners. Swing components do respond to user input and the events generated by those interactions need to be handled. Events can also be generated in ways not directly related to user input. For example, an event is generated when a timer goes off. Whatever the case, event handling is a large part of any Swing-based application.

1.1 SWINGS

The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents, or peers. This means that the look and feel of a component is defined by the platform, not by Java. Because the AWT components use native code resources, they are referred to as heavyweight. The use of native peers led to several problems. First, because of variations between operating systems, a component might look, or even act, differently on different platforms. This potential variability threatened the overarching philosophy of Java: write once, run anywhere. Second, the look and feel of each component was fixed (because it is defined by the platform) and could not be (easily) changed. Third, the use of heavyweight components caused some frustrating restrictions.

1.2 JFRAME

JFrame is the top-level container that is commonly used for Swing applications. JLabel is the Swing component that creates a label, which is a component that displays information. The label is Swing's simplest component because it is passive. That is, a label does not respond to user input. It just displays output. The program uses a JFrame container to hold an instance of a JLabel. The label displays a short text message.
A simple java program for JFrame:

```
import javax.swing.*;

class SwingDemo {

SwingDemo() {

// Create a new JFrame container.

JFramejfrm = new JFrame("A Simple Swing Application");

// Give the frame an initial size.

jfrm.setSize(275, 100);

// Terminate the program when the user closes the application.

jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Create a text-based label

JLabeljlab = new JLabel(" Swing means powerful GUIs.");

// Add the label to the content pane.

jfrm.add(jlab);

// Display the frame.

jfrm.setVisible(true);
```

```
}
public static void main(String args[]) {
// Create the frame on the event dispatching thread.
SwingUtilities.invokeLater(new Runnable() {
public void run() {
new SwingDemo();
}
});
}
}
```

Theconstructor is where most of the action of the program occurs. It begins by creating a JFrame, using this line of code:

```
JFramejfrm = new JFrame("A Simple Swing Application");
```

This creates a container called jfrm that defines a rectangular window complete with a titlebar; close, minimize, maximize, and restore buttons; and a system menu. Thus, it creates astandard, top-level window. The title of the window is passed to the constructor. Next, the window is sized using this statement:

```
jfrm.setSize(275, 100);
```

The **setSize( )** method (which is inherited by JFrame from the AWT class Component) setsthe dimensions of the window, which are 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 275 and the height is set to 100.By default, when a top-level window is closed (such as when the user clicks the closebox), the window is removed from the screen, but the application is not terminated. Whilethis default behavior is useful in some situations, it is not what is needed for most applications.Instead, you will usually want the entire application to terminate when its top-levelwindow is closed. There are a couple of ways to achieve this. The easiest way is to call**setDefaultCloseOperation**( ), as the program does:

```
jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

After this call executes, closing the window causes the entire application to terminate. The general form of setDefaultCloseOperation( ) is shown here:

void setDefaultCloseOperation(int what)

The value passed in what determines what happens when the window is closed. There areseveral other options in addition to JFrame.EXIT_ON_CLOSE. They are shown here:

JFrame.DISPOSE_ON_CLOSE

JFrame.HIDE_ON_CLOSE

JFrame.DO_NOTHING_ON_CLOSE

Their names reflect their actions. These constants are declared in WindowConstants, which is an interface declared in javax.swing that is implemented by JFrame. The next line of code creates a Swing JLabel component:

JLabeljlab = new JLabel(" Swing means powerful GUIs.");

JLabel is the simplest and easiest-to-use component because it does not accept user input. It simply displays information, which can consist of text, an icon, or a combination of the two.The label created by the program contains only text, which is passed to its constructor.The next line of code adds the label to the content pane of the frame:

jfrm.add(jlab);

1.2.1 JFrame constructors:

1. **JFrame()** – JFrame() is a JFrame class constructor which creates a new Frame. By default, it remains invisible.
2. **JFrame(String title, GraphicsConfigurationgc)** – This constructor creates a JFrame in the specified graphical configuration & with the specified title as in parameter.
3. **JFrame(GraphicsConfigurationgc)** – This constructor creates a JFrame in the specified graphical configuration as it is in the parameter.
4. **JFrame(String title)** – This constructor creates a JFrame with the specified title as in parameter.

1.2.2 JFrame Methods:

JFrame class provides some methods which play an important role in working with JFrame**.**

1. **AccessibleContextgetAccessibleContext() –** This method gets the accessible context that remains associated with the JFrame.

2. **Container getContentPane() –** This method creates the JFrame'scontentPane object**.**
3. **Component getGlassPane() –** This method creates the glassPane object for JFrame**.**
4. **int getDefaultCloseOperation() –** When the user clicks on the close button on this Frame then this method returns the operation**.**
5. **JMenuBargetJMenuBar() –** Menubar set created at the Frame by using this method.
6. **JLayeredPanegetLayeredPane() –** LayeredPane object is returned by this method.
7. **JRootPanegetRootPane() –** The rootPane object is returned by this method**.**

1.3 LISTS

List in Java provides the facility to maintain the ordered collection. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.The List interface is found in the **java.util** package and inherits the Collection interface. It is a factory of **ListIterator** interface. Through the **ListIterator**, we can iterate the list in forward and backward directions. The implementation classes of List interface are ArrayList, LinkedList, Stack and Vector. The ArrayList and LinkedList are widely used in Java programming. The syntax is given as

public interface List<E>**extends** Collection<E>

1.3.1 List Methods:

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
booleanadd(E e)	It is used to append the specified element at the end of a list.
booleanaddAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of a list.
booleanaddAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void clear()	It is used to remove all of the elements from this list.
booleanequals(Object o)	It is used to compare the specified object with the elements of a list.
int hashcode()	It is used to return the hash code value for a list.
Eget(int index)	It is used to fetch the element from the particular position of the list.

booleanisEmpty()	It returns true if the list is empty, otherwise false.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
<T>T[] toArray(T[] a)	It is used to return an array containing all of the elements in this list in the correct order.
booleancontains(Object o)	It returns true if the list contains the specified element
booleancontainsAll(Collection<?> c)	It returns true if the list contains all the specified element
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
E remove(int index)	It is used to remove the element present at the specified position in the list.
booleanremove(Object o)	It is used to remove the first occurrence of the specified element.
booleanremoveAll(Collection<?> c)	It is used to remove all the elements from the list.
void replaceAll(UnaryOperator<E> operator)	It is used to replace all the elements from the list with the specified element.
void retainAll(Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collection.
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of specified comparator.

Example:

```
import java.util.*;
public class ListExample1{
public static void main(String args[]){
 //Creating a List
List<String> list=new ArrayList<String>();
//Adding elements in the List
list.add("Mango");
list.add("Apple");
list.add("Banana");
```

```
list.add("Grapes");
 //Iterating the List element using for-each loop
for(String fruit:list)
System.out.println(fruit);
  }
}
```

1.4 TABLE

JTable is a component that displays rows and columns of data. You can drag the cursor on column boundaries to resize columns. You can also drag a column to a new position. Depending on its configuration, it is also possible to select a row, column, or cell within the table, and to change the data within a cell. JTable is a sophisticated component that offers many more options and features than can be discussed here. (It is perhaps Swing's most complicated component.) However, in its default configuration, JTable still offers substantial functionality that is easy to use—especially if you simply want to use the table to present data in a tabular format. The brief overview presented here will give you a general understanding of this powerful component. Like JTree, JTable has many classes and interfaces associated with it. These are packaged in javax.swing.table. JTable supplies several constructors. The one used here is JTable(Object data[ ][ ], Object colHeads[ ]) Here, data is a two-dimensional array of the information to be presented, and colHeads is a one-dimensional array with the column headings. JTable relies on three models. The first is the table model, which is defined by the TableModel 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 by TableColumnModel. JTable is defined in terms of columns, and it is TableColumnModel that specifies the characteristics of a column. These two models are packaged in javax.swing.table.A JTable can generate several different events. The two most fundamental to a table's operation are ListSelectionEvent and TableModelEvent. A ListSelectionEvent is generated when the user selects something in the table. By default, JTable allows you to select one or more complete rows, but you can change this behavior to allow one or more columns, or one or more individual cells to be selected. A TableModelEvent is fired when that table's data changes in some way.

Constructors:

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

Program for JTable:

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JTableDemo" width=400 height=200>
</applet>
*/
```

```
public class JTableDemo extends JApplet {
public void init() {
try {
SwingUtilities.invokeAndWait(
new Runnable() {
public void run() {
makeGUI();
}
}
);
} catch (Exception exc) {
System.out.println("Can't create because of " + exc);
}
}
private void makeGUI() {
// Initialize column headings.
String[] colHeads = { "Name", "Extension", "ID#" };
// Initialize data.
Object[][] data = {
{ "Gail", "4567", "865" },
{ "Ken", "7566", "555" },
{ "Viviane", "5634", "587" },
{ "Melanie", "7345", "922" },
{ "Anne", "1237", "333" },
{ "John", "5656", "314" },
{ "Matt", "5672", "217" },
{ "Claire", "6741", "444" },
{ "Erwin", "9023", "519" },
{ "Ellen", "1134", "532" },
{ "Jennifer", "5689", "112" },
{ "Ed", "9030", "133" },
{ "Helen", "6751", "145" }
};
  // Create the table.
JTable table = new JTable(data, colHeads);
// Add the table to a scroll pane.
JScrollPanejsp = new JScrollPane(table);
// Add the scroll pane to the content pane.
add(jsp);
}
}
```

1.5 TREES

A tree is a component that presents a hierarchical view of data. The user has the ability to expand or collapse individual subtrees in this display. Trees are implemented in Swing by the JTree class.

JTree(Object obj[ ])

JTree(Vector v)

JTree(TreeNodetn)

In the first form, the tree is constructed from the elements in the array obj. The second form constructs the tree from the elements of vector v. In the third form, the tree whose root nodeis specified by tn specifies the tree. Although JTree is packaged in **javax.swing**, its support classes and interfaces are packaged in **javax.swing.tree**. This is because the number of classes and interfaces neededto support JTree is quite large. JTree relies on two models: TreeModel and TreeSelectionModel. A JTree generates a variety of events, but three relate specifically to trees: TreeExpansionEvent, TreeSelectionEvent, and TreeModelEvent. TreeExpansionEvent events occur when a nodeis expanded or collapsed. A TreeSelectionEvent is generated when the user selects ordeselects a node within the tree. A TreeModelEvent is fired when the data or structure of thetree changes. The listeners for these events are TreeExpansionListener, TreeSelectionListener, and TreeModelListener, respectively. The tree event classes and listener interfaces are packaged in **javax.swing.event**.The event handled by the sample program shown in this section is TreeSelectionEvent. To listen for this event, implement TreeSelectionListener. It defines only one method, called valueChanged( ), which receives the TreeSelectionEvent object. You can obtain the path to the selected object by calling getPath( ), shown here, on the event object.

TreePathgetPath( )

It returns a TreePath object that describes the path to the changed node. The TreePath classencapsulates information about a path to a particular node in a tree. The TreeNode interface declares methods that obtain information about a tree node. The MutableTreeNode interface extends TreeNode. It declares methods that can insert and remove child nodes or change the parent node. The DefaultMutableTreeNode class implements the MutableTreeNode interface. It represents a node in a tree. One of its constructors is shown here:

DefaultMutableTreeNode(Object obj)

Here, obj is the object to be enclosed in this tree node. The new tree node doesn't have a parent or children. To create a hierarchy of tree nodes, the add( ) method of DefaultMutableTreeNode canbe used. Its signature is shown here:

void add(MutableTreeNode child)

Here, child is a mutable tree node that is to be added as a child to the current node. JTree does not provide any scrolling capabilities of its own. Instead, a JTree is typically placed within a JScrollPane. This way, a large tree can be scrolled through a smaller viewport. Theprogram creates a DefaultMutableTreeNode instance labeled “Options.” This is the topnode of the tree hierarchy. Additional tree nodes are then created, and the add( ) method is called to connect these nodes to the tree. A reference to the top node in the tree is provided as the argument to the JTree constructor. The tree is then provided as the argument to the JScrollPane constructor. This scroll pane is then added to the content pane. Next, a label is created and added to the content pane. The tree selection is displayed in this label. To receive selection events from the tree, a TreeSelectionListener is registered for the tree. Inside the valueChanged( ) method, the path to the current selection is obtained and displayed.

```
// Demonstrate JTree.
import java.awt.*;
import javax.swing.event.*;
import javax.swing.*;
import javax.swing.tree.*;
/*
<applet code="JTreeDemo" width=400 height=200>
</applet>
*/
public class JTreeDemo extends JApplet {
JTree tree;
JLabeljlab;
public void init() {
try {
SwingUtilities.invokeAndWait(
new Runnable() {
public void run() {
makeGUI();
}
}
);
} catch (Exception exc) {
System.out.println("Can't create because of " + exc);
}
}
private void makeGUI() {
// Create top node of tree.
DefaultMutableTreeNode top = new DefaultMutableTreeNode("Options");
```

```
// Create subtree of "A".
DefaultMutableTreeNode a = new DefaultMutableTreeNode("A");
top.add(a);
DefaultMutableTreeNode a1 = new DefaultMutableTreeNode("A1");
a.add(a1);
DefaultMutableTreeNode a2 = new DefaultMutableTreeNode("A2");
a.add(a2);
// Create subtree of "B".
DefaultMutableTreeNode b = new DefaultMutableTreeNode("B");
top.add(b);
DefaultMutableTreeNode b1 = new DefaultMutableTreeNode("B1");
b.add(b1);
DefaultMutableTreeNode b2 = new DefaultMutableTreeNode("B2");
b.add(b2);
DefaultMutableTreeNode b3 = new DefaultMutableTreeNode("B3");
b.add(b3);
// Create the tree.
tree = new JTree(top);
// Add the tree to a scroll pane.
JScrollPanejsp = new JScrollPane(tree);
// Add the scroll pane to the content pane.
add(jsp);
// Add the label to the content pane.
jlab = new JLabel();
add(jlab, BorderLayout.SOUTH);
// Handle tree selection events.
tree.addTreeSelectionListener(new TreeSelectionListener() {
public void valueChanged(TreeSelectionEventtse) {
jlab.setText("Selection is " + tse.getPath());
}
});
}
}
```

1.6 TEXT COMPONENTS

It generates text events when the user enters a character. The plain text components (text field, password field, and text area) are the easiest and most commonly used components. In a few lines of code, you can easily create, configure, and use a plain text component in your program. The components that can display styled text (editor pane and text pane) typically require more effort to use. Most programmers using editor pane

or text pane need to build a user interface that lets the user manipulate the text styles. Also, getting the content from a styled text component typically requires more code than a simple call to getText. Yet, as the diagram shows, both plain and styled text components inherit from JTextComponent. This abstract base class provides a highly-configurable and powerful foundation for text manipulation. JTextComponent provides these customizable features for all of its descendants.

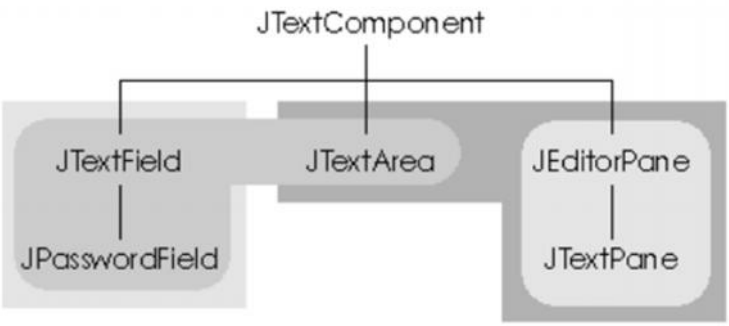


Figure: Text components

1.6.1 JTextField:

JTextField is the simplest Swing text component. It is also probably its most widely used text component. JTextField allows you to edit one line of text. It is derived from JTextComponent, which provides the basic functionality common to Swing text components. JTextField uses the Document interface for its model.

Three of JTextField's constructors are shown here:

JTextField(int cols)

JTextField(String str, int cols)

JTextField(String str)

Here, str is the string to be initially presented, and cols is the number of columns in the text field. JTextField generates events in response to user interaction. For example, an ActionEvent is fired when the user presses ENTER. To obtain the text currently in the text field, call getText( ). It creates a JTextField and adds it to the content pane. When the user presses ENTER, an action event is generated. This is handled by displaying the text in the status window.

```
// Demonstrate JTextField.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JTextFieldDemo" width=300 height=50>
</applet>
*/
public class JTextFieldDemo extends JApplet {
```

```
JTextFieldjtf;
public void init() {
try {
SwingUtilities.invokeAndWait(
new Runnable() {
public void run() {
makeGUI();
}
}
);
} catch (Exception exc) {
System.out.println("Can't create because of " + exc);
}
}
private void makeGUI() {
// Change to flow layout.
setLayout(new FlowLayout());
// Add text field to content pane.
jtf = new JTextField(15);
add(jtf);
jtf.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ae) {
// Show text when user presses ENTER.
showStatus(jtf.getText());
}
});
}
}
```

1.6.2 JPasswordField:

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

public class JPasswordField extends JTextField

Constructors:

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**.**

```
import javax.swing.*;
public class PasswordFieldExample {
    public static void main(String[] args) {
    JFrame f=new JFrame("Password Field Example");
    JPasswordField value = new JPasswordField();
    JLabel l1=new JLabel("Password:");
       l1.setBounds(20,100, 80,30);
        value.setBounds(100,100,100,30);
           f.add(value);  f.add(l1);
           f.setSize(300,300);
           f.setLayout(null);
           f.setVisible(true);
}
}
```

1.6.3 JTextArea:

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class.

public class JTextArea extends JTextComponent

Constructors:

JTextArea(): Creates a text area that displays no text initially.

JTextArea(String s): Creates a text area that displays specified text initially.

JTextArea(int row, int column): Creates a text area with the specified number of rows and columns that displays no text initially.

JTextArea(String s, int row, int column): Creates a text area with the specified number of rows and columns that displays specified text.

Methods:

void setRows(int rows): It is used to set specified number of rows.

void setColumns(int cols): It is used to set specified number of columns.

void setFont(Font f): It is used to set the specified font.

void insert(String s, int position): It is used to insert the specified text on the specified position.

void append(String s): It is used to append the given text to the end of the document.

```
import javax.swing.*;
```

```
public class TextAreaExample
{
TextAreaExample(){
JFrame f= new JFrame();
JTextArea area=new JTextArea("Welcome to javatpoint");
area.setBounds(10,30, 200,200);
f.add(area);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
    }
public static void main(String args[])
   {
  new TextAreaExample();
   }}
```

1.7 PROGRESS INDICATORS

A JProgressBar is a Swing component that indicates progress. A ProgressMonitor is a dialog box that contains a progress bar. A ProgressMonitorInputStream displays a progress monitor dialog box while the stream is read. JProgressBar is already discussed in previous section.

Progress Monitors:

A progress bar is a simple component that can be placed inside a window. In contrast, a ProgressMonitor is a complete dialog box that contains a progress bar. The dialog box contains a Cancel button. If you click it, the monitor dialog box is closed. In addition, your program can query whether the user has canceled the dialog box and terminate the monitored action. (Note that the class name does not start with a "J".)

Figure: Progress Monitors

1.8 COMPONENT ORGANISER

The study of advanced Swing features comes to a close with a look at components that aid in the organisation of other components. These include the split pane, which divides an area into many pieces with adjustable bounds, the tabbed pane, which allows a user to flip among multiple panels using tab dividers, and the desktop pane, which can be used to implement apps.

Split Panes:

Split panes divide a component into two pieces separated by an adjustable boundary. A frame with two split pan with a text area on the bottom and another split pane on top, the outer pane is split vertically. A list on the left and a label with an image on the right make up that pane, which is split horizontally. We construct a split pane by specifying the orientation, one of JSplitPane. HORIZONTAL_SPLIT or JSplitPane.VERTICAL_SPLIT, followed by the two components. For example,

JSplitPaneinnerPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, planetList, planetImage);

That's all you have to do. If you like, you can add "one-touch expand" icons to the splitter bar. We see those icons in the top pane inthe Metal look and feel, they are small triangles. If you click one of them, the splitter moves all the way in the direction to which the triangle is pointing, expanding one of the panes completely.

```
innerPane.setOneTouchExpandable(true);
```

TabbedPane:

Tabbed panes are a familiar user interface device to break up a complex dialog box into subsets of related options. We can also use tabs to let a user flip through a set of documents or images. To create a tabbed pane, you first construct a JTabbedPane object, then you add tabs to it.

JTabbedPanetabbedPane = new JTabbedPane(); tabbedPane.addTab(title, icon, component);

The last parameter of the addTab method has type Component. To add multiple components into the same tab, you first pack them up in a container, such as a JPanel. The icon is optional; for example, the addTab method does not require an icon:

tabbedPane.addTab(title, component);

We can also add a tab in the middle of the tab collection with the insertTab method:

tabbedPane.insertTab(title, icon, component, tooltip, index);

To remove a tab from the tab collection, use
tabPane.removeTabAt(index);

When we add a new tab to the tab collection, it is not automatically displayed. We must select it with the setSelectedIndex method. For example, here is how you show a tab that you just added to the end:

tabbedPane.setSelectedIndex(tabbedPane.getTabCount() - 1);

If we have a lot of tabs, then they can take up quite a bit of space. Starting with JDK 1.4, we can display the tabs in scrolling mode, in which only one row of tabs is displayed, together with a set of arrow buttons that allow the user to scroll through the tab set.

We set the tab layout to wrapped or scrolling mode by calling

tabbedPane.setTabLayoutPolicy(JTabbedPane.WRAP_TAB_LAYOUT);

REFERENCES

- Java Complete Reference, Herbert Schildt, Seventh Edition, Tata McGraw Hill. (Unit I Chapter 20,21,22)
- https://www.javatpoint.com/

UNIT END QUESTIONS

1. What is Servlets?
2. Explain Text components.
3. Explain Progress Bar in detail.

*****

UNIT II

2

INTRODUCTION TO SERVLETS

Unit Structure

2.0 NEED FOR DYNAMIC CONTENT

In order to understand the advantages of servlets, you must have a basic understanding of how web browsers and servers cooperate to provide content to a user. Consider a request for a static web page. A user enters a Uniform Resource Locator (URL) into a browser. The browser generates an HTTP request to the appropriate web server. The web server maps this request to a specific file. That file is returned in an HTTP response to the browser. The HTTP header in the response indicates the type of the content. The Multipurpose Internet Mail Extensions (MIME) are used for this purpose. For example, ordinary ASCII text has a MIME type of text/plain. The Hypertext Markup Language (HTML) source code of a web page has a MIME type of text/html. Now consider dynamic content. Assume that an online store uses a database to store information

about its business. This would include items for sale, prices, availability, orders, and so forth. It wishes to make this information accessible to customers via web pages. The contents of those web pages must be dynamically generated to reflect the latest information in the database. In the early days of the Web, a server could dynamically construct a page by creating a separate process to handle each client request. The process would open connections to one or more databases in order to obtain the necessary information. It communicated with the web server via an interface known as the Common Gateway Interface (CGI). CGI allowed the separate process to read data from the HTTP request and write data to the HTTP response. A variety of different languages were used to build CGI programs. These included C, C++, and Perl. However, CGI suffered serious performance problems. It was expensive in terms of processor and memory resources to create a separate process for each client request. It was also expensive to open and close database connections for each client request. In addition, the CGI programs were not platform-independent. Therefore, other techniques were introduced. Among these are servlets.

Servlets offer several advantages in comparison with CGI. First, performance is significantly better. Servlets execute within the address space of a web server. It is not necessary to create a separate process to handle each client request. Second, servlets are platform-independent because they are written in Java. Third, the Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. Finally, the full functionality of the Java class libraries is available to a servlet

2.1 JAVA SERVLET TECHNOLOGY

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server. Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically. Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

2.2 WHY SERVLETS?

There are many advantages of Servlet over CGI. 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:

1. **Better performance:** because it creates a thread for each request, not process.

2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

2.3 SERVLET API

Two packages contain the classes and interfaces that are required to build servlets. These are javax.servlet and javax.servlet.http. They constitute the Servlet API. Keep in mind that these packages are not part of the Java core packages. Instead, they are standard extensions provided by Tomcat. Therefore, they are not included with Java SE 6. The Servlet API has been in a process of ongoing development and enhancement. The current servlet specification is version 2.4, and that is the one used in this book. However, because changes happen fast in the world of Java, you will want to check for any additions or alterations. The javax.servlet package contains a number of interfaces and classes that establish the framework in which servlets operate. The following table summarizes the core interfaces that are provided in this package. The most significant of these is Servlet. All servlets must implement this interface or extend a class that implements the interface. The ServletRequest and ServletResponse interfaces are also very important.

Interface	Description
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.

Figure 2.1: Interfaces provided by javax.servlet package

Class	Description
GenericServlet	Implements the **Servlet** and **ServletConfig** interfaces.
ServletInputStream	Provides an input stream for reading requests from a client.
ServletOutputStream	Provides an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable.

Figure 2.2:Core classes of javax.servlet package

2.4 SERVLET AND SERVLETCONFIG INTERFACE

All servlets must implement the Servlet interface. It declares the init( ), service( ), and destroy( ) methods that are called by the server during the life cycle of a servlet. A method is also provided that allows a servlet to obtain any initialization parameters. The init( ), service( ), and destroy( ) methods are the life cycle methods of the servlet. These are invoked by the server. The getServletConfig( ) method is called by the

servlet to obtain initialization parameters. A servlet developer overrides the getServletInfo( ) method to provide a string with useful information (for example, author, version, date, copyright). This method is also invoked by the server.

The ServletConfig interface allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are summarized here:

Method	Description
ServletContext getServletContext( )	Returns the context for this servlet.
String getInitParameter(String *param*)	Returns the value of the initialization parameter named *param*.
Enumeration getInitParameterNames( )	Returns an enumeration of all initialization parameter names.
String getServletName( )	Returns the name of the invoking servlet.

Table 2.3: ServletConfig interface methods

2.5 SERVLETREQUEST INTERFACE

The ServletRequest interface enables a servlet to obtain information about a client request. Several of its methods are summarized in Table 2.4.

Method	Description
Object getAttribute(String *attr*)	Returns the value of the attribute named *attr*.
String getCharacterEncoding( )	Returns the character encoding of the request.
int getContentLength( )	Returns the size of the request. The value –1 is returned if the size is unavailable.
String getContentType( )	Returns the type of the request. A **null** value is returned if the type cannot be determined.
ServletInputStream getInputStream( ) throws IOException	Returns a **ServletInputStream** that can be used to read binary data from the request. An **IllegalStateException** is thrown if **getReader( )** has already been invoked for this request.
String getParameter(String *pname*)	Returns the value of the parameter named *pname*.
Enumeration getParameterNames( )	Returns an enumeration of the parameter names for this request.
String[ ] getParameterValues(String *name*)	Returns an array containing values associated with the parameter specified by *name*.
String getProtocol( )	Returns a description of the protocol.
BufferedReader getReader( ) throws IOException	Returns a buffered reader that can be used to read text from the request. An **IllegalStateException** is thrown if **getInputStream( )** has already been invoked for this request.
String getRemoteAddr( )	Returns the string equivalent of the client IP address.
String getRemoteHost( )	Returns the string equivalent of the client host name.
String getScheme( )	Returns the transmission scheme of the URL used for the request (for example, "http", "ftp").
String getServerName( )	Returns the name of the server.
int getServerPort( )	Returns the port number.

Table 2.4: ServletRequest interface methods

2.6 SERVLETRESPONSE INTERFACE

The ServletResponse interface enables a servlet to formulate a response for a client. Several of its methods are summarized in Table 2.5.

Method	Description
String getCharacterEncoding( )	Returns the character encoding for the response.
ServletOutputStream getOutputStream( ) throws IOException	Returns a **ServletOutputStream** that can be used to write binary data to the response. An **IllegalStateException** is thrown if **getWriter( )** has already been invoked for this request.
PrintWriter getWriter( ) throws IOException	Returns a **PrintWriter** that can be used to write character data to the response. An **IllegalStateException** is thrown if **getOutputStream( )** has already been invoked for this request.
void setContentLength(int *size*)	Sets the content length for the response to *size*.
void setContentType(String *type*)	Sets the content type for the response to *type*.

Table 2.5: ServletResponse Interface methods

2.7 GENERICSERVLET CLASS

The GenericServlet class provides implementations of the basic life cycle methods for a servlet. GenericServlet implements the Servlet and ServletConfig interfaces. In addition, a method to append a string to the server log file is available. The signatures of this method are shown here: void log(String s) void log(String s, Throwable e) Here, s is the string to be appended to the log, and e is an exception that occurred.A generic servlet is a protocol independent Servlet that should always override the service() method to handle the client request. The service() method accepts two arguments ServletRequest object and ServletResponse object. The request object tells the servlet about the request made by client while the response object is used to return a response back to the client.

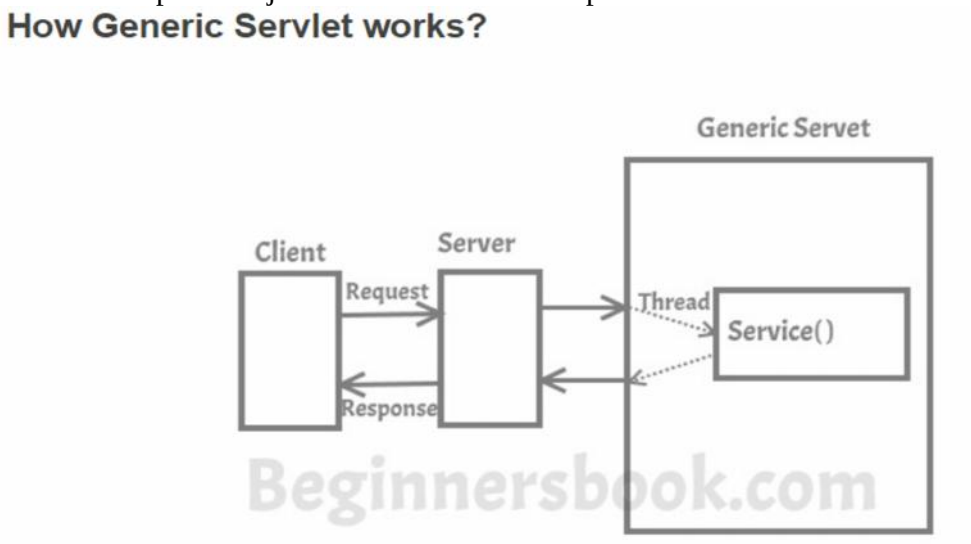


Figure 2.1: Generic Servlet

2.8 SERVLETINPUTSTREAM CLASS

The ServletInputStream class extends InputStream. It is implemented by the servlet container and provides an input stream that a servlet developer can use to read the data from a client request. It defines the default constructor. In addition, a method is provided to read bytes from the stream. It is shown here: int readLine(byte[ ] buffer, int offset, int size) throws IOException Here, buffer is the array into which size bytes are placed starting at offset. The method returns the actual number of bytes read or –1 if an end-of-stream condition is encountered.

2.9 SERVLETOUTPUTSTREAM CLASS

The ServletOutputStream class extends OutputStream. It is implemented by the servlet container and provides an output stream that a servlet developer can use to write data to a client response. A default constructor is defined. It also defines the print( ) and println( ) methods, which output data to the stream.

2.10 REQUESTDISPATCHER INTERFACE:

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. It is one of the way of servlet collaboration.

There are two methods defined in the **RequestDispatcher** interface.

2.10.1 Methods of RequestDispatcher interface:

The RequestDispatcher interface provides two methods. They are:

1. **public void forward(ServletRequestrequest,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **public void include(ServletRequestrequest,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

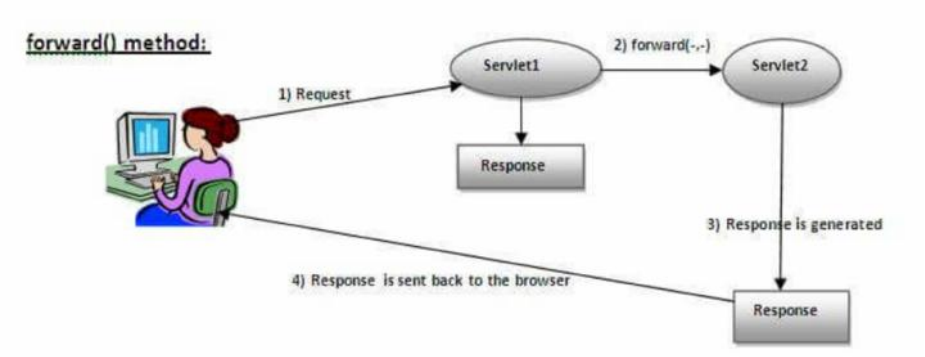


Figure 2.2: forward() method

As we see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.

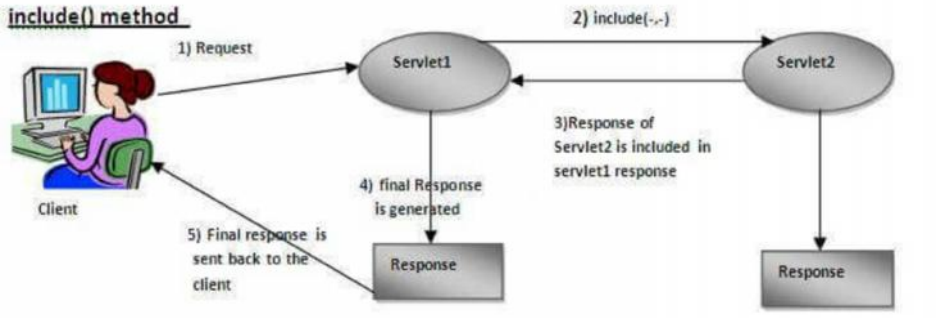


Figure 2.3: include() method

As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.

2.11 HTTPSERVLET CLASS

The HttpServlet class extends GenericServlet. It is commonly used when developing servlets that receive and process HTTP requests. The methods of the HttpServlet class are summarized in Table 2.6

Method	Description
void doDelete(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP DELETE request.
void doGet(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP GET request.
void doHead(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP HEAD request.
void doOptions(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP OPTIONS request.
void doPost(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP POST request.
void doPut(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP PUT request.
void doTrace(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP TRACE request.

long getLastModified(HttpServletRequest *req*)	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the requested resource was last modified.
void service(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Called by the server when an HTTP request arrives for this servlet. The arguments provide access to the HTTP request and response, respectively.

Table 2.6: HttpServlet class methods

2.12 HTTPSERVLETREQUEST

The HttpServletRequest interface enables a servlet to obtain information about a client request. Several of its methods are shown in Table 2.7

Method	Description
String getAuthType( )	Returns authentication scheme.
Cookie[ ] getCookies( )	Returns an array of the cookies in this request.
long getDateHeader(String *field*)	Returns the value of the date header field named *field*.
String getHeader(String *field*)	Returns the value of the header field named *field*.
Enumeration getHeaderNames( )	Returns an enumeration of the header names.
int getIntHeader(String *field*)	Returns the **int** equivalent of the header field named *field*.
String getMethod( )	Returns the HTTP method for this request.
String getPathInfo( )	Returns any path information that is located after the servlet path and before a query string of the URL.
String getPathTranslated( )	Returns any path information that is located after the servlet path and before a query string of the URL after translating it to a real path.
String getQueryString( )	Returns any query string in the URL.
String getRemoteUser( )	Returns the name of the user who issued this request.
String getRequestedSessionId( )	Returns the ID of the session.
String getRequestURI( )	Returns the URI.
StringBuffer getRequestURL( )	Returns the URL.
String getServletPath( )	Returns that part of the URL that identifies the servlet.
HttpSession getSession( )	Returns the session for this request. If a session does not exist, one is created and then returned.
HttpSession getSession(boolean *new*)	If *new* is **true** and no session exists, creates and returns a session for this request. Otherwise, returns the existing session for this request.
boolean isRequestedSessionIdFromCookie( )	Returns **true** if a cookie contains the session ID. Otherwise, returns **false**.
boolean isRequestedSessionIdFromURL( )	Returns **true** if the URL contains the session ID. Otherwise, returns **false**.
boolean isRequestedSessionIdValid( )	Returns **true** if the requested session ID is valid in the current session context.

Table 2.7: HttpServletRequest

2.13 HTTPSERVLETRESPONSE INTERFACES

Method	Description
void addCookie(Cookie *cookie*)	Adds *cookie* to the HTTP response.
boolean containsHeader(String *field*)	Returns **true** if the HTTP response header contains a field named *field*.
String encodeURL(String *url*)	Determines if the session ID must be encoded in the URL identified as *url*. If so, returns the modified version of *url*. Otherwise, returns *url*. All URLs generated by a servlet should be processed by this method.
String encodeRedirectURL(String *url*)	Determines if the session ID must be encoded in the URL identified as *url*. If so, returns the modified version of *url*. Otherwise, returns *url*. All URLs passed to **sendRedirect( )** should be processed by this method.

Table 2.8(a): HttpServletResponse interface methods

Method	Description
void sendError(int *c*) throws IOException	Sends the error code *c* to the client.
void sendError(int *c*, String *s*) throws IOExcception	Sends the error code *c* and message *s* to the client.
void sendRedirect(String *url*) throws IOException	Redirects the client to *url*.
void setDateHeader(String *field*, long *msec*)	Adds *field* to the header with date value equal to *msec* (milliseconds since midnight, January 1, 1970, GMT).
void setHeader(String *field*, String *value*)	Adds *field* to the header with value equal to *value*.
void setIntHeader(String *field*, int *value*)	Adds *field* to the header with value equal to *value*.
void setStatus(int *code*)	Sets the status code for this response to *code*.

Table 2.8(b): HttpServletResponse interface methods

2.14 HTTPSESSION INTERFACE

The HttpSession interface enables a servlet to read and write the state information that is associated with an HTTP session. Several of its methods are summarized in Table 2.9. All of these methods throw an IllegalStateException if the session has already been invalidated.

Method	Description
Object getAttribute(String *attr*)	Returns the value associated with the name passed in *attr*. Returns **null** if *attr* is not found.
Enumeration getAttributeNames( )	Returns an enumeration of the attribute names associated with the session.
long getCreationTime( )	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when this session was created.
String getId( )	Returns the session ID.
long getLastAccessedTime( )	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the client last made a request for this session.
void invalidate( )	Invalidates this session and removes it from the context.
boolean isNew( )	Returns **true** if the server created the session and it has not yet been accessed by the client.
void removeAttribute(String *attr*)	Removes the attribute specified by *attr* from the session.
void setAttribute(String *attr*, Object *val*)	Associates the value passed in *val* with the attribute name passed in *attr*.

Table 2.9: HttpSession methods

2.15 SERVLET LIFECYCLE

Three methods are central to the life cycle of a servlet. These are init( ), service( ), and destroy( ). They are implemented by every servlet and are invoked at specific times by the server. Let us consider a typical user scenario to understand when these methods are called. First, assume that a user enters a Uniform Resource Locator (URL) to a web browser. The browser then generates an HTTP request for this URL. This request is then sent to the appropriate server. Second, this HTTP request is received by the web server. The server maps this request to a particular servlet. The servlet is dynamically retrieved and loaded into the address space of the server. Third, the server invokes the init( ) method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure

itself. Fourth, the server invokes the service( ) method of the servlet. This method is called to process the HTTP request. You will see that it is possible for the servlet to read data that has been provided in the HTTP request. It may also formulate an HTTP response for the client. The servlet remains in the server's address space and is available to process any other HTTP requests received from clients. The service( ) method is called for each HTTP request. Finally, the server may decide to unload the servlet from its memory. The algorithms by which this determination is made are specific to each server. The server calls the destroy( ) method to relinquish any resources such as file handles that are allocated for the servlet. Important data may be saved to a persistent store. The memory allocated for the servlet and its objects can then be garbage collected.

2.16 ORGANIZATION OF A WEB APPLICATION

Before we see how the servlet works, let's get familiar with these three terms.

Web Server: it can handle HTTP Requests send by clients and responds the request with an HTTP Response.

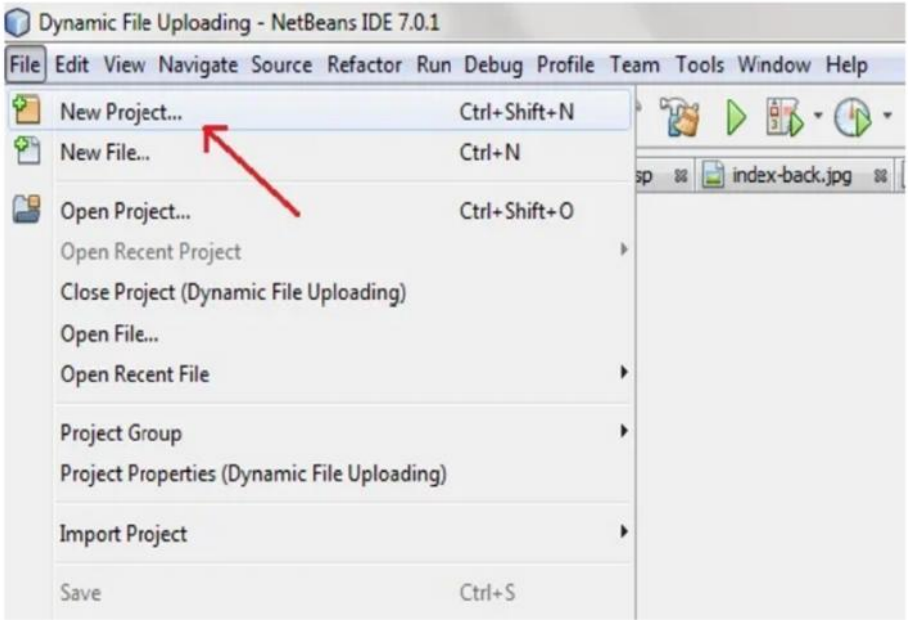
Web Application(webapp): I would refer this as webapp in this guide. Basically, the project is your web application, it is the collection of servlets.

Web Container: Also known as Servlet Container and Servlet Engine. It is a part of Web Server that interacts with Servlets. This is the main component of Web Server that manages the life cycle of Servlets.

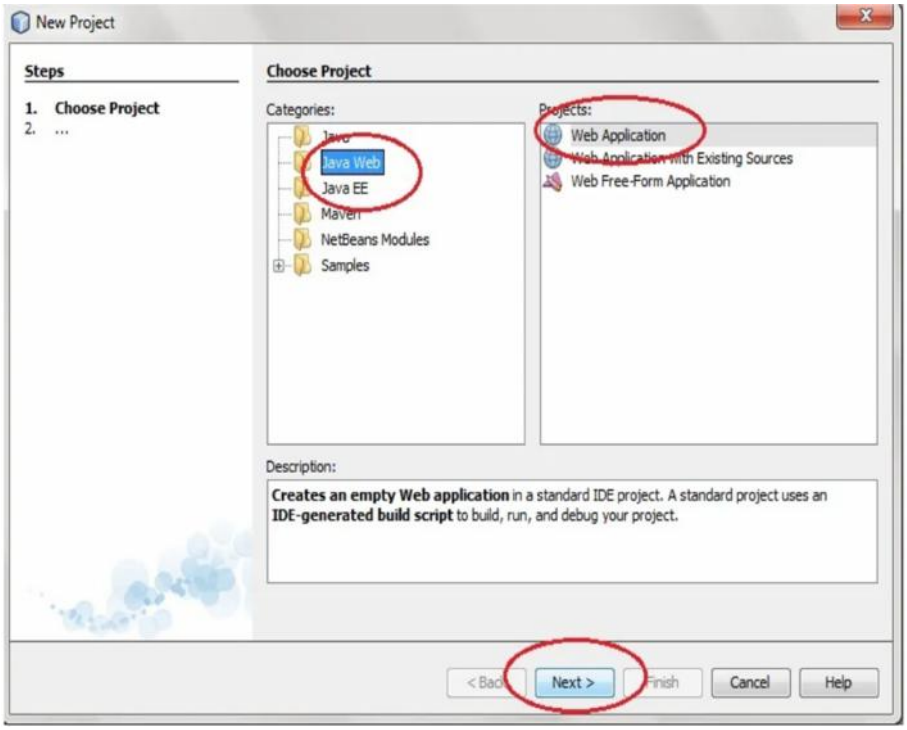
2.17 CREATING A WEB APPLICATION (USING NETBEANS)

To create a servlet application in Netbeans IDE, you will need to follow the following (simple) steps:

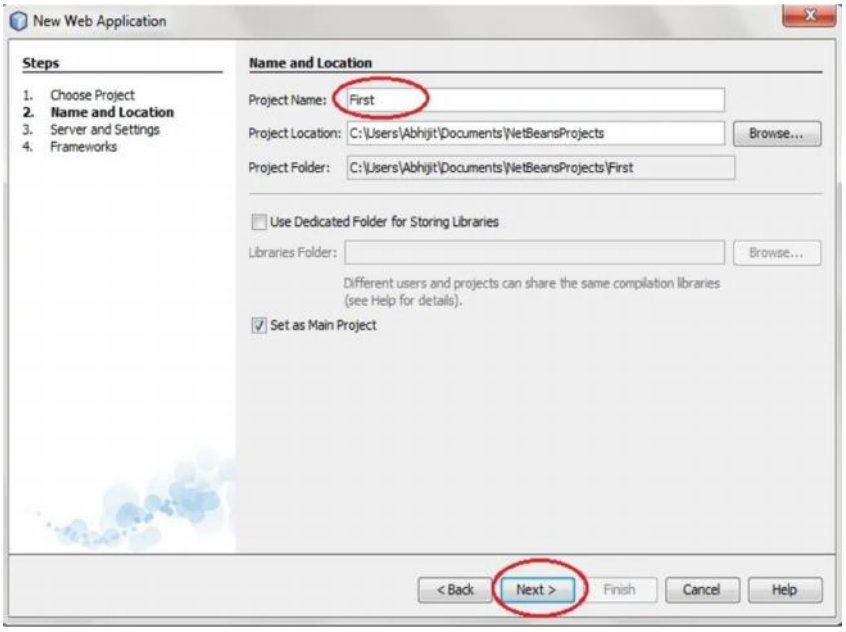
1. Open Netbeans IDE, Select File ->**New Project**



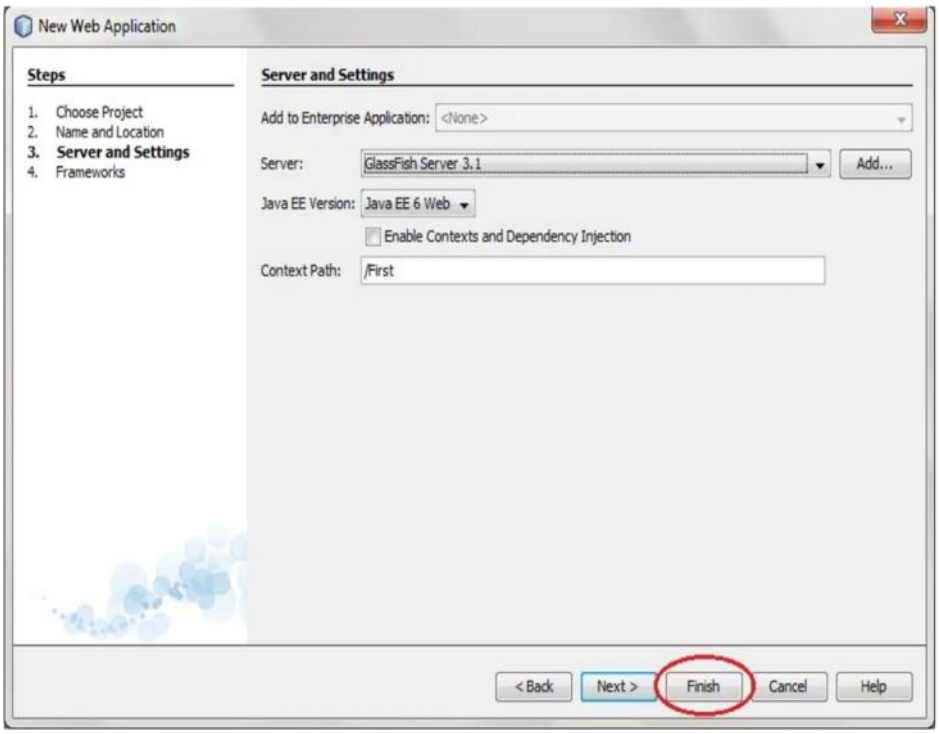
2. Select **Java Web** -> **Web Application**, then click on Next.



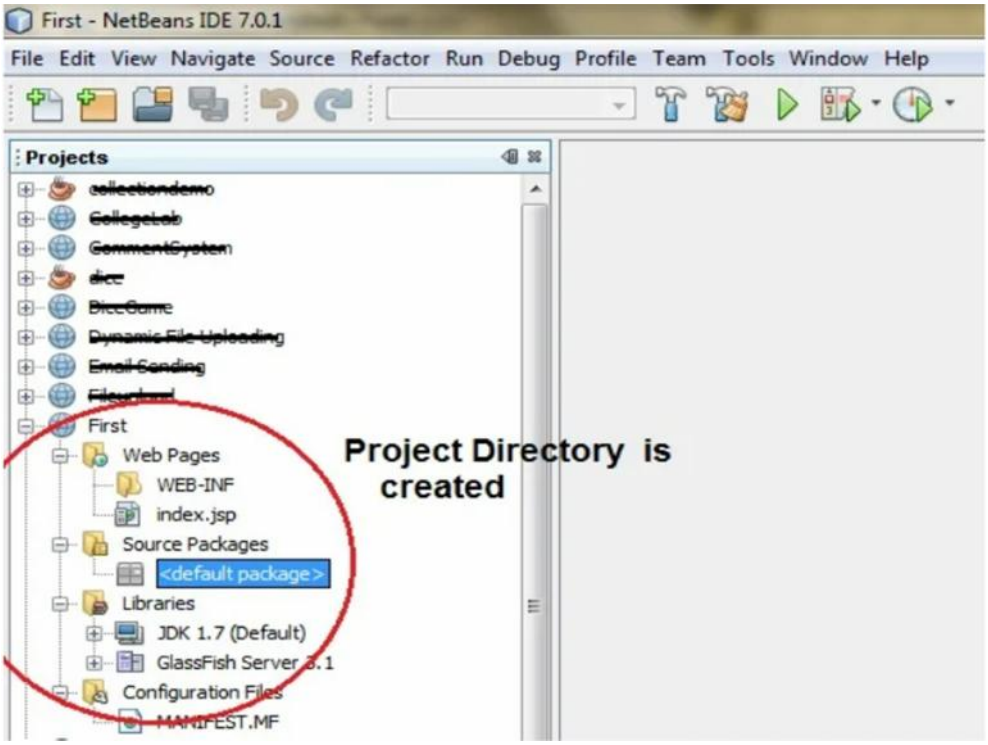
3. Give a name to your project and click on Next.



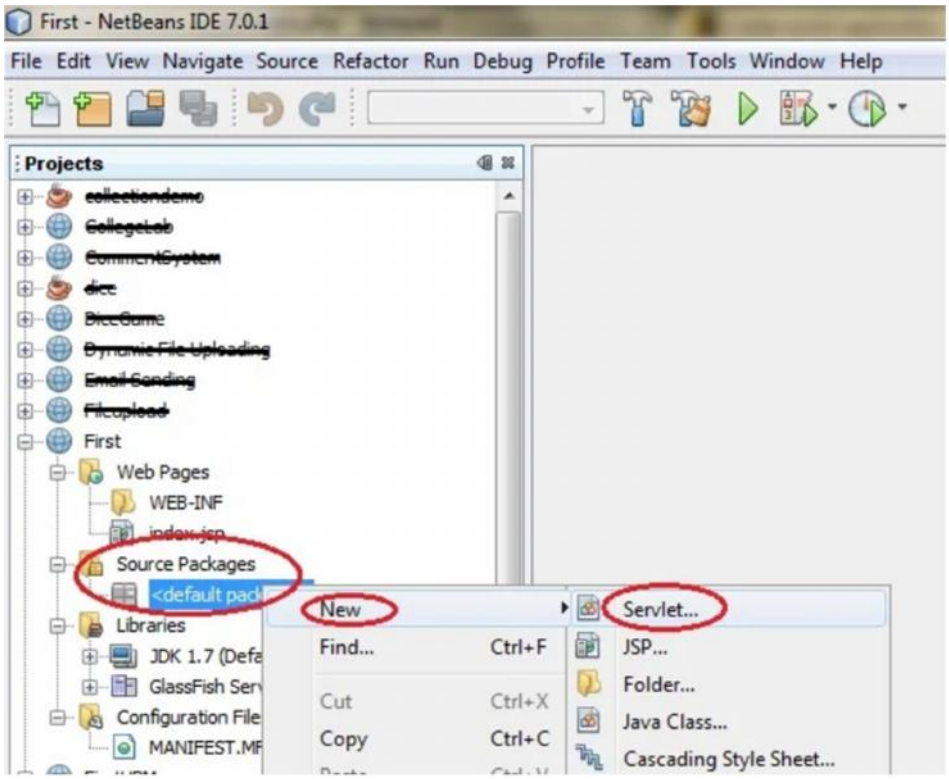
4. and then, Click **Finish**



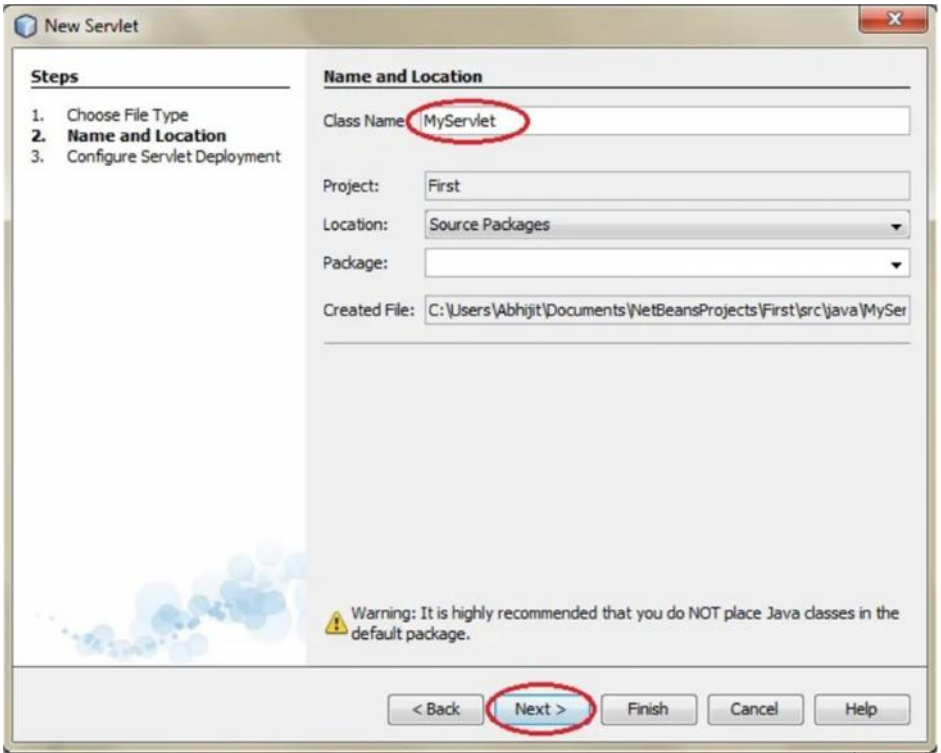
5. The complete directory structure required for the Servlet Application will be created automatically by the IDE.

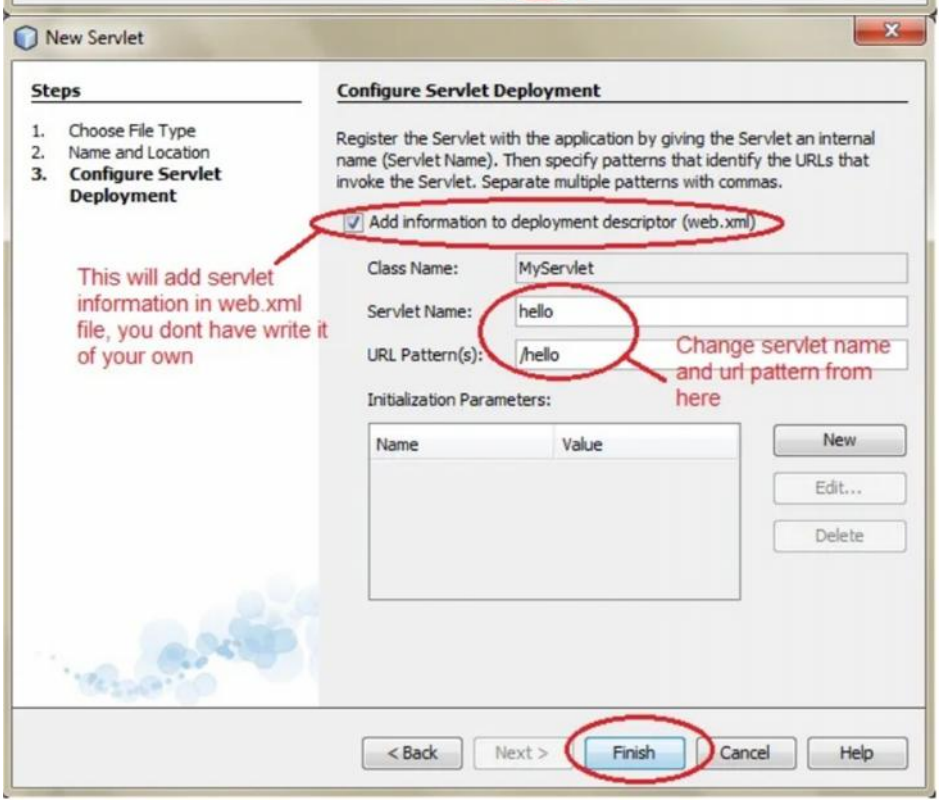


6. To create a Servlet, open-Source **Package**, right click on **default packages** -> **New** -> **Servlet**.

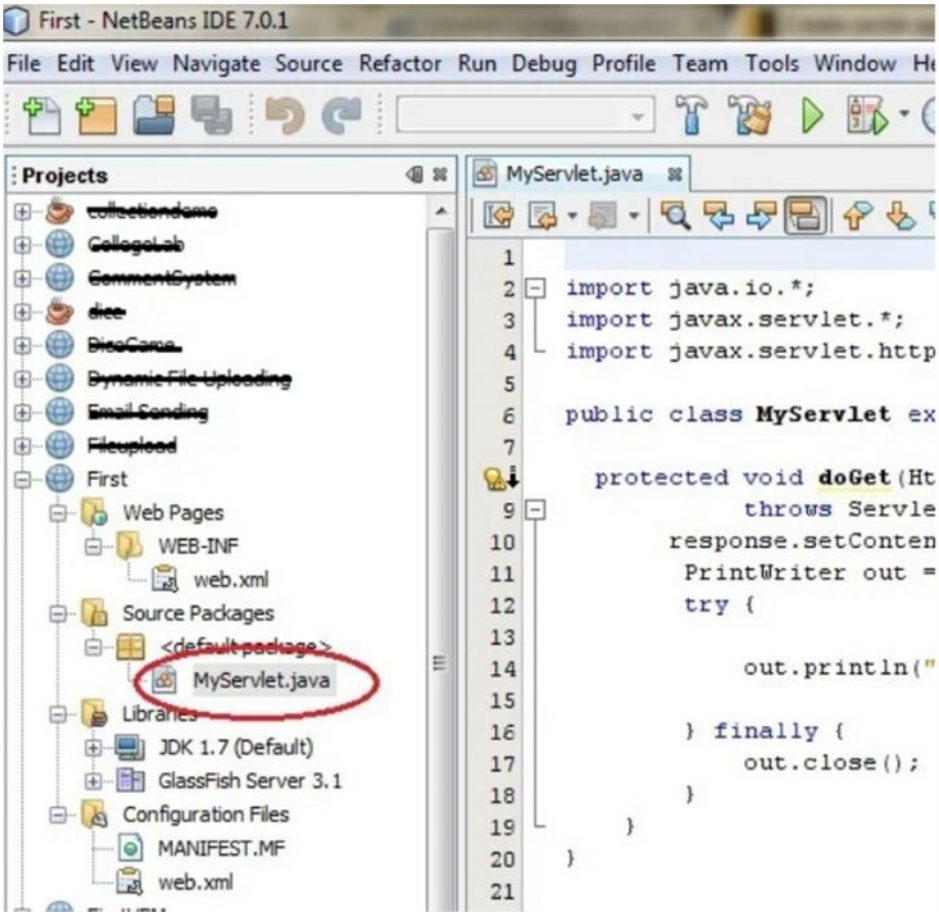


7. Give a Name to your Servlet class file.

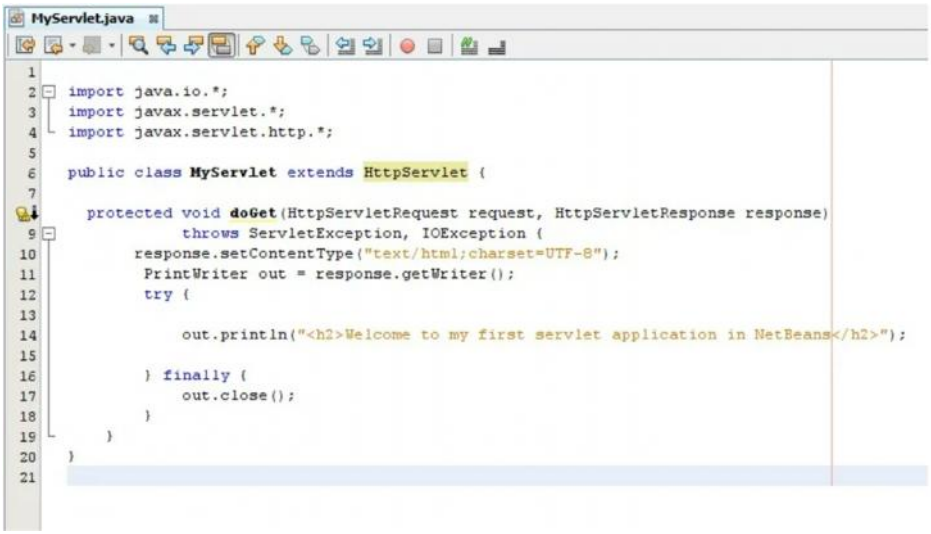




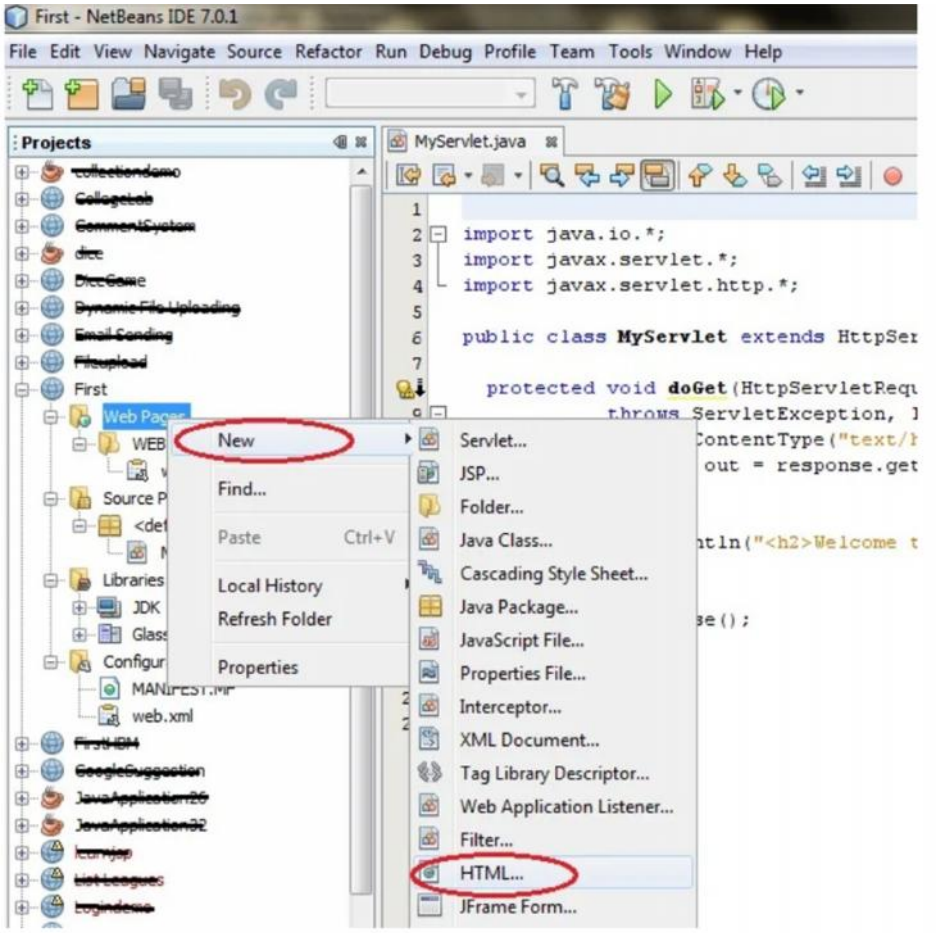
8. Now, your Servlet class is ready, and you just need to change the method definitions and you will good to go.



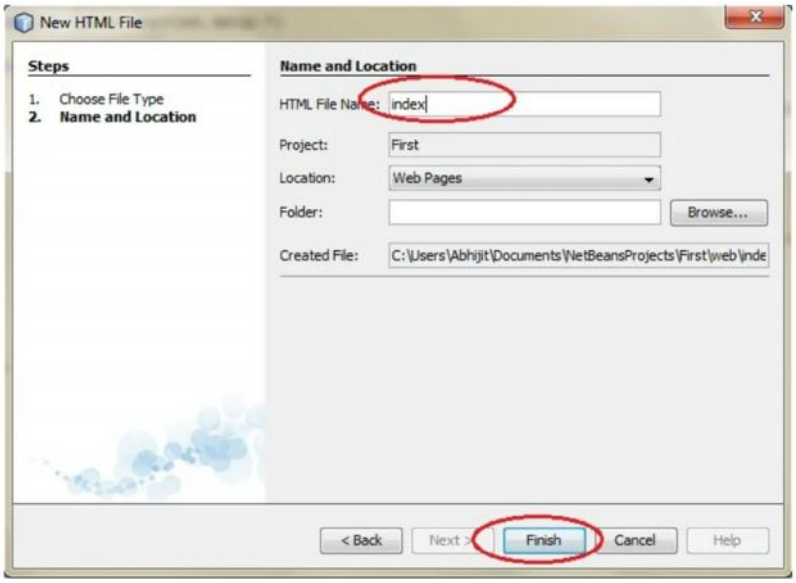
9. Write some code inside your Servlet class.



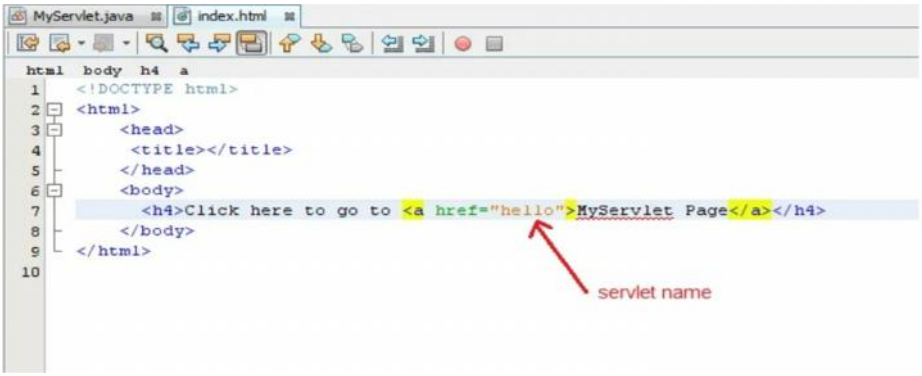
10. Create an HTML file, right click on **Web Pages** -> **New** -> **HTML**



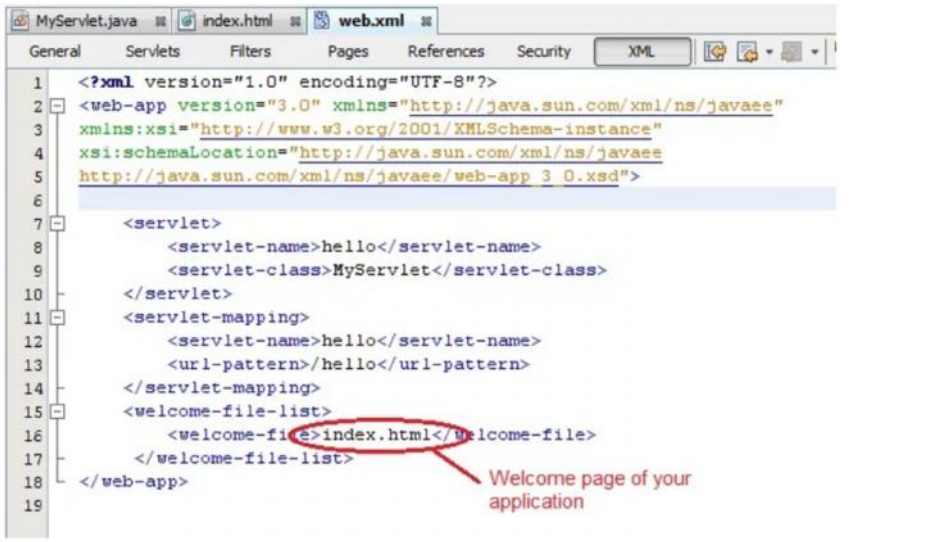
11. Give it a name. We recommend you to name it index, because browser will always pick up the index.html file automatically from a directory. Index file is read as the first page of the web application.



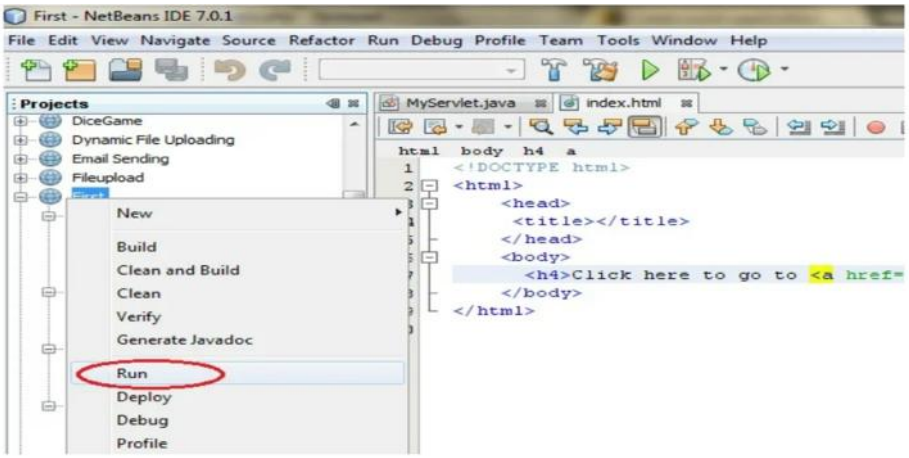
12. Write some code inside your HTML file. We have created a hyperlink to our Servlet in our HTML file.



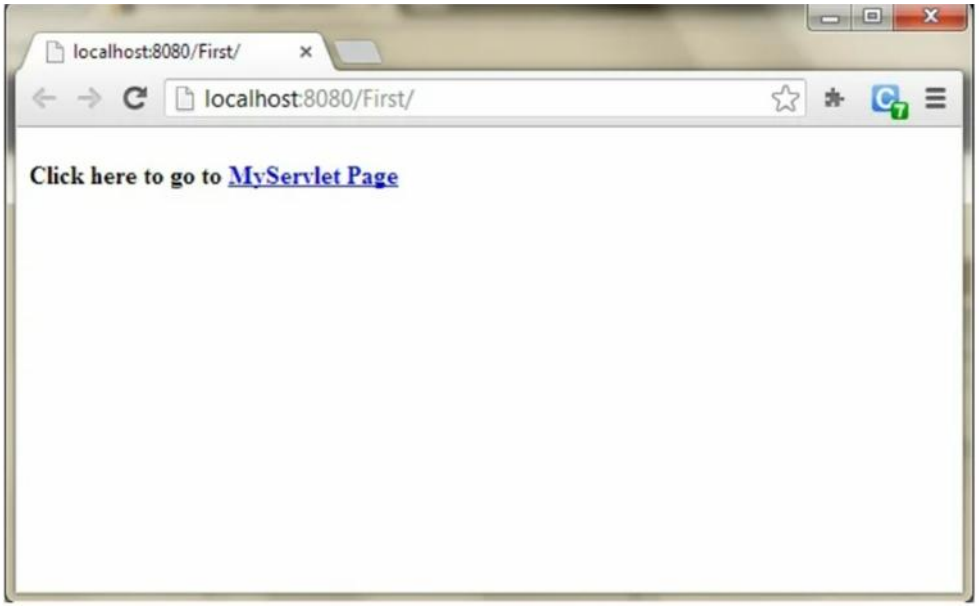
13. Edit **web.xml** file. In the web.xml file you can see, we have specified the **url-pattern** and the **servlet-name**, this means when hello url is accessed our Servlet file will be executed.



14. Run your application, right click on your Project and select **Run.**



15. Click on the link created, to open your Servlet.



UNIT END QUESTIONS

1. Why there is need of Servlets?
2. Explain Servlet lifecycle?
3. Explain the difference between GenericServlet and HttpServlet?

REFERENCES

- Java EE 6 for Beginners, Sharanam Shah, Vaishali Shah, SPD
- https://www.studytonight.com/servlet/creating-servlet-in-netbeans.php

*****

UNIT III

3

JAVA DATABASE CONNECTIVITY

Unit Structure

3.0 OBJECTIVE

- To understand the structure of java database (JDBC)
- To learn design of JDBC database
- To learn how to configure JDBC with database
- To understand the execution of database query and retrieval of data from the database using JDBC.

3.1 INTRODUCTION

JDBC is a java database connectivity standard that provides the interface for connecting from Java to relational databases. The JDBC standard is defined by early Sun Microsystems and implemented through the standard java.sql interfaces. This permits individual providers to implement and extend the standard with their own JDBC drivers.

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a broad range of databases.

The JDBC library includes APIs for each of the tasks commonly associated with database usage:

- Establish a connection to a database
- Creating SQL or MySQL statements
- Executing that SQL or MySQL queries in the database
- Viewing & Modifying the resulting records using resultset.

JDBC API is a Java programming API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on different platforms, such as Windows, Mac OS, and the various versions of Unix/Linux.

3.3 DESIGN OF JDBC

Java language was designed to provide platform independence from hardware/software platforms, so too JDBC has been designed to provide some degree of database independence for java developers. JDBC is designed to provide a database API for accessing relational databases from different vendors. JDBC developed to work with the most common type of databases, JDBC drivers that allow the API to be used to connect to both high-end, mainframe databases.

The relationships between the database objects are described using a query language, the most popular of which is the Structured Query Language (SQL).

JDBC Architecture:

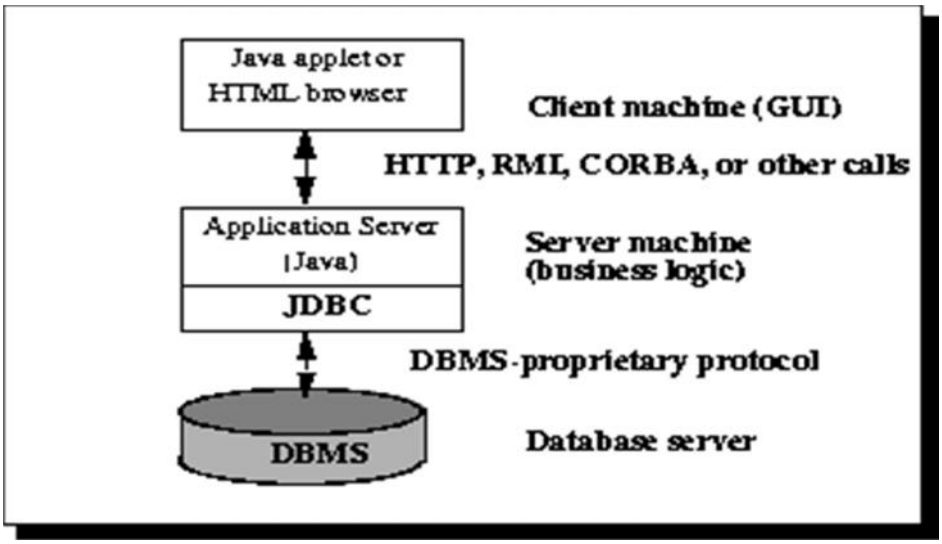


Fig. 1 JDBC Architecture

- The JDBC interface supports both two-tier and three-tier processing models for database access but in general JDBC Architecture consists of two layers:
- JDBC API: This provides the application-to-JDBC Manager connection.

- JDBC driver supports the JDBC Manager-to-Driver Connection.
- The JDBC API connects with driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- The JDBC driver manager ensures that the right driver is used to access each data source. The driver manager is supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Common JDBC Components:
The JDBC API provides the following interfaces and classes:

- DriverManager:
  - Manages a list of database drivers.
  - Matches connection requests with the proper database driver
  - Establish a database Connection.
- Driver:
  - Handles the communications with the database server.
  - DriverManager to manage objects
- Connection :
  - This interface with all methods for contacting a database.
  - Connection object represents the communication context
- Statement :
  - This interface to submit the SQL statements to the database.
  - Interfaces accept parameters in addition to executing stored procedures.
- ResultSet:
  - These objects hold data retrieved from a database after execute an SQL query using Statement objects.
  - It acts as an iterator.
- SQLException:
  - This class handles any errors that occur in a database application.

3.4 JDBC CONFIGURATION

- In JDBC configuration JDBC drivers implement the defined interfaces in the JDBC API for interacting with the database server. For example, using JDBC drivers to enable open database connections and to interact with databases by sending SQL or database commands then receiving results with Java.
- The Java.sql package with JDK contains various classes with their methods defined and their actual implementations are done in third-party drivers. Third party vendors implement the java.sql.driver interface in their database driver.

JDBC drivers are divided into four types or levels. The different types of jdbc drivers are:

1. Type 1: JDBC-ODBC Bridge driver (Bridge)
2. Type 2: Native-API/partly Java driver (Native)
3. Type 3: JDBC Network-All JAVA driver (Middleware)
4. Type 4: All Java/Native-protocol driver (Pure)

JDBC-ODBC Bridge driver:

- Type 1 drivers act as a "bridge" between JDBC and another database connectivity mechanism such as ODBC. The JDBC- ODBC bridge provides JDBC access using most standard ODBC drivers. This driver is included in the Java 2 SDK within the sun.jdbc.odbc package.
- In this driver the java statements are converted to a jdbc statements. JDBC statements call the ODBC by using the JDBC-ODBC Bridge. And finally the query is executed by the database.
- The Type1 driver translates all JDBC calls into ODBC calls and sends them to the ODBC driver. ODBC is a generic API.

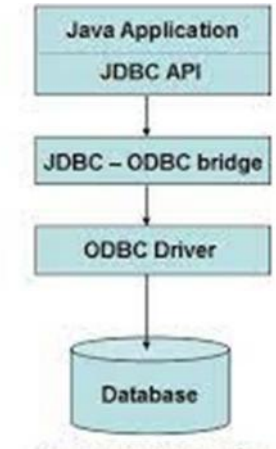


Fig. 2 Type 1: JDBC-ODBC Bridge Driver

Native-API/partly Java driver:

- Type 2 drivers use the Java Native Interface (JNI) to make calls to a local database library API. This driver converts the JDBC calls into a database specific call for databases such as SQL, ORACLE etc.
- This driver communicates directly with the database server. It requires some native code to connect to the database. Type 2 drivers are usually faster than Type 1 drivers.
- Like Type 1 drivers, Type 2 drivers require native database client libraries to be installed and configured on the client machine. The distinctive characteristic of type 2 jdbc drivers is that Type 2 drivers convert JDBC calls into database-specific calls i.e. this driver is specific to a particular database.

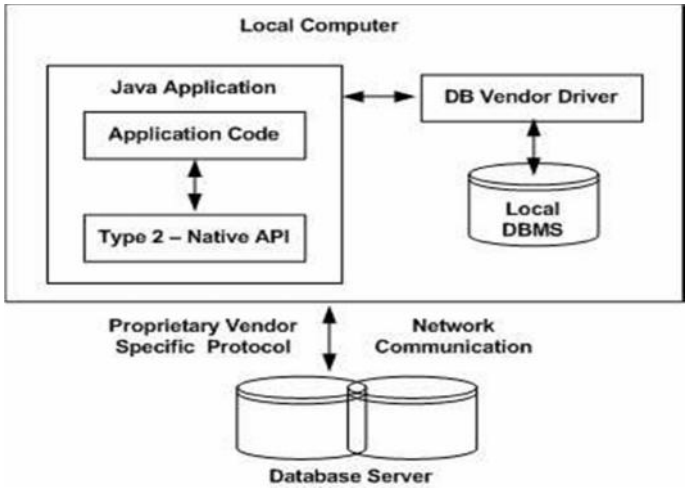


Fig. 3 Type 2: JDBC-Native API

All Java/Net-protocol driver:

- Type 3 drivers are pure Java drivers that use a proprietary network protocol to communicate with JDBC middleware on the server. The middleware then translates the network protocol to database-specific function calls.
- Type 3 drivers are the most flexible JDBC solution because they do not require native database libraries on the client and can connect to many different databases on the back end. Type 3 drivers can be deployed over the Internet without client installation.
- Java-------> JDBC statements------> SQL statements ------> databases.

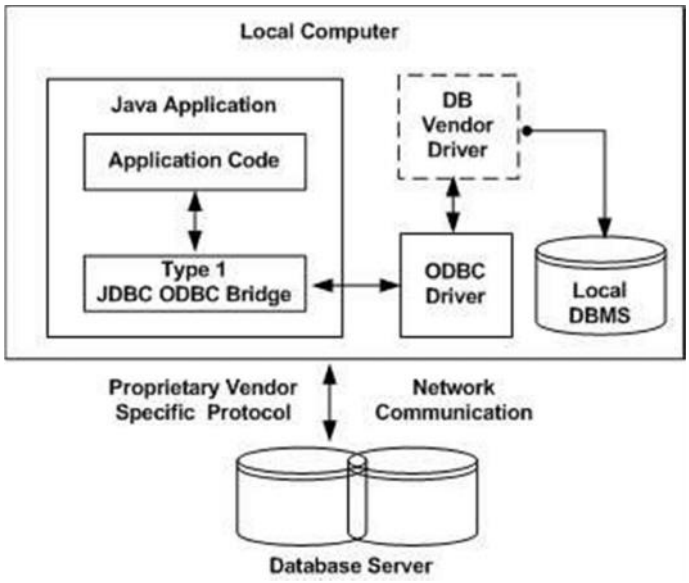


Fig. 4 Type 3: JDBC-Net pure Java

Native-protocol/all-Java driver:
The Type 4 uses java networking libraries to communicate directly with the database server.

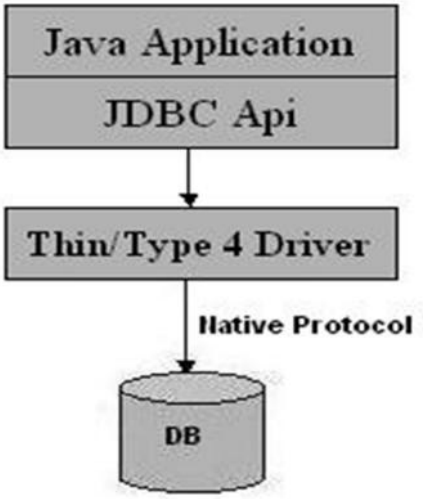


Fig. 5 Type 4: 100% pure Java

The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps:

- **Import JDBC Packages:** Add import statements to Java program to import required classes in Java code.
- **Register with JDBC Driver:** This step causes the JVM to load the desired driver implementation into memory so it can fulfill JDBC requests.
- **Database URL expression:** This is to create a properly formatted address that points to the database to which wish to connect.
- **Create Connection Object:** Finally, code a call to the *DriverManager* object's *getConnection( )* method to establish actual database connection.

Import JDBC Packages:

The Import statements tell the Java compiler where to find the classes reference in r code and are placed at the very beginning of r source code.

import java.sql.* ; // for standard JDBC programs

Register JDBC Driver:

To register the JDBC driver in r program before using it. Registering the driver is the process by which the Oracle driver's class file is loaded into memory so it can be utilized as an implementation of the JDBC interfaces.

Class.forName():

The most common approach to register a driver is to use Java's Class.forName() method to dynamically load the driver's class file into

memory, which automatically registers it. This method is preferable because it allows to make the driver registration configurable and portable. The following example uses Class.forName( ) to register the Oracle driver:

```
try
{
        Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex)
{
                System.out.println("Error: unable to load driver class!");
                System.exit(1);
}
```

DriverManager.registerDriver():

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

The following example uses registerDriver() to register the Oracle driver:

```
try {
Driver myDriver = new oracle.jdbc.driver.OracleDriver();
DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
System.out.println("Error: unable to load driver class!");
System.exit(1);
}
```

Database URL Formulation:

After loading the driver, JDBC establishes a connection using the DriverManager.getConnection() method.

The three overloaded DriverManager.getConnection() methods:

- getConnection(String url)
- getConnection(String url, Properties prop)
- getConnection(String url, String user, String password)

Following table lists down popular JDBC driver names and database URLs.

RDBMS	**JDBC driver name**	**URL format**
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@hostname:port Number:databaseName

DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:hostname: port Number/databaseName

Create Connection Object:

Using a database URL with a username and password:

- There are three forms of DriverManager.getConnection() method to create a connection object. The most commonly used form of getConnection() requires to pass a database URL, a *username*, and a *password*:

  host:port:databaseName value for the database portion of the URL.

  Example, jdbc:oracle:thin:@amrood:1521:EMP

- to call getConnection() method with appropriate username and password to get a Connection object as follows:

  String URL = "jdbc:oracle:thin:@amrood:1521:EMP";

  String USER = "username";

  String PASS = "password"

  Connection conn = DriverManager.getConnection(URL, USER, PASS);

- Using only a database URL:

A second form of the DriverManager.getConnection( ) method requires only a database URL:

DriverManager.getConnection(String url);

Example, String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";

Connection conn = DriverManager.getConnection(URL);

```
import java.util.*;
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
Properties info = new Properties( );
info.put( "user", "username" );
info.put( "password", "password" );
Connection conn = DriverManager.getConnection(URL);
```

Closing JDBC connections:

At the end of JDBC program, it is required to explicitly close all the connections to the database to end each database session. .

To close above opened connection should call close() method as follows:

conn.close();

3.5 EXECUTING SQL STATEMENT, QUERY EXECUTION

There are 3 objects in JDBC

- Statement Object.
- PreparedStatement Object
- CallableStatement Object

Statement Class:

- For sending SQL Statements, JDBC uses the executeQuery (String SQL) method of this Statement Class.
- The execute (String SQL) method which will return a boolean value, whether the same have been executed or not. This is normally used when the statement returns more than one Result Set.
- We can also use the executeUpdate (String SQL) method, which will return an int value, which the number of rows updated (that is inserted, deleted, or modified).
- JDBC provides two kinds of objects that can be used to execute SQL Statements and they are PreparedStatement and CallableStatement interfaces which are sub-interfaces of the Statement Interface. The PreparedStatement Interface extends the Statement Interface and the CallableStatement Interface extends the PreparedStatement interface.
- PreparedStatement objects differ from the Statement objects in that the SQL statement is pre-compiled and can have placeholders (?) for runtime parameters values.
- The PreparedStatement objects are particularly useful when a statement will be executed many times (for example, adding new rows) since substantial performance gains can be achieved.

Example:

```
import java.sql.*;
class DBTest
{
  public static void main(String[] args)
        {
        try
        {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection c =
DriverManager.getConnection("jdbc:odbc:dum");
        Statement s = c.createStatement();
        s.execute("create table DeptMaster(deptno Number, dname
Text,   loc Text)");
        System.out.println("Table successfully created");
```

```
                }
        catch(SQLException e)
        {
                System.out.println("Datebase Error"+e.getMessage());
        }
        catch(Exception e)
        {
                System.out.println("General Error"+e.getMessage());
        }
  }
}
```

Creating and Executing SQL statements

The commonly used SQL statements are:

1) Select
2) Insert
3) Update
4) Delete

SQL Select statement:

- The SELECT statement is used to select data from a table.

  Syntax: Select column_names FROM table_name;
- The result from a SQL query is stored in a resultset.
- The Select specifies the table columns that are retrieved.
- The From clause tells from where the table has been accessed.
- The Where clause specifies which tables are used. The Where clause is optional, if not used then all the table rows will be selected.

SQL INSERT Statement:

- This statement allows to insert a single or multiple records into the database.
- We can specify the name of the column in which we want to insert the data.

Syntax: Insert into table_name values (value1, value2..);

We can also specify the columns for which we want to insert data.

The UPDATE Statement:

- The Update statement is used to modify the data in the table.
- Whenever we want to update or delete a row then we use the Update statement.

The syntax is:

UPDATE table_name Set colunm_name = new_value WHERE column_name = some_name;

The Update statement has mainly three clauses.

1) UPDATE: It specifies which table column has to be updated.
2) Set: It sets the column in which the data has to be updated.
3) Where: It tells which tables are used.

SQL DELETE Statement:

- This delete statement is used to delete rows in a table.
  Syntax: DELETE FROM table_name WHERE column_name = some_name;

Scrollable and updatable result sets:

1. scrollable ResultSets:

- These ResultSet objects will allow the users to interact with the data in both forward and backward directions.
- Scrollable ResultSets can be divided into following two types.

  a) ScrollSensitive ResultSet:
  - It is a ResultSet object, it will allow the later database modifications.
  - To represent this ResultSet object we have to use the following constant from the ResultSet interface.

    Public static final TYPE_SCROLL_SENSITIVE

  b) ScrollInSensitive ResultSet:
  - These are scrollable ResultSet objects, which will allow the later database modifications after creation.
  - To represent this ResultSet object we have to use the following constant from the ResultSet interface.

    Public static final TYPE_SCROLL_INSENSITIVE

on the basis of ResultSet concurrency there are two types of ResultSets

1. Read only ResultSet:

This ResultSet will allow the users only to read the data.To represent this ResultSet object we have to use the following constant from the ResultSet interface.

Public static final int CONCUR_READ_ONLY

2. Updatable ResultSet:

This ResultSet object will allow the user to perform updates on it's content.

To represent this ResultSet object to use following constant from ResultSet interface.

Public static final int CONCUR_UPDATABLE.

- To refresh the present report a scroll sensitive ResultSet object we have to use the following method.

  Public void refreshRow() throws SQLException

- In case of scrollable ResultSet object to move ResultSet cursor before first record position we have to use following method

  public void beforeFirst()

- To move ResultSet cursor after last record position we have to use following method

  public void afterLast()

- To move ResultSet cursor to first record position we will use the following method

  public boolean first()

- To move ResultSet cursor to last record position we will use the following method

  public boolean last()

- To move ResultSet cursor to a particular record position we will use the following method

  public boolean absolute(int rec_position)

- To skip particular no.of records from the current position of the ResultSet we have to use the following method

  public boolean relative(int no_of_records)

- To insert new row in updatable ResultSet object we have to use following method

  public void moveToInsertRow()

- To insert record data temporarily in a row we have to use the following method

  public void updatexxx(int column_index,xxx value)

- In order to make temporary insertion as permanent insertion in the ResultSet object and database we have to use the following method.

  public void insertRow()

Example:

```
import java.sql.Connection;
        import java.sql.DriverManager;
```

```
        import java.sql.ResultSet;
        import java.sql.Statement;
        import java.util.Properties;
public class JdbcApp14 {
        public static void main(String args[])throws Exception
        {
        Class.forName("com.mysql.jdbc.Driver");
        Properties p=new Properties();
        Connection
        con=DriverManager.getConnection("jdbc:mysql://localhos
t:3306/test      ","root","system");

        Statement
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.      CONCUR_UPDATABLE);
        boolean b=st.execute("select * from emp1″);
        System.out.println(b);
        ResultSet rs=st.getResultSet();
        System.out.println("————————————");
        System.out.println("ENO   ENAME  ESAL");
        while(rs.next())
        {
        System.out.println(rs.getString("eno")+"
"+rs.getString("ename")       +"   "+rs.getString("esal"));
        }
        System.out.println("application in pausing state");
        System.out.println("perform updations at database");
        System.in.read();
        rs.beforeFirst();
        System.out.println("data after updations");
        System.out.println("————————————");
        System.out.println("ENO   ENAME  ESAL");
        System.out.println("————————————");
        while(rs.next())
        {
        rs.refreshRow();
        System.out.println(rs.getString("eno")+"
"+rs.getString("ename")       +"   "+rs.getString("esal"));
        }
        con.close();
        }
}
```

7. Row sets:

- A RowSet object is a java bean component and extends the ResultSet interface Thus, it has a set of JavaBeans properties and follows the JavaBeans event model.

- A RowSet object's properties allow it to create its own database connection and to execute its own query in order to fill itself with data.
- Types of Rowset

The RowSet is mainly classified into two types as per their properties:-

1. ***Connected Rowset.:-*** The connected rowset as the name suggests is connected to the database connection object like the resultset.
2. ***Disconnected RowSet:-*** The disconnected RowSet only connects to the database whenever needed and after finishing the communication they close the database connection. So if the connection pool is minimally used in this case.

- There are following ways to create the JDBCRowSet:-
  1. Passing the ResultSet:- In this type the data is populated in the object and then can retrieve the data by using the getter methods as we did in case of the ResultSet.

Example:-

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from employee");
JdbcRowSet jdbcRs = new JdbcRowSetImpl(rs);
```

2. Creating the default object:- This method is useful to set the data sources dynamically. In this method the *Database URL, username and password is explicitly set in the RowSetObject.*

Example:-

```
JdbcRowSet jdbcRs = new JdbcRowSetImpl();
 jdbcRs.setUsername("user");
 jdbcRs.setPassword("password");
 jdbcRs.setUrl("jdbc:mySubprotocol:mySubname");
 jdbcRs.setCommand("select * from EMP ");
 jdbcRs.execute();
```

- The following example will illustrate the basic operation using the JDBCRowSet interface.

```
import java.sql.SQLException;
import javax.sql.rowset.JdbcRowSet;
import com.sun.rowset.JdbcRowSetImpl;
public class JDBCRowSetExample
{
        public static void main(String[] args) throws
SQLException
{
```

```
            JdbcRowSet jdbcRs = new JdbcRowSetImpl();
            jdbcRs.setUsername("scott");
            jdbcRs.setPassword("tiger");
            jdbcRs.setUrl("jdbc:odbc:MyDsn");
            jdbcRs.setCommand("select * from employee");
            jdbcRs.execute();
            while(jdbcRs.next())
            {
                System.out.println(jdbcRs.getString("ename"));
            }
    }
}
```

8. Metadata:

- Data about the data is called metadata. In JDBC there are two types of metadata.

  1.Database metadata

  2.ResultSet metadata.

- JDBC MetaData is the collective information about the data structure and property of a column available in the table. The meta data of any table tells the name of the columns,datatype used in column and constraint used to enter the value of data into the column of the table.
- Loading a driver by calling a class.forname( ),this accepts the driver class as argument.

DriverManager.getConnection ( ) -This method returns a connection object and builds a connection between url and database. Once a connection is set up, a front end can access, insert ,update and retrieve the data in the backend database.

con.createStatement ( ) -This is used to create a sql object. An object con of the connection class is used to send and create a sql query in the database backend.

executeQuery ( ) -This method retrieves a record set from a table in the database. The retrieve record set is assigned to a result set object.

getMetaData ( ) - The Result Set call get Metadata( ),which returns the property of the retrieve record set (length,field,column).MetaData accounts for the data element and its attribute.

Getcolumncount ( ) -The method returns an integer data type and provides the number of columns in the Result set object.

Example:

JdbcMetaDataGettables.java

```
import java.sql.*;
public class JdbcMetaDataGettab
{
   static public final String driver = "com.mysql.jdbc.Driver";
   static public final String connection =
"jdbc:mysql://localhost:3306/test";
   static public final String user = "root";
   static public final String password = "root";
   public static void main(String args[])
{
      try
{
      Class.forName(driver);
      Connection con = DriverManager.getConnection(connection,
user,password);
      Statement st = con.createStatement();
      String sql = "select * from person";
      ResultSet rs = st.executeQuery(sql);
      ResultSetMetaData metaData = rs.getMetaData();
      int rowCount = metaData.getColumnCount();
      System.out.println("Table Name : " +metaData.getTableName(2));
      System.out.println("Field \tsize\tDataType");
      for (int i = 0; i < rowCount; i++)
{
   System.out.print(metaData.getColumnName(i + 1) + "\t");
   System.out.print(metaData.getColumnDisplaySize(i + 1)+"\t");
   System.out.println(metaData.getColumnTypeNa  me(i + 1));
}
   } catch (Exception e)
{
      System.out.println(e);
}
}
}
```

Output:-

Table Name: person

Field	Size	DataTypes
id	2	VARCHAR
cname	50	VARCHAR
dob	10	DATE

9. Transaction:

- Transaction represents a single unit of work. The ACID properties describe the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.
- Advantage of Transaction Management is fast performance It makes the performance fast because the database is hit at the time of commit.

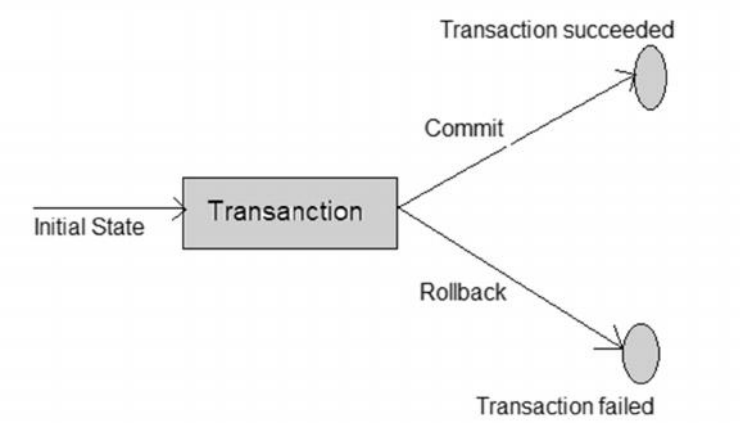


Fig. 6 Transaction Management in JDBC

- In JDBC, the Connection interface provides methods to manage transactions.

Method	Description
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

- Simple example of transaction management in jdbc using Statement

```
import java.sql.*;
class FetchRecords
{
public static void main(String args[])throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@
localhost:1521:xe","system","oracle");
con.setAutoCommit(false);
Statement stmt=con.createStatement();
stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");
con.commit();
con.close();
}
}
```

10 SUMMARY

- Working with databases using java is very simple as java supports various database systems.
- Java has an API called JDBC API which works with databases.

- The JDBC API is industrially accepted for database-independent connectivity between the java programming language and a wide variety of databases and other tabular data sources.

REFERENCE FOR FURTHER READING

1. Java EE 6 for Beginners, Sharanam Shah, Vaishali Shah, SPD
2. https://docs.oracle.com/cd/B10501_01/java.920/a96654/overvw.htm

UNIT END EXERCISES

1. What are the seven basic steps for using JDBC to access a database? Explain briefly with syntax.?
2. Explain different types of JDBC Driver Managers.?
3. Explain the different types of JDBC Driver?
4. Write a short note on: metadata & resultset.?

*****

4

JAVA SERVER PAGES-1

Unit Structure

4.0 OBJECTIVE

1. To understand the difference between JSP & Servlet
2. To study the life cycle of JSP
3. Using JSP separates the design and implementation of web applications.
4. To study the different JSP tag, element, object and their scope.
5. To study the different Action elements available in JSP.

4.1 INTRODUCTION

- Java Server Pages (JSP) is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request.
- The technology allows Java code and certain predefined actions to be embedded into static content. Java Server Pages or JSP for short is Sun's solution for
- Developing dynamic web sites.
- JSP pages typically comprise of:
- Static HTML/XML components.
- Special JSP tags
- Optionally, snippets of code written in the Java programming language called "scriptlets."

4.2 DISADVANTAGES OF JSP

1. Not able to use the feature-rich Swing or AWT controls to construct the user interface.
2. The HTML language has fewer features than the Swing controls for creating a user interface.
3. In addition, simple functions such as scrolling down in a list of records, deleting a record, or changing the way information is sorted requires a refresh of the page.
4. Embedding of JavaScript in the HTML page to enhance functionality, but this solution requires that the JavaScript that you use be supported by the ability of users' browsers to interpret it correctly.
5. As JSP pages are translated to servlets and compiled, it is difficult to trace errors occurred in JSP pages.
6. JSP pages require double the disk space to hold the JSP page.
7. JSP pages require more time when accessed for the first time as they are to be compiled on the server.

4.3 JSP V/S SERVLETS

JSP	Servlets
JSP is a web page scripting language that can generate dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
In MVC(Model View Controller), jsp acts as a view.	In MVC(Model View Controller), servlets act as a controller.
It’s easier to code in JSP than in Java Servlets.	There's little code to write here.
JSP are generally preferred when there is not much processing of data required.	servlets are best for use when there is more processing and manipulation involved.
JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets.	Servlets run faster compared to JSP.
The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans.	There is no such custom tag facility in servlets.
We can achieve functionality of JSP at client side by running JavaScript at client side.	There are no such methods for servlets.

4.4 LIFE CYCLE OF JSP

- JSPs are HTML web pages with special JSP tags embedded.
- These JSP tags can contain Java code. The JSP file extension is .jsp instead of .html.
- The JSP engine parses the .jsp and creates a servlet source file. It then compiles the source file into a class file, this is done the first time and this is why the JSP is probably slower when first time it is accessed.
- Any time after this the special process or compiled servlet is executed and is therefore returned faster.

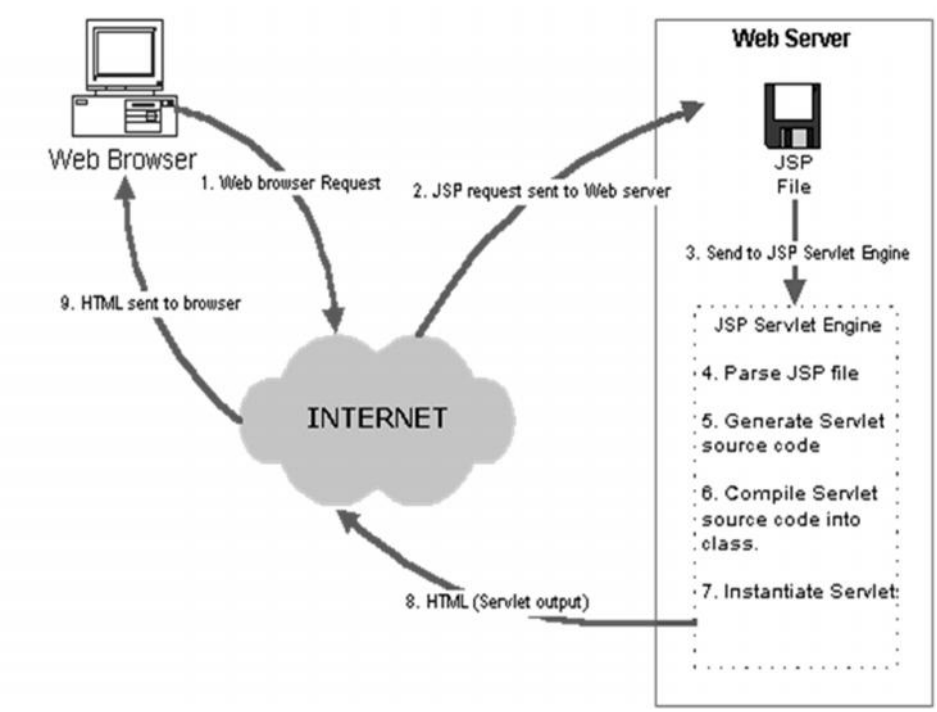


Fig. 1 Life Cycle of JSP

Steps required for a JSP request:

1. The user goes to a web site made using JSP. The user goes to a JSP page (ending with .jsp). The web browser requests the web page via the Internet.
2. The JSP request sent to the Web server for processing.
3. The Web server grant that the file required is special (.jsp),therefore passes the JSP file to the JSP Servlet Engine.
4. If the JSP file has been called the first time,the JSP file is parsed, else go to step 7.
5. The next step is to generate a special Servlet file from the JSP file. All the HTML required is converted to println or output statements.
6. The Servlet source code is compiled into a class file .
7. The Servlet is instantiated, calling the init and service methods.

8. HTML from the Servlet output is sent via the Internet.
9. HTML results are displayed on the user's web browser

Translation phase and a request processing phase:

1. Translation phase:

- Web server needs a servlet container to provide an interface to servlets, the server needs a JSP container to process JSP pages. The JSP container is responsible for intercepting incoming requests for JSP pages.
- To process all JSP elements in the page, the container first turns the JSP page into a servlet (known as the JSP page implementation class). The container then compiles the servlet class.
- JSP page Converted to a servlet and compiling the servlet form the translation phase. The JSP container begins the translation phase for a page automatically when it receives the first request for the page.
- The translation phase takes a bit of time, the first user to request a JSP page notices a slight delay.

2. Request processing phase:

- The JSP container is responsible for invoking the JSP page implementation class to process each request and generate the response. This is called the request processing phase.
- The JSP page remains unchanged; any subsequent request goes straight to the request processing phase. When the JSP page or its content is modified, it goes through the translation phase again before entering the request processing phase.
- The two phases are illustrated in following Figure

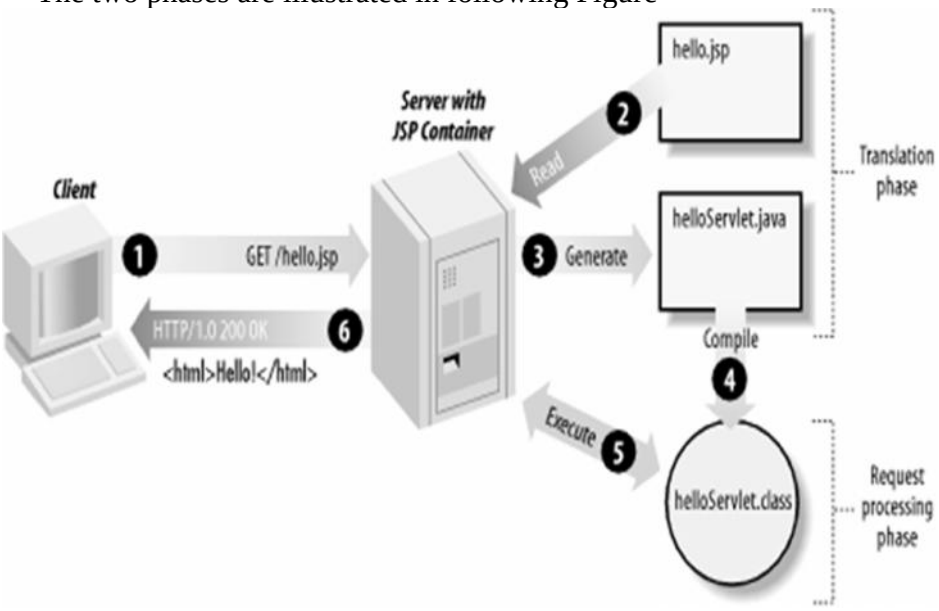


Fig. 2 Translation & Request Processing Phase

JSP Access Model:

- The JSP specifications have two approaches, popularly known as Model 1 and Model 2 architectures, for applying JSP technology.
- These approaches vary essentially in the location at which the bulk of the request processing was performed, and offer a useful paradigm for building applications using JSP technology.
- In the Model 1 architecture, the arriving request from a web browser is sent directly to the JSP page, which is responsible for processing it and replying back to the client. There is still separation of presentation from content, because all data access is performed using beans.
- Although the Model 1 architecture is suitable for simple applications, it may not be desirable for complex implementations. Utilization of this architecture usually leads to a significant amount of scriptlets or Java code embedded within the JSP page, especially if there is a significant amount of request processing to be performed.
- Another drawback of this architecture is that each of the JSP pages must be individually responsible for managing application state and verifying authentication and security.

Model 1 architecture:

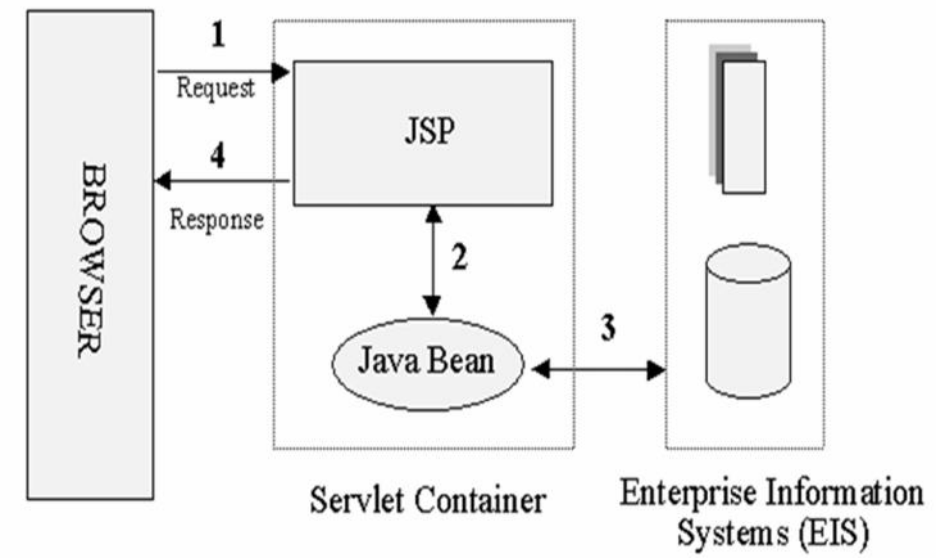


Fig. 3 Model 1Architecture

Model 2 architecture:

- The Model 2 architecture, shown below, is a server-side implementation of the popular Model/View/Controller design pattern. In Model 2 processing is divided between presentation and front components.

- Model 2 Presentation components are JSP pages that generate the HTML/XML response that determines the user interface when rendered by the browser.
- Front components (controllers) do not handle any presentation issues, but rather, process all the HTTP requests. They are responsible for creating any beans or objects used by the presentation components, as well as deciding, depending on the user's actions, which presentation component to forward the request to. Servlet or JSP pages can be implemented by front components.
- The benefits of this architecture is that there is no processing logic within the presentation component itself; it is simply responsible for retrieving any objects or beans that may have been previously created by the controller, and extracting the dynamic content within for insertion within its static templates.
- Another benefit of this approach is that the front components present a single point of entry into the application, making the management of application state, security, and presentation uniform and simple to maintain.

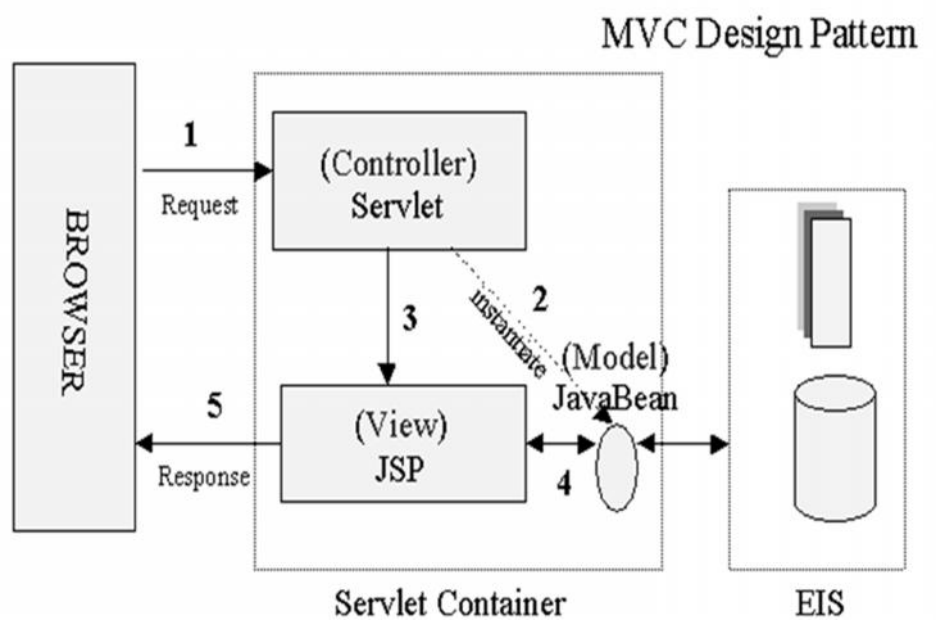


Fig. 4 Model 2 Architecture

4.5 COMMENTS

- JSP comments are ignored by JSP containers that should be marked. A JSP comment is useful for hiding or "comment out" part of your JSP page.
- Following is the syntax of JSP comments:
- <%-- This is JSP comment --%>
- Following is the simple example for JSP Comments:

```
<html>
<head><title>A Comment Test</title></head>
<body>
<h2>A Test of Comments</h2>
<%-- This comment will not be visible in the page source --%>
</body>
</html>
```

- There are a small number of special constructs:

Syntax	Purpose
<%-- comment --%>	A JSP comment. Ignored by the JSP engine.
<!-- comment -->	An HTML comment. Ignored by the browser.
<\%	Represents static <% literal.
%\>	Represents static %> literal.
\'	A single quote in an attribute that uses single quotes.
\"	A double quote in an attribute that uses double quotes.

4.6 JSP DOCUMENTS

- A JSP written in an XML format with JSP elements expressed as XML elements.
- A JSP document is JSP written in XML format and therefore must comply with the XML standard rules.
- JSP document must be well formed
- A JSP Document should be saved with the .jspx extension
- A JSP Document must have a root element called "root" with a "version" attribute like: <jsp:root version="2.1" xmlns:jsp="http://java.sun.com/JSP/Page">
- Identification of JSP documents can occur in three ways:

4.7 JSP ELEMENTS

- A JSP (.jsp) file can contain JSP elements, fixed template data, or any combination of the two.
- JSP elements are instructions to the JSP container about what code to generate and how it should operate.
- JSP elements have distinct start and end tags that identify them to the JSP compiler. Template data is everything else that is not recognized by the JSP container.

- Template or HTML data is passed through unmodified one, so the HTML that is ultimately generated contains the template data exactly as it was coded in the .jsp file.

Three types of JSP elements exist:

1. Directives
2. Scripting elements, including expressions, scriptlets, and declarations
3. Actions

1. Directives:

Directives are instructions to the JSP container that describe what code should be generated. They have the general form

<%@ directive-name [attribute="value" attribute="value" ...] %>

- Zero or more spaces, tabs, and newline characters can be after the opening <%@ and before the ending %>, and one or more whitespace characters can be after the directive name and between attributes/value pairs.
- The only restriction is that the opening <%@ tag must be in the same physical file as the ending %> tag.
- The JSP 1.1 specification describes three standard directives available in all compliant JSP environments:
    1. page
    2. include
    3. taglib
- No custom directives can be used in the JSP environment, this leaves open the possibility that user-defined directives may be included in a later specification.

The next three sections provide an overview of each of these directives.

1. The page Directive

- The page directive is used to specify attributes for the JSP page as a whole. It has the following syntax:

<%@ page [attribute="value" attribute="value" ...] %>

Attributes of JSP page directive

- import
- contentType
- extends
- info

- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

1) Import:

The import attribute is used to import class, interface or all the members of a package. It is similar to importing keywords in java class or interface.

Example of import attribute

```
<html>
<body>
<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>
</body>
</html>
```

2) contentType:

- The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

Example of contentType attribute

```
<html>
<body>
<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

3) extends:

The extends attribute defines the parent class that will be inherited by the generated servlet. It is rarely used.

4) Info:

- This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of the Servlet interface.

Example of info attribute

```
<html>
<body>
<%@ page info="composed by ABC" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

The web container will create a method getServletInfo() in the resulting servlet.

For example:

```
public String getServletInfo()
{
return "composed by ABC";
}
```

5) Buffer:

- The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page. The default size of the buffer is 8Kb.

Example of buffer attribute

```
<html>
<body>
<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

6) Language:

- The language attribute specifies the scripting language used in the JSP page. The default value is "java".

7) isELIgnored:

- We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute.

- By default its value is false i.e. Expression Language is enabled by default.

  <%@ page isELIgnored="true" %>//Now EL will be ignored

8) isThreadSafe:

- Servlet and JSP both are multithreaded. If you want to control this behaviour of a JSP page, you can use the isThreadSafe attribute of the page directive.
- The value of isThreadSafe value is true. If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.
- The value of isThreadSafe attribute like:

  <%@ page isThreadSafe="false" %>

The web container in such a case, will generate the servlet as:

```
public class SimplePage_jsp extends HttpJspBase
implements SingleThreadModel{
.....
}
```

9) errorPage:

- The errorPage attribute is used to define the error page, if an exception occurs in the current page, it will be redirected to the error page.

Example:

```
//index.jsp
<html>
<body>
<%@ page errorPage="myerrorpage.jsp" %>
<%= 100/0 %>
</body>
</html>
```

10) isErrorPage:

The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example:

```
//myerrorpage.jsp
```

```
<html><body>
<%@ page isErrorPage="true" %>
 Sorry an exception occured!
<br/>
The exception is: <%= exception %>
</body>
</html>
```

2. JSP Include directive:

- The include directive is used to include the contents of any resource it may be jsp file, html file or text file.
- The include directive includes the original content of the included resource at page translation time.
- Advantage of Include directive is code reusability
- Syntax of include directive:

<%@ include file="resourceName" %>

Example of include directive

It is including the content of the header.html file. To run this example you must create a header.html file.

```
<html>
<body>
<%@ include file="header.html" %>
Today is: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

3. JSP Taglib directive:

- The JSP taglib directive is used to define a tag library that defines many tags.
- The TLD (Tag Library Descriptor) file to define the tags.
- In the custom tag section use this tag so it will be better to learn it in custom tag.

Syntax JSP Taglib directive

<%@ tagliburi="uriofthetaglibrary" prefix="prefixoftaglibrary" %>

Example of JSP Taglib directive:

In this example, using our tag named currentDate. To use this tag it must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>
<%@ tagliburi="http://www.javatpoint.com/tags" prefix="mytag" %>
<mytag:currentDate/>
</body>
</html>
```

There are four types of scripting elements:

1. Scriptlet tag
2. Expression tag
3. Declaration tag
4. Comment tag

1. scriptlet tag:

- A scriptlet is a valid code in java and is placed in the _jspService() method of the JSP engine at the time of running. The scriptlet syntax is-

  <% java code %>

- There are variables available exclusively for the scriplets.
- They are request, response, out,session and pageContext, application, config and exception.

Simple Example of JSP scriptlet tag

In this example, displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name

- In this example, create two files index.html and welcome.jsp.
- The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

```
index.html
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

```
welcome.jsp
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

2. JSP expression tag:

- The code placed within the expression tag is written to the output stream of the response.
- Need not write out.print() to write data. It is mainly used to print the values of variables or methods.

Syntax of JSP expression tag

```
<%=  statement %>
```

Example of JSP expression tag

In this example of jsp expression tag, simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Example of JSP expression tag that prints current time

- To display the current time, it is used in the getTime() method of Calendar class.
- The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

Example of JSP expression tag that prints the user name

```
index.jsp
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

- In this example, printing the username using the expression tag.
- The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

```
index.html
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

```
welcome.jsp
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</form>
</body>
</html>
```

3. JSP Declaration Tag

- The JSP declaration tag is used to declare fields and methods.
- The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

Syntax of JSP declaration tag

<%! field or method declaration %>

Difference between the jspscriptlet tag and jsp declaration tag

JspScriptlet Tag	**Jsp Declaration Tag**
The jspscriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example of JSP declaration tag that declares field

- In this example of JSP declaration tag, declaring the field and printing the value of the declared field using the jsp expression tag.

```
index.jsp
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example of JSP declaration tag that declares method

- In this example of JSP declaration tag, hence defining the method which returns the cube of given number and calling this method from the jsp expression tag. But it can also use jspscriptlet tag to call the declared method.

```
index.jsp
<html>
<body>
<%!
int cube(int n)
        {
             return n*n*n*;
         }
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

4. JSP Comment:

- There is only one type of JSP comment available by JSP specification.
  JSP Comment Syntax:

  <%-- comment --%>

- This JSP comment tag tells the JSP container to ignore the comment part from compilation. That is, the commented part of source code is not considered for the content parsed for 'response'.

```
<html>
<body>
<%-- This JSP comment part will not be included in the response object --%>
</body>
</html>
```

4.8 SUMMARY

1. Java Server Pages (JSP) is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request.
2. JSP pages typically consist of: Static HTML/XML components. Special JSP tags optionally, snippets of code written in the Java programming language called "scriptlets."
3. The JSP page goes through the Translation Phase and Request Processing Phase.

REFERENCE FOR FURTHER READING

1. JSP: The Complete Reference Phil Hanna
2. Java EE 6 for Beginners, Sharanam Shah, Vaishali Shah, SPD

UNIT END EXERCISES

1. What is JSP? Explain the Life Cycle of JSP Page?
2. Explain the difference between JSP & Servlet?
3. Write a JSP Program to print current Date & Time.

*****

5

JAVA SERVER PAGES-2

Unit Structure

5.0 OBJECTIVE

1. To understand the different Action tag available in JSP
2. To study the Implicit Objects
3. Understand Difference between jsp include directive and include action
4. To study the use of JavaBeans in JSP.
5. To study the scope of JSP Objects.

5.1 INTRODUCTION

- There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.
- The action tags are used to control the flow between pages and to use Java Bean.
- There are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages.
- The available implicit objects are out, request, config, session, application etc.

5.2 ACTION ELEMENTS

- There are many JSP action tags or elements.

- Each JSP action tag is used to perform some specific tasks.
- The action tags are used to control the flow between pages and to use Java Bean.

The Jsp action tags are given below.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in a bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds other components such as applets.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if the plugin is working. It is used in jsp:plugin.

- The jsp:useBean, jsp:setProperty and jsp:getProperty tags are used for bean development.

A. jsp:forward action tag
   - The jsp:forward action tag is used to forward the request to another resource; it may be jsp, html or another resource.
   - Syntax of jsp:forward action tag without parameter

     <jsp:forward page="relativeURL | <%= expression %>" />

   - Syntax of jsp:forward action tag with parameter

     <jsp:forward page="relativeURL | <%= expression %>">
     <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
     </jsp:forward>

   - Example of jsp:forward action tag without parameter

In this example, we are simply forwarding the request to the printdate.jsp file.

index.jsp
```
<html>
<body>
<h2>this is index page</h2>
<jsp:forward page="printdate.jsp" />
</body>
</html>
```
printdate.jsp
```
<html>
```

```
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime());
%>
</body>
</html>
```

● Example of jsp:forward action tag with parameter

In this example, we are forwarding the request to the printdate.jsp file with parameter and printdate.jsp file prints the parameter value with date and time.

```
index.jsp
<html>
<body>
<h2>this is index page</h2>
<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="ABC" />
</jsp:forward>
</body>
</html>

printdate.jsp
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime());
%>
<%= request.getParameter("name") %>
</body>
</html>
```

B. jsp:include action tag:

- The jsp:include action tag is used to include the content of another resource. It may be jsp, html or servlet.
- The jsp include action tag includes the resource at request time so it is better for dynamic pages.
- The jsp:include tag can be used to include static as well as dynamic pages.
- Advantage of jsp:include action tag is code reusability
- Difference between jsp include directive and include action

JSP include directive	JSP include action
includes resources at translation time.	includes resources at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

- Syntax of jsp:include action tag without parameter
  <jsp:include page="relativeURL | <%= expression %>" />

- Syntax of jsp:include action tag with parameter
  <jsp:include page="relativeURL | <%= expression %>">
  <jsp:param name="parametername"
  value="parametervalue | <%=expression%>" />
  </jsp:include>

- Example of jsp:include action tag without parameter

In this example, index.jsp file includes the content of the printdate.jsp file.

```
index.jsp
<html>
<body>
<h2>this is index page</h2><jsp:include page="printdate.jsp" />
<h2>end section of index page</h2>
</body>
</html>

printdate.jsp
<html>
<body>
<%                                                    out.print("Today
is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

C. jsp:useBean action tag:

- A Java Bean is a java class that should follow following conventions:
  1. It should have a no-arg constructor.
  2. It should be Serializable.
  3. It should provide methods to set and get the values of the properties, known as getter and setter methods.
- Use Java Bean: According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so it can access this object from multiple places. Moreover, it provides easy maintenance.

Example of java bean class:

```
//Employee.java
package mypack;
public class Employee implements java.io.Serializable
{
        private int id;
        private String name;
```

```
        public Employee()
        {
         }
        public void setId(int id){this.id=id;
}
public intgetId()
{
      return id;
}
public void setName(String name){this.name=name;
} public String getName(){return name;
}
}
```

- **Access the java bean class:**

To access the java bean class, we should use getter and setter methods.

```
package mypack;
public class Test{
public static void main(String args[])
{
Employee e=new Employee();//object is created
e.setName("Subodh");//setting value to the object
System.out.println(e.getName()); }}
```

D. jsp:useBean action tag

- The jsp:useBean action tag is used to locate or instantiate a bean class.
- If the bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if an object of the bean is not created, it instantiates the bean.

Syntax of jsp:useBean action tag

```
<jsp:useBean id= "instanceName" scope= "page | request |
session | application"
        class= "packageName.className" type=
"packageName.className"
        beanName="packageName.className | <%= expression
>" >
</jsp:useBean>
```

Attributes and Usage of jsp:useBean action tag

1. id: is used to identify the bean in the identity scope.
2. scope:represents the scope of the bean. It may be page, request, session or application. The default scope is page.

3. page: specifies that you can use this bean within the JSP page. The default scope is page.
4. request: specifies that you can use this bean from any JSP page that processes the same request. It has a wider scope than the page.
5. session: specifies that you can use this bean from any JSP page in the same session whether it processes the same request or not. It has a wider scope than request.
6. application: specifies that you can use this bean from any JSP page in the same application. It has a wider scope than session.
7. class: instantiates the specified bean class but it must have no-arg or no constructor and must not be abstract.
8. type:provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attributes. If you use it without class or beanName, no bean is instantiated.
9. beanName:instantiates the bean using the java.beans.Beans.instantiate() method.

Simple example of jsp:useBean action tag

```
Calcub.java
public class Calcube{
public int cube(int n){return n*n*n;}
}
index.jsp file
<jsp:useBean id="obj" class="ABC"/>
<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

E. jsp:setProperty and jsp:getProperty action tags:

- The setProperty and getProperty action tags are used for developing web applications using Java Bean.
- In web applications, bean class is mostly used because it is a reusable software component that represents data.
- The jsp:setProperty action tag sets a property value or values in a bean using the setter method.
- Syntax of jsp:setProperty action tag

```
<jsp:setProperty name="instanceOfBean" property= "*"  |
        property="propertyName" param="parameterName"  |
        property="propertyName" value="{ string | <%= expression %>}"
/>
```

- Example of jsp:setProperty:

```
<jsp:setProperty name="bean" property="*" />
```

- Example of jsp:setProperty:
  <jsp:setProperty name="bean" property="username" />
- Example of jsp:setProperty:
  <jsp:setProperty name="bean" property="username" value="ABC" />
- jsp:getProperty action tag
  The jsp:getProperty action tag returns the value of the property.
- Syntax of jsp:getProperty action tag
  <jsp:getProperty name="instanceOfBean" property="propertyName" />
- Simple example of jsp:getProperty action tag
  <jsp:getProperty name="obj" property="name" />

F. Displaying applet in JSP (jsp:plugin action tag)

- The jsp:plugin action tag is used for embedded applets in the jsp file. The jsp:plugin action tag downloads plugin at client side to execute an applet or bean programs.
- Syntax of jsp:plugin action tag
  <jsp:plugin type= "applet | bean" code= "nameOfClassFile"
  codebase= "directoryNameOfClassFile"
  </jsp:plugin>

4. implicit objects:

- There are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages.
- The available implicit objects are out, request, config, session, application etc.
- A list of the 9 implicit objects is given below:

Object	Type
Out	JspWriter
Request	HttpServletRequest
Response	HttpServletResponse
Config	ServletConfig
application	ServletContext
Session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

A. JSP out:

- For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

PrintWriter out=response.getWriter();

Example of out implicit object

```
index.jsp
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime());
%>
</body>
</html>
```

B. JSP request:

- The JSP request is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can also be used to set, get and remove attributes from the jsp request scope.

```
Example of JSP request implicit objectindex.html
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go">
<br/>
</form>
welcome.jsp
<% String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

C. JSP response:

- JSP response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.
- Example of response implicit object

```
index.html
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
welcome.jsp
<%
response.sendRedirect("http://www.google.com");
%>
```

D. JSP config:

- In JSP, config is an implicit object of type ServletConfig.
- This object can be used to get an initialization parameter for a particular JSP page.
- The config object is created by the web container for each jsp page.

Example of config implicit object:

```
index.html
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
web.xml file
<web-app>
<servlet>
<servlet-name>ABC</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
<init-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>ABC</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
welcome.jsp
<%
out.print("Welcome "+request.getParameter("uname"));
String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

E. JSP application:

- In JSP, application is an implicit object of type ServletContext.
- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

Example:

```
index.html
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
```

```
</form>
web.xml file
<web-app>
<servlet>
<servlet-name>ABC</servlet-name>
<jsp-file>/welcome.jsp</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>ABC</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>
</web-app>
welcome.jsp
<%
out.print("Welcome "+request.getParameter("uname"));
String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

F. session:

- In JSP, session is an implicit object of type HttpSession.
- The Java developer can use this object to set,get or remove attributes or to get session information.

Example:

```
index.html
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
welcome.jsp
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
session.setAttribute("user",name);
```

```
<a href="second.jsp">second jsp page</a>
%>
</body>
</html>
second.jsp
<html>
<body>
<%
String name=(String)session.getAttribute("user");
out.print("Hello "+name);
%>
</body>
</html>
```

G. pageContext:

- In JSP, pageContext is an implicit object of type PageContext class.
- The pageContext object can be used to set,get or remove attribute from one of the following scopes:
    1. Page
    2. request
    3. Session
    4. Application

In JSP, page scope is the default scope.

Example:

```
index.html
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
welcome.jsp
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
<a href="second.jsp">second jsp page</a>
%>
</body>
```

```
</html>
second.jsp
<html>
<body>
<%
                String name=(String)pageContext.getAttribute
                    ("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);
%>
</body>
</html>
```

5. Scope:

- The availability of a JSP object for use from a particular place of the application is defined as the scope of that JSP object.
- Every object created in a JSP page will have a scope. Object scope in JSP is segregated into four parts and they are page, request, session and application.

a) Page:

- 'page' scope means the JSP object can be accessed only from within the same page where it was created.
- The default scope for JSP objects created using <jsp:useBean> tag is page.
- JSP implicit objects out, exception, response, pageContext, config and page have 'page' scope.

b) Request:

- A JSP object created using the 'request' scope can be accessed from any pages that serves that request.
- More than one page can serve a single request.
- The JSP object will be bound to the request object.
- Implicit object request has the 'request' scope.

c) Session:

- 'Session scope means the JSP object is accessible from pages that belong to the same session from where it was created.
- The JSP object that is created using the session scope is bound to the session object.
- Implicit object session has the 'session' scope.

d) Application:

- A JSP object created using the 'application' scope can be accessed from any pages across the application.
- The JSP object is bound to the application object.
- Implicit object application has the 'application' scope.

6. Character quoting conventions:

- There are a small number of special constructs we can use in various cases to insert comments or characters that would otherwise be treated specially.
- Character Quoting And Data Conventions :

Syntax	Purpose
<\%	Used in template text (static HTML) where you really want "<%".
%\>	Used in scripting elements where you really want "%>".
\'	A single quote in an attribute that uses single quotes. Remember, however, that you can use either single or double quotes, and the other type of quote will then be a regular character.
\"	A double quote in an attribute that uses double quotes. Remember, however, that you can use either single or double quotes, and the other type of quote will then be a regular character.
%\>	%>in an attribute.
<\%	<% in an attribute.
\	Used as a delimiter.
<%-- comment--%>	A JSP comment. Ignored by JSP-to- scriptlet translator. Any embedded JSP scripting elements, directives, or actions are ignored.
<!-- comment-->	An HTML comment. Passed through to resultant HTML. Any embedded JSP scripting elements, directives, or actions are executed normally.

7. Unified expression language:

Expression Language(EL):

- The Expression Language (EL) provides a way to simplify expressions in JSP.
- It is a simple language used for accessing implicit objects and Java classes, and for manipulating collections in an elegant manner.
- EL provides the ability to use run-time expressions outside of JSP scripting elements.

"Unified" Expression Language:

- JavaServer Pages each has its own expression language.
- The expression language included in JSP provides greater flexibility to the web application developer.
- Deferred evaluation means that the technology using the unified EL takes over the responsibility of evaluating the expression from the JSP engine and evaluates the expression at the appropriate time during the page lifecycle.
- But the JSP EL is designed for immediate evaluation of expressions.
- The Expression Language (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc.
- There are many implicit objects, operators and reserve words in EL.
- Syntax for Expression Language (EL)

${ expression }

- Implicit Objects in Expression Language (EL)

There are many implicit objects in the Expression Language. They are as follows:

Implicit Objects	Usage
pageScope	it maps the given attribute name with the value set in the page scope
requestScope	it maps the given attribute name with the value set in the request scope
sessionScope	it maps the given attribute name with the value set in the session scope
applicationScope	it maps the given attribute name with the value set in the application scope
param	it maps the request parameter to the single value
paramValues	it maps the request parameter to an array of values
header	it maps the request header name to the single value
headerValues	it maps the request header name to an array of values
cookie	it maps the given cookie name to the cookie value
initParam	it maps the initialization parameter
pageContext	it provides access to many objects request, session etc.

Example:

index.jsp
<form action="process.jsp">
Enter Name:<input type="text" name="name" />

<input type="submit" value="go"/>
</form>
process.jsp
Welcome, ${ param.name }
Example of Expression Language that prints the value set in the session scope
In this example, we print the data stored in the session scope using EL. For this purpose, we have used a sessionScope object.
Index.jsp

<h3>welcome to index page</h3>
<%
session.setAttribute("user","ABC");
%>
visit
process.jsp
Value is ${ sessionScope.user }

- Precedence of Operators in EL

There are many operators that have been provided in the Expression Language.

1. [] .
2. ()
3. -(unary) not ! empty
4. * / div % mod
5. + - (binary)
6. <<= >>= lt le gtge
7. == != eq ne
8. && and
9. || or
10. ?:

- Reverse words in EL

There are many reserved words in the Expression Language. They are as follows:

lt	le	gt	ge
eq	ne	true	false
and	or	not	instanceof
div	mod	empty	null

SUMMARY

1. JSP specification provides Standard(Action) tags for use within your JSP pages.
2. These tags are used to remove or eliminate scriptlet code from your JSP page because scriplet code is technically not recommended nowadays.
3. It's considered to be bad practice to put java code directly inside your JSP page.
4. There are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages.
5. The available implicit objects are out, request, config, session, application etc.
6. JSP provides the capability to the user to define the scope of these variables.

REFERENCE FOR FURTHER READING

1. Java EE 6 for Beginners, Sharanam Shah, Vaishali Shah, SPD
2. JSP: The Complete Reference Phil Hanna

UNIT END EXERCISES

1. Write a short note on Expression Language? With example.
2. Explain the different action tag in jsp with example?
3. What is an implicit object?
4. What is scope in JSP object?

*****

UNIT IV

6

JAVA SERVER FACES

Unit Structure

6.1 OBJECTIVE

- To reduce the effort in creating and maintaining applications using JSF, this will run on a Java application server and will render application UI on to a target client.
- To Providing reusable UI components
- Making easy data transfer between UI components
- To Managing UI state across multiple server requests
- Enabling implementation of custom components
- To Understand the client-side event to server-side application code

6.2 INTRODUCTION

- Development of the Web is constantly changing from plain HTML to Servlet, JSP to JavaServer Faces.
- JSF is the latest and most advanced web component technology.
- Initially a web application written using Servlets, which is used to deliver HTML directly.
- A JSP is a composite between an HTML page and a servlet.
- JSF is new and most advanced technology for instant building web applications using java.

- JSF follows the Model View Controller architecture.

6.3 NEED OF MVC

- In the past, the first problem during Software development always desired to simplify modification of the user interface.
- The second one is that programmer who develop application have different skills sets which is classified as follows:
  - Server side programmers
  - HTML code writers
  - Graphics Designers
- MVC does separation of logic from presentation, which allows each member of the WAD team to concentrate on its own development proccess.

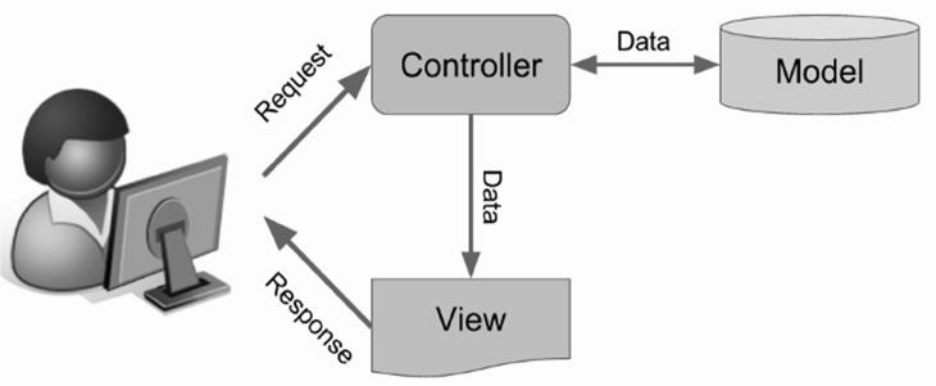


Fig. 1 MVC Design

- MVC design pattern designs an application using three separate modules:

Module	Description
Model	Carries Data and login
View	Shows User Interface
Controller	Handles processing of an application.

6.4 WHAT IS JSF?

- JSF is a Server side user interface component framework to build web applications
- JSF simplifies web application development by providing a compose centric approach to developing java web interfaces.
- Server side:
  - Present Enterprise application are multi-tier and mostly server side
- User Interface
  - JSF is used to create a user interface.
- Component Framework

- Components are the core of the framework.
- JSF is the standard web component technology for java

- The framework is not only limited to developing customized User Interface Components but also provides support for various Advanced Features like Event handling Mechanism, Validating User Inputs that are sent by the clients, Easy Page Navigation Mechanism etc.
- JSF is a Java standard developed through Java Community Process (JCP), development tools vendors are fully empowered to provide easy to use, visual, and productive develop environments for JavaServer Faces.
- JSF provides a rich component model complete with event handling and component rendering.

6.5 COMPONENTS OF JSF

- In JSF, a component is a group of interacting classes that together provide a reusable piece of web-based user interface code. A component is made up of three classes that work closely together.
- Components in JSF are elements like text box, button, table etc..that are used to create UI of JSF Applications. These are objects that manage interaction with a user.
  - Simple components like text box, button.
  - Compound components like table, data grid
- A component containing many components inside it is called a compound component.
- Components help developers to create UIs by assembling a number of components, associating them with

There are two types of Components in JSF

A. Standard UI Components

B. Custom UI Components

A. Standard UI Components:

- It contains a basic set of UI components like text fields, check boxes , list boxes, panel, label, radio button etc. These are called standard components

B. Custom UI Components:

- JSF lets create and use your own set of reusable components, These components are called custom components.

JSF–Basic Tags

- JSF provides a standard HTML tag library. These tags get rendered into corresponding html output.

- These tags need to use the following namespaces of URI in the html node.

<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html" >

Following are important Basic Tags in JSF:

S.N.	Tag & Description
1	h:inputText Renders a HTML input of type="text", text box.
2	h:inputSecret Renders a HTML input of type="password", text box.
3	h:inputTextarea Renders a HTML textarea field.
4	h:inputHidden Renders a HTML input of type="hidden".
5	h:selectBooleanCheckbox Renders a single HTML check box.
7	h:selectOneRadio Renders a single HTML radio button.
8	h:selectOneListbox Renders a HTML single list box.
9	h:selectManyListbox Renders a HTML multiple list box.
10	h:selectOneMenu Renders a HTML combo box.
11	h:outputText Renders a HTML text.
12	h:outputFormat Renders a HTML text. It accepts parameters.
13	h:graphicImage Renders an image.
14	h:outputStylesheet Includes a CSS style sheet in HTML output.
15	h:outputScript Includes a script in HTML output.
16	h:commandButton Renders a HTML input of type="submit" button.
17	h:Link Renders a HTML anchor.
18	h:commandLink Renders a HTML anchor.
19	h:outputLink Renders a HTML anchor.
20	h:panelGrid Renders an HTML Table in form of grid.
21	h:message Renders message for a JSF UI Component.
22	h:messages Renders all messages for JSF UI Components.
23	f:param Pass parameters to JSF UI Component.
24	f:attribute Pass attribute to a JSF UI Component.
25	f:setPropertyActionListener Sets value of a managed bean's property

JSF – Facelets Tags:

- JSF provides special tags to create a common layout for a web application called facelets tags.
- These tags give flexibility to manage common parts of multiple pages at one place.
- The following namespaces of URI in the html node.

<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets" >

Following are important Facelets Tags in JSF:

Sr. No.	Tag & Description
1.	Templates We'll demonstrate how to use templates using following tags <ui:insert> <ui:define> <ui:include> <ui:define>
2.	Parameters We'll demonstrate how to pass parameters to a template file using following tag <ui:param>
3.	Custom We'll demonstrate how to create custom tags.
4.	Remove We'll demonstrate capability to remove JSF code from generated HTML pages.

JSF–Convertor Tags:

- JSF provides inbuilt converters to convert its UI component's data to objects used in a managed bean and vice versa.
- For example, these tags can convert a text into date objects and can validate the format of input as well.
- For these tags you need to use the following namespaces of URI in html node.

<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:f="http://java.sun.com/jsf/core" >

Following are important *Convertor Tags* in JSF :

S.No.	Tag & Description
1	f:convertNumber Converts a String into a Number of desired format
2	f:convertDateTime Converts a String into a Date of desired format
3	Custom Convertor Creating a custom convertor

JSF–Validator Tags:

- JSF provides inbuilt validators to validate its UI components. These tags can validate length of field, type of input which can be a custom object.
- For these tags you need to use the following namespaces of URI in the html node.

<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:f="http://java.sun.com/jsf/core" >

Following are important *Validator Tags* in JSF :

Sr. No.	Tag & Description
1.	f:validateLength Validates length of a string
2.	f:validateLongRange Validates range of numeric value
3.	f:validateDoubleRange Validates range of float value
4.	f:validateRegex Validate JSF component with a given regular expression.
5.	Custom Validator Creating a custom validator

JSF–Composite Components

- JSF provides developers a powerful capability to define their own custom components which can be used to render custom contents.

S.No.	Tag & Description
1.	composite:interface Declare configurable values to be used in composite:implementation
2.	composite:attribute Configuration values are declared using this tag
3.	composite:implementation Declares JSF component. Can access the configurable values defined in composite:interface using #{cc.attrs.attribute-name} expression.

6.6 JSF AS AN APPLICATION

- A JSF application is like a simple java web application, which is run by a servlet web container.
- Example,
  - A client makes an HTTP request for a page via an HTTP server.
  - The server responds back by rendering a user interface using JSF technology after translating to a pure HTML page.
- JSF user interface manages:
  - The User Interface component object that maps to the tag on the page.
  - Event listener, validation and converters that register on the component.
  - Java bean component that encapsulate the data and application specific functionality of the component.

Facelets

- Facelets is a powerful but lightweight page declaration language that is used to build JavaServer Faces views using HTML style templates and to build component trees. Facelets features include the following:
  - Use of XHTML for creating web pages

    - Support for Facelets tag libraries in addition to JavaServer Faces and JSTL tag libraries
    - Support for the Expression Language (EL)
    - Templating for components and pages
- Advantages of Facelets for large-scale development projects include the following:
    - Support for code reuse through templating and composite components
    - Functional extensibility of components and other server-side objects through customization
    - Faster compilation time
    - Compile-time EL validation
    - High-performance rendering

6.7 JSF LIFECYCLE

- The lifecycle of an application refers to the various stages of processing of that application, from its initiation to its conclusion.
- All applications have life cycles.
- During a web application lifecycle, common tasks are performed, including the following.
    - Handling incoming requests
    - Decoding parameters
    - Modifying and saving state
    - Rendering web pages to the browser
- The JavaServer Faces web application framework manages lifecycle phases automatically for simple applications or allows you to manage them manually for more complex applications as required.
- JavaServer Faces applications that use advanced features may require interaction with the lifecycle at certain phases.
- The lifecycle of a JavaServer Faces application begins when the client makes an HTTP request for a page and ends when the server responds with the page, translated to HTML.
- The lifecycle can be divided into two main phases: Execute and Render. The Execute phase is further divided into subphases to support the sophisticated component tree. This structure requires that component data be converted and validated, component events be handled, and component data be propagated to beans in an orderly fashion.

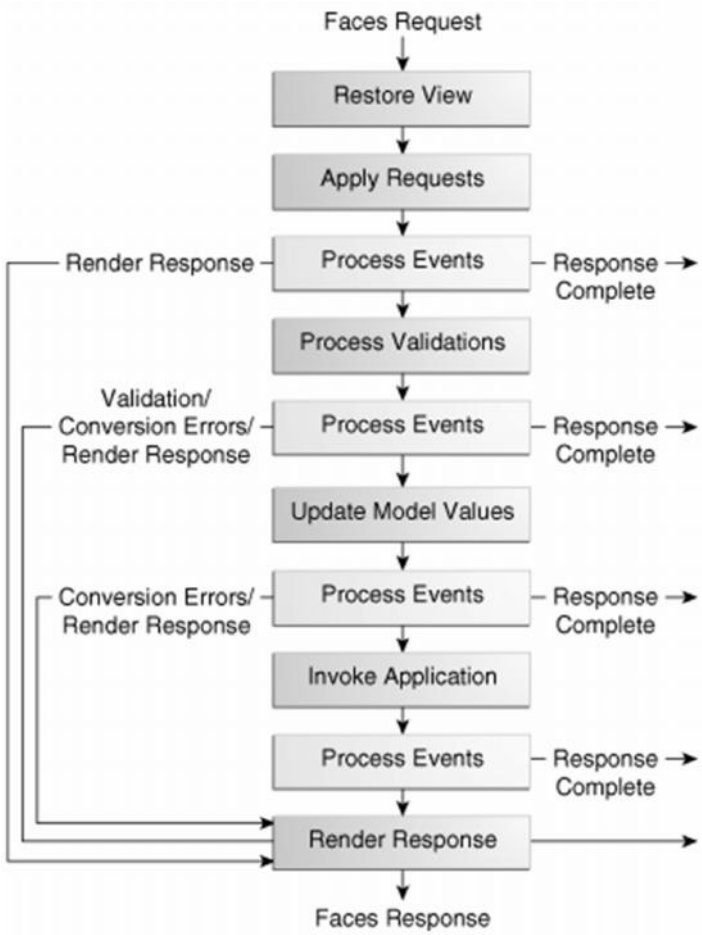


Fig. 1 JavaServer Faces Standard Request-Response Lifecycle

- The JavaServer Faces application lifecycle Execute phase contains the following subphases:
  - Restore View Phase
  - Apply Request Values Phase
  - Process Validations Phase
  - Update Model Values Phase
  - Invoke Application Phase
  - Render Response Phase

Restore View Phase

- When a request for a JavaServer Faces page, by an action, such as when a link or a button component is clicked, the JavaServer Faces implementation starts the Restore View phase.
- Through this phase, the JavaServer Faces implementation constructs the view of the page, wires event handlers and validators to components in the view, and saves the view in the FacesContext instance, which has all the information needed to process a single request. All the components, event handlers, converters, and validators have access to the FacesContext instance.

Apply Request Values Phase:

- The component tree is restored during a postback request, each component in the tree extracts its new value from the request parameters by using its decode method. The value is stored locally on each component.
- If any decode methods or event listeners methods have called the renderResponse method on the current FacesContext instance, the JavaServer Faces implementation skips to the Render Response phase.
- If any events have been queued during this phase, the JavaServer Faces implementation broadcasts the events to interested listeners.

Process Validations Phase

- The JavaServer Faces implementation processes all validators registered on the components in the tree by using its processValidators method.
- It examines the component attributes that specify the rules for the validation and compares these rules to the local value stored for the component.
- The JavaServer Faces implementation also completes conversions for input components that do not have the immediate attribute set to true.
- If the local value is invalid, or if any conversion fails, the JavaServer Faces implementation adds an error message to the FacesContext instance, and the lifecycle advances directly to the Render Response phase so that the page is rendered again with the error messages displayed. If there were conversion errors from the Apply Request Values phase, the messages for these errors are also displayed.
- If any validate methods or event listeners have called the renderResponse method on the current FacesContext, the JavaServer Faces implementation skips to the Render Response phase.

Update Model Values Phase:

- The JavaServer Faces implementation determines that the data is valid, it traverses the component tree and sets the corresponding server-side object properties to the components local values. The JavaServer Faces implementation updates only the bean properties pointed at by an input component's value attribute.
- If any updateModels methods or any listeners have called the renderResponse method on the current FacesContext instance, the JavaServer Faces implementation skips to the Render Response phase.

Invoke Application Phase

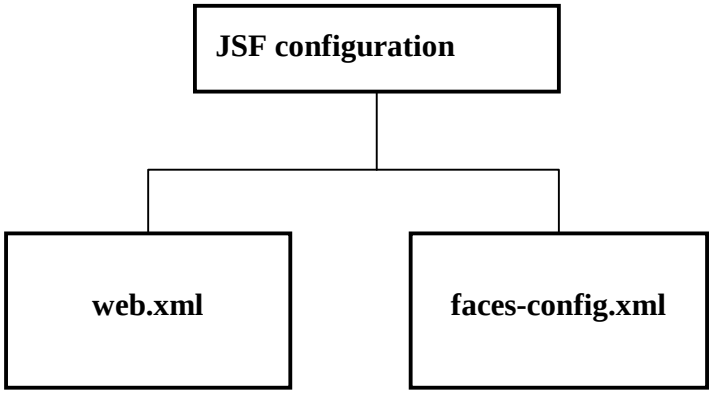
- In this phase the JavaServer Faces implementation handles any application-level events, such as submitting a form or linking to another page.
- At this point, if the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call the FacesContext.responseComplete method.
- If the view being processed was reconstructed from state information from a previous request and if a component has fired an event, these events are broadcast to interested listeners.
- Finally, the JavaServer Faces implementation transfers control to the Render Response phase.

Render Response Phase

- During this phase, JavaServer Faces constructs the view and typical authority to the suitable resource for rendering the pages.
- If the first request is this, the components that are represented on the page will be added to the component tree. If this is not the first request, the components are already added to the tree and need not be added again.
- If the request is a postback and errors were experienced during the Apply Request Values phase, Process Validations phase, or Update Model Values phase, the original page is rendered again during this phase. If the pages consist h:message or h:messages tags, any queued error messages are displayed on the page.
- After the content of the view is relinquished, the state of the response is saved so that upcoming requests can access it.

6.8 JSF CONFIGURATION

JSF is based on the following configuration files:



- web.xml - General web application configuration file
- faces-config.xml - Contains the configuration of the JSF application.

web.xml

- JSF requires the central configuration list web.xml in the directory WEB-INF of the application. This is similar to other web-applications which are based on servlets.
- A FacesServlet is responsible for handling JSF applications.
- FacesServlet is the central controller for the JSF application.
- FacesServlet receives all requests for the JSF application and initializes the JSF components before the JSP is displayed.

Initial format:

```
<?xml version="1.0"?>
<web-app>
 .....
 .....
 .....
</web-app>
```

faces-config.xml:

- Second file faces-config.xml that will be in the same place where web.xml is i.e. WEB-INF folder.
- Here to take care of mentioning the version of xml as we did in the web.xml file.
- All tag elements will be within faces-config opening and closing tag i.e. <faces-config>and </faces-config>.So the root element of this file is <faces-config> tag.
- "faces-config.xml" allows us to configure the application, managed beans, convertors, validators, and navigation.

Initial format:

```
<?xml version="1.0"?>
<faces-config>
 .....
 .....
 .....
</faces-config>
```

6.9 JSF WEB APPLICATIONS (LOGIN FORM, JSF PAGES)

- This application will take a user First Name and Last Name. Later these fields will be validated by JSF and using the controller bean and Navigation rule the output will be displayed.

- This application will also introduce a UI component which is a submit button.

Step 1: Create the table Users in mysql database as

```
CREATETABLEUsers(
uidint(20) NOTNULL AUTO_INCREMENT,
unameVARCHAR(60) NOTNULL,
passwordVARCHAR(60) NOTNULL,
PRIMARY KEY(uid));
```

Step 2: Insert data into the table Users as;

```
INSERTINTOUsersVALUES(1,'adam','adam');
```

Step 3: Create the JSF login page login.xhtml as;

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "https://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="https://www.w3.org/1999/xhtml"
	xmlns:h="https://java.sun.com/jsf/html">
<h:head>
	<title>login</title>
</h:head>
<h:body>
	<h:form>
		<h3>JSF Login Logout</h3>
		<h:outputText value="Username" />
		<h:inputText id="username" value="#{login.user}"></h:inputText>
		<h:message for="username"></h:message>
		<br></br><br></br>

		<h:outputText value="Password" />
		<h:inputSecret id="password" value="#{login.pwd}"></h:inputSecret>
		<h:message for="password"></h:message>
		<br></br><br></br>

		<h:commandButton action="#{login.validateUsernamePassword}"
			value="Login"></h:commandButton>
	</h:form>
</h:body>
</html>
```

Step 4: Create the managed bean Login.java as;

```
package com.journaldev.jsf.beans;

import java.io.Serializable;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpSession;

import com.journaldev.jsf.dao.LoginDAO;
import com.journaldev.jsf.util.SessionUtils;

@ManagedBean
@SessionScoped
public class Login implements Serializable {

	private static final long serialVersionUID = 1094801825228386363L;

	private String pwd;
	private String msg;
	private String user;

	public String getPwd() {
		return pwd;
	}

	public void setPwd(String pwd) {
		this.pwd = pwd;
	}

	public String getMsg() {
		return msg;
	}

	public void setMsg(String msg) {
		this.msg = msg;
	}

	public String getUser() {
		return user;
	}

	public void setUser(String user) {
		this.user = user;
```

```
	}

	//validate login
	public String validateUsernamePassword() {
		boolean valid = LoginDAO.validate(user, pwd);
		if (valid) {
			HttpSession session = 
SessionUtils.getSession();
			session.setAttribute("username", user);
			return "admin";
		} else {

	FacesContext.getCurrentInstance().addMessage(
					null,
					new 
FacesMessage(FacesMessage.SEVERITY_WARN,
							"Incorrect 
Username and Passowrd",
							"Please enter 
correct username and Password"));
			return "login";
		}
	}

	//logout event, invalidate session
	public String logout() {
		HttpSession session = SessionUtils.getSession();
		session.invalidate();
		return "login";
	}
}
```

Step 5: Now create the LoginDAO java class

```
package com.journaldev.jsf.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.journaldev.jsf.util.DataConnect;

public class LoginDAO {

	public static boolean validate(String user, String password) {
		Connection con = null;
```

```
                PreparedStatementps = null;

                try {
                        con = DataConnect.getConnection();
                        ps = con.prepareStatement("Select uname,
password from Users where uname = ? and password = ?");
                        ps.setString(1, user);
                        ps.setString(2, password);

                        ResultSetrs = ps.executeQuery();

                        if (rs.next()) {
                                //result found, means valid inputs
                                return true;
                        }
                } catch (SQLException ex) {
                        System.out.println("Login error -->" +
ex.getMessage());
                        return false;
                } finally {
                        DataConnect.close(con);
                }
                return false;
        }
}
```

Step 6: Create the DataConnect.java class

```
package com.journaldev.jsf.util;

import java.sql.Connection;
import java.sql.DriverManager;

public class DataConnect {

        public static Connection getConnection() {
                try {
                        Class.forName("com.mysql.jdbc.Driver");
                        Connection con =
DriverManager.getConnection(

        "jdbc:mysql://localhost:3306/cardb", "pankaj", "pankaj123");
                        return con;
                } catch (Exception ex) {
                        System.out.println("Database.getConnection()
Error -->"
                                        + ex.getMessage());
```

```
                return null;
            }
        }

        public static void close(Connection con) {
            try {
                con.close();
            } catch (Exception ex) {
            }
        }
}
```

Step 7: Create SessionUtils.java to obtain and manage session related user information.

```
package com.journaldev.jsf.beans;

import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

public class SessionUtils {

        public static HttpSessiongetSession() {
            return (HttpSession)
FacesContext.getCurrentInstance()

        .getExternalContext().getSession(false);
        }

        public static HttpServletRequestgetRequest() {
            return (HttpServletRequest)
FacesContext.getCurrentInstance()
                    .getExternalContext().getRequest();
        }

        public static String getUserName() {
            HttpSession session = (HttpSession)
FacesContext.getCurrentInstance()

        .getExternalContext().getSession(false);
            return session.getAttribute("username").toString();
        }

        public static String getUserId() {
            HttpSession session = getSession();
            if (session != null)
                return (String) session.getAttribute("userid");
```

```
                else
                        return null;
        }
}
```

Step 8: Create the authorization filter class as;

```
package com.journaldev.jsf.filter;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebFilter(filterName = "AuthFilter", urlPatterns = { "*.xhtml" })
public class AuthorizationFilter implements Filter {

        public AuthorizationFilter() {
        }

        @Override
        public void init(FilterConfigfilterConfig) throws
ServletException {

        }

        @Override
        public void doFilter(ServletRequest request, ServletResponse
response,
                        FilterChain chain) throws IOException,
ServletException {
                try {

                        HttpServletRequestreqt = (HttpServletRequest)
request;
                        HttpServletResponseresp = (HttpServletResponse)
response;
                        HttpSessionses = reqt.getSession(false);

                        String reqURI = reqt.getRequestURI();
```

```
                if (reqURI.indexOf("/login.xhtml") >= 0
                        || (ses != null
&&ses.getAttribute("username") != null)
                        || reqURI.indexOf("/public/") >= 0
                        ||
reqURI.contains("javax.faces.resource"))
                    chain.doFilter(request, response);
                else
                    resp.sendRedirect(reqt.getContextPath() +
"/faces/login.xhtml");
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }

        @Override
        public void destroy() {

        }
}
```

Step 9: Create admin.xhtml as;

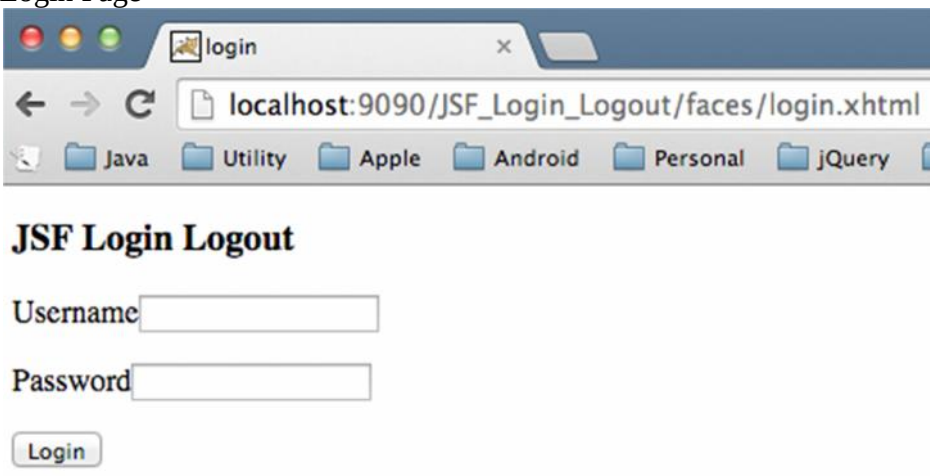
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
 "https://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="https://www.w3.org/1999/xhtml"
      xmlns:h="https://java.sun.com/jsf/html">
<h:head>
      <title>Facelet Title</title>
</h:head>
<h:body>
      <h:form>
            <p>Welcome #{login.user}</p>
            <h:commandLink action="#{login.logout}"
value="Logout"></h:commandLink>
      </h:form>
</h:body>
</html>
```

Step 10: Create faces-config.xml file as;

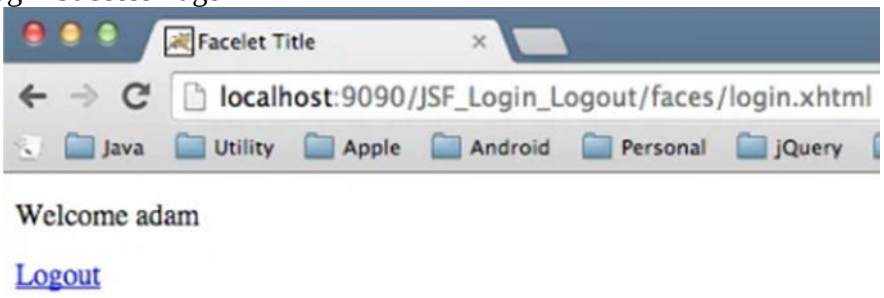
```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
xmlns="https://xmlns.jcp.org/xml/ns/javaee"
      xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="https://xmlns.jcp.org/xml/ns/javaee
        https://xmlns.jcp.org/xml/ns/javaee/web-
facesconfig_2_2.xsd">

        <navigation-rule>
                <from-view-id>/login.xhtml</from-view-id>
                <navigation-case>
                        <from-outcome>admin</from-outcome>
                        <to-view-id>/admin.xhtml</to-view-id>
                </navigation-case>
        </navigation-rule>

</faces-config>
```

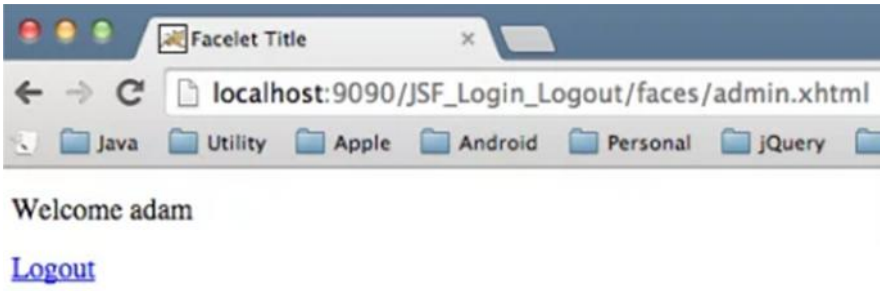
Login Page



Login Success Page



Accessing admin.xhtml while logged in



6.10 SUMMARY

1. JSF is a Java standard technology for building component-based, event-oriented web interfaces.
2. JSF is an XML document that represents formal components in a logical tree.
3. JSF components are backed by Java objects, which are independent of the HTML and have the full range of Java abilities, including accessing remote APIs and databases.
4. JavaServer Faces is a standardized display technology, which was formalized in a specification through the Java Community Process.
5. JSF reduces the effort in creating and maintaining applications, which will run on a Java application server and will render application UI on to a target client.
6. JSF provides the developers with the capability to create Web applications from collections of UI components that can render themselves in different ways for multiple client types.

REFERENCE FOR FURTHER READING

1. Java EE 6 for Beginners, Sharanam Shah, Vaishali Shah, SPD
2. Core Java Vol. II – Advanced Features, Cay S. Horstmans, Gary Coronell, Eight Edition, Pearson
3. https://docs.oracle.com/javaee/6/tutorial/doc/gijtu.html
4. https://docs.oracle.com/javaee/7/tutorial/jsf-intro006.htm

UNIT END EXERCISES

1. What is the need of MVC in JSF?
2. What are the Components of JSF?
3. Explain the JSF lifecycle?
4. Write a short note on JSF configuration?

*****

7

ENTERPRISE JAVA BEAN (EJB)

Unit Structure

7.1 OBJECTIVE

- To understand the whole system broken into layers, so each layer is independent and serves a distinct purpose & each layer is made up of one and more logical components.
- To manage the complexity of software projects in amore easier way.
  - To follow great discipline to the software development process.
  - To reduce the development time.
  - Build EJB technology-based distributed systems.
  - Create entity beans.
  - Create session beans.
  - Deploy solutions in a server.
  - Create standalone enterprise bean clients.
  - Use entity beans from within session beans.

7.2 INTRODUCTION

- The Enterprise JavaBeans (EJB) specification describes architecture for the development and deployment of transactional, distributed object applications based, server-side software components.
- Organizations can develop their own components or purchase components from third party vendors.
- These server-side components, called enterprise beans, are distributed objects that are hosted in Enterprise JavaBean containers and provide remote services for clients distributed all-round the network.

7.3 ENTERPRISE BEAN ARCHITECTURE

Enterprise Bean architecture defines a model for the development and deployment of reusable java server components. EJB component architecture is the backbone of the java EE platform.

- **EJB Architecture is composed of:**

1. An Enterprise Bean Server.
2. Enterprise Bean Containers that runs on these servers.
3. Enterprise Beans that run in these containers.
4. Others Systems Such as Java Naming and Directory Interface [JNDI] and Java Transaction service [JTS].

- **Request And Response:**

The communication between the components happens as follow:

1. Client object makes a request for a method that is available in the bean.
2. Container comes into picture and checks whether the client is in the approved list for calling a method on the bean.
3. If the client is authorized, the container either creates a new instance or activates the requested bean from the pool.
4. Container ensures the bean gets its own new transaction.
5. Container informs the EJB object (wrapper class generated by container) that the beans ready and passes the client's method request to the bean.

Enterprise Bean Server:

- An Enterprise Java Beans (EJB) Server is a Component Transaction Server.
- It Supports the EJB server side component model for developing and deploying distributed enterprise -level applications in a multi-tiered environment.

An Enterprise Bean Server provides:

- The framework for creating, Deploying and managing middle -tier business logic.
- An Environment that allows the execution of applications developed using Enterprise Java Beans (EJB) components.

In a three -tier environment:

- The client provides the user Interface logic.
- The business rules are split to the middle tier.
- The database is the information repository.
- The client does not access the database directly. Instead, the client makes a call to the EJB server on the middle tier, which then accesses the database.

An EJB server takes care of:

- Managing and coordinating the allocation of resources to the applications
- security
- threads
- Connection pooling
- Access to a distributed transaction management service
- The EJB server provides one or more containers for the enterprise beans, which is called an EJB Container
- An EJB container handles the enterprise beans contained within it.
- Enterprise beans or components are reusable modules of code that combine related tasks or methods into a well-defined interface. These enterprise beans EJB components hold the methods that execute business logic and access data sources.
- These components (i.e. the executable code) are installed on an EJB Server. All number of independent Java or EJB applications or clients can use the EJBs.

An Enterprise bean (EJP):

- Business components developed using the EJB architecture are called Enterprise javaBeans components or simply Enterprise Beans.
- An Enterprise Bean (EJB) is a server-side component that encapsulates the code that fulfills the purpose of the application. They can be merged with other components and rapidly produce a custom application.
- The fundamental purpose of introducing EJB was for building distributed components. EJB xx introduced a promise to solve all issues and complexities of CORBA.

- EJB technology is powerful and sophisticated. The technology helps developers to build business applications that support a very large number of users simultaneously. They applications developed with EJB are capable of maintaining data integrity even though data is processed concurrently by multiple users, thus making them transaction enabled.
- The EJB component is assembled and reassembled into different distributed application

EJB Evolution:

- EJB specification evolved significantly.
- It is capable of embracing the requirement of many enterprises.
- EJB is complicated due to:
  - Lack of good persistence strategy
  - Long and tedious deployment descriptors.
  - Limited capacity of testing.

7.4 BENEFITS OF ENTERPRISE BEAN:

1. EJB container provides system-level services to enterprise beans, the bean developer can focus on solving business problems.
2. The responsibility of EJB containers for system-level services, such as transaction management and security authorization.
3. The beans rather than the clients consist of the application's business logic; the client developer can concentrate on the presentation of the client.
4. The clients are thinner, a benefit that is especially important for clients that run on small devices.
5. Enterprise beans are portable components; the application assembler can develop new applications from existing one. They use the standard APIs, these applications can run on any compliant Java EE server.

7.5 TYPES OF BEANS

There are two types of EJBs: session beans and *message-driven beans*.

1. Session Beans
2. Message-Driven Beans

Session Beans:

- A session bean implements one or more business tasks.
- A session bean might consist of methods that query and update data in a relational table. Session beans are frequently used to implement different services.

- For example, an application developer implements one or several session beans that retrieve and update inventory data in a database.
- A session bean implements the **javax.ejb.SessionBean** interface, following is the definition:

```
public interface javax.ejb.SessionBean extends javax.ejb.EnterpriseBean
{
  public abstract void ejbActivate();
  public abstract void ejbPassivate();
  public abstract void ejbRemove();
  public abstract void setSessionContext(SessionContextctx);
}
```

- An EJB must implement the following methods, as specified in the javax.ejb.SessionBean interface:

ejbCreate()	The container invokes this method right before it creates the bean.
ejbActivate()	The container invokes this method right after it reactivates the bean.
ejbPassivate()	The container invokes this method right before it passivates the bean.
ejbRemove()	A container invokes this method before it ends the life of the session object.
setSessionContext	This method associates a bean instance with its context information.

There are two types of session beans:

- **Stateless Session Beans**

Stateless session beans do not share state or specification between method invocations. They are effective mainly in middle-tier application servers that provide a pool of beans to process frequent and brief requests.

- **Stateful Session Beans:**

Stateful session beans are useful for informal sessions, in which it is necessary to maintain state of the bean, like instance variable values or transactional state, between method invocations. These session beans are depicted to a single client for the life of that client.

Stateless Session Beans:

- A stateless session bean does not keep any state for the client.

- It is strictly a single invocation bean. It is active for reusable business services that are not connected to any specific client, such as currency calculations, mortgage rate calculations etc.
- Stateless session beans may carry client-independent, read-only state across a call. Succeeding calls are controlled by other stateless session beans in the pool. The information is used only for the single invocation.

Implementation	Methods
Home Interface	Extends javax.ejb.EJBHome and requires a single create() factory method, with no arguments, and a single remove() method.
Remote Interface	Extends javax.ejb.EJBObject and defines the business logic methods, which are implemented in the bean implementation.
Bean implementation	Implements SessionBean. This class must be declared as public, contain a public, empty, default constructor, no finalize() method, and implement the methods defined in the remote interface.

Stateful Session Beans:

- A stateful session bean keeps up its state between method calls. Therefore, there is one instance of a stateful session bean created for each client.
- Each stateful session bean has its own identity and a one-to-one mapping with an individual client.
- The state of this type of bean is maintained across various calls through serialization of its state, called passivation.

Implementation	Methods
Home Interface	Extends javax.ejb.EJBHome and requires one or more create() factory methods, and a single remove() method.
Remote Interface	Extends javax.ejb.EJBObject and defines the business logic methods, which are implemented in the bean implementation.

Message-Driven Beans:

- Message-Driven Beans (MDB) provide a simple method to implement asynchronous communication than using straight JMS. MDBs were created to receive asynchronous JMS messages.
- The container controls much of the setup required for JMS queues and topics.
- A MDB is the same as a stateless session bean because it does not save conversational state and is used for handling various incoming requests. Instead of handling direct requests from a client, MDBs handle requests placed on a queue. Figure 1 shows how clients place requests on a queue. The container takes the requests off of the queue and sends the request to an MDB in its pool.

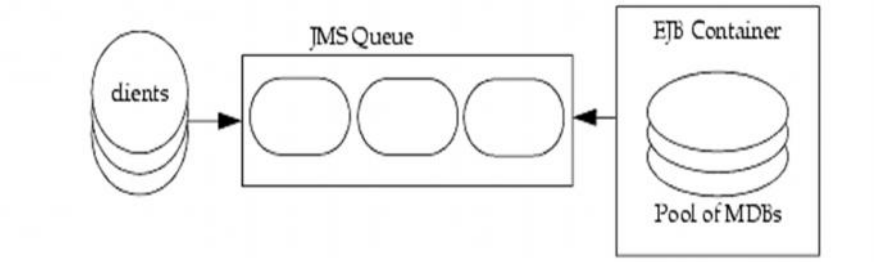


Figure 1 Message Driven Beans

- MDBs implement the javax.ejb.MessageDrivenBean interface, which also inherits the javax.jms.MessageListener methods. Within these interfaces, the following methods must be implemented:

Method	Description
onMessage(msg)	The container dequeues a message from the JMS queue associated with this MDB and gives it to this instance by invoking this method.
setMessageDrivenContext(ctx)	After the bean is created, the setMessageDrivenContext method is invoked.
ejbCreate()	This method is used just like the stateless session bean ejbCreate method. No initialization should be done in this method.
ejbRemove()	Delete any resources allocated within the ejbCreate method.

7.6 ACCESSING BEANS

- **Accessed of Enterprise Beans:**
  a. No-interface view
  b. Business interface

a. No-interface view:

- A no-interface view of an enterprise bean reveals the public methods of the enterprise bean implementation class to clients.
- Clients using the non-interface view of an enterprise bean may invoke any public methods in the enterprise bean implementation class.

b. Business interface:

- A Business interface is a standard java programming language interface that contains the business methods in the enterprise bean.
- A client can access a session bean through the methods defined in the bean's interface or through the public methods of an enterprise bean that has a no-interface view.

7.7 PACKAGING BEANS

Enterprise beans can be packaged in EJB JAR or WAR modules.

Packaging Enterprise Beans in EJB JAR Modules:

- EJB JAR file is portable.
- Used for different applications.
- To assemble a Java EE application, package one or more modules, such as EJB JAR files, into an EAR file, the archive file that holds the application.
- When deploying the EAR file that contains the enterprise bean's EJB JAR file, this also deploys the enterprise bean to GlassFish Server.
- Beans can also deploy an EJB JAR that is not contained in an EAR file. Figure 2 shows the contents of an EJB JAR file.

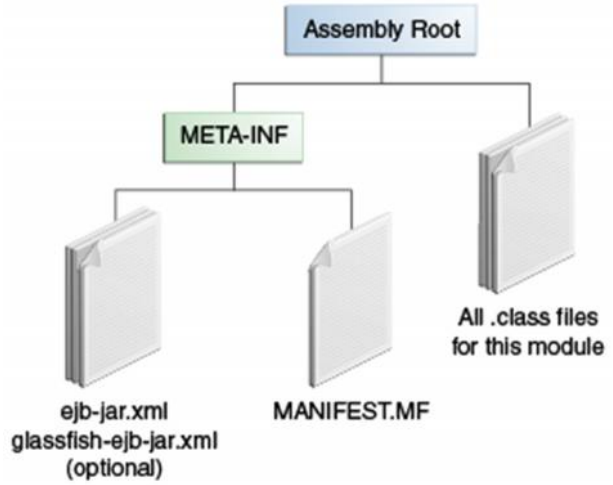


Figure 2 Structure of an Enterprise Bean JAR

Packaging Enterprise Beans in WAR Modules

- EJB provides the business logic of a web application.
- WAR module simplifies deployment and application organization.
- Enterprise beans may be packaged within a WAR module.
- To include, the class files should be in the WEB-INF/classes directory.
- To include a JAR file, add the JAR to the WEB-INF/lib directory of the WAR module.
- Enterprise beans do not require an ejb-jar.xml deployment descriptor.
- JAR files that contain enterprise bean classes packaged within a WAR module are not considered EJB JAR files.
- JAR files are semantically equivalent to enterprise beans.
- For example, The shopping cart bean exposes a local, no-interface view and is defined as follows:

```
package com.example.cart;
@Stateless
public class CartBean { ... }
```

The credit card processing bean is packaged within its own JAR file, cc.jar, exposes a local, no-interface view, and is defined as follows:

```
package com.example.cc;
@Stateless
public class CreditCardBean { ... }
```

7.8 CREATING WEB APPLICATIONS

- To develop, deploy, and run a simple web application named currency converter. The purpose of the converter is to calculate currency conversions between Japanese yen and Eurodollars.
- Currency converter consists of an enterprise bean, which performs the calculations, and two types of clients: an application client and a web client.

Steps:

1. Create the enterprise bean: CurrencyConverterBean.
2. Create the application client: CurrencyConverterClient.
3. Create the web client in converter-war.
4. Deploy converter onto the server.
5. Run the application client.
6. Using a browser, run the web client.

7.9 CREATING ENTERPRISE BEAN

Creating the Enterprise Bean:
The enterprise bean in our example is a stateless session bean called CurrencyConverterBean.
Creating CurrencyConverterBean requires these steps:

1. Coding the bean's business interface and class (the source code is provided)
2. Compiling the source code with the Ant tool

Enterprise Bean:
The enterprise bean in this example needs the following code:

- Remote business interface
- Enterprise bean class

Business Interface:

The **business interface** defines the business methods that a client can call. The business methods are implemented in the enterprise bean class. The source code for the CurrencyConverter remote business interface follows.

```
package com.sun.tutorial.javaee.ejb;

import java.math.BigDecimal;
import javax.ejb.Remote;

@Remote
public interface CurrencyConverter {
   public BigDecimaldollarToYen(BigDecimal dollars);
   public BigDecimalyenToEuro(BigDecimal yen);
}
```

Note the @Remote annotation decorating the interface definition. This lets the container know that CurrencyConverterBean will be accessed by remote clients.

Enterprise Bean Class:
The enterprise bean class for this example is called CurrencyConverterBean. This class implements the two business methods (dollarToYen and yenToEuro) that the CurrencyConverter remote business interface defines. The source code for the CurrencyConverterBean class follows.

```
package com.sun.tutorial.javaee.ejb;
```

```
import java.math.BigDecimal;
importjavax.ejb.*;

@Stateless
public class CurrencyConverterBean implements CurrencyConverter {
   private BigDecimalyenRate = new BigDecimal("115.3100");
   private BigDecimaleuroRate = new BigDecimal("0.0071");

   public BigDecimaldollarToYen(BigDecimal dollars) {
BigDecimal result = dollars.multiply(yenRate);
      return result.setScale(2, BigDecimal.ROUND_UP);
   }

   public BigDecimalyenToEuro(BigDecimal yen) {
BigDecimal result = yen.multiply(euroRate);
      return result.setScale(2, BigDecimal.ROUND_UP);
   }
}
```

7.10 CREATING WEB CLIENT

Creating the converter Application Client:

An application client is a program written in the Java programming language. At runtime, the client program executes in a different virtual machine than the Application Server. For detailed information on the appclient command-line tool, see the man page at appclient.

The CurrencyConverterClient.java source code illustrates the basic tasks performed by the client of an enterprise bean:

- Creating an enterprise bean instance
- Invoking a business method

CurrencyConverterClient program follows.

```
packagecom.sun.tutorial.javaee.ejb;

importjava.math.BigDecimal;
importjavax.ejb.EJB;

public class CurrencyConverterClient {
   @EJB
private static CurrencyConverter converter;

publicCurrencyConverterClient(String[] args) {
```

```
    }

public static void main(String[] args) {
CurrencyConverterClient client = new CurrencyConverterClient(args);
client.doConversion();
    }

public void doConversion() {
try {
BigDecimalparam = new BigDecimal("100.00");
BigDecimalyenAmount = converter.dollarToYen(param);

System.out.println("$" + param + " is " + yenAmount
                + " Yen.");
BigDecimaleuroAmount = converter.yenToEuro(yenAmount);
System.out.println(yenAmount + " Yen is " + euroAmount
                + " Euro.");

System.exit(0);
       } catch (Exception ex) {
System.err.println("Caught an unexpected exception!");
ex.printStackTrace();
       }
    }
}
```

Compiling the converter Application Client

The application client files are compiled at the same time as the enterprise bean files, as described in Compiling and Packaging the converter Example.

7.11 CREATING JSP FILE

Creating the converter Web Client

```
<%@ page import="converter.ejb.CurrencyConverter,
java.math.*, javax.naming.*"%>

<%!
    private CurrencyConverter converter = null;
    public void jspInit() {
        try {
InitialContextic = new InitialContext();
converter = (CurrencyConverter)
ic.lookup(CurrencyConverter.class.getName());
       } catch (Exception ex) {
System.out.println("Couldn't create converter bean."+
ex.getMessage());
```

```
      }
   }

   public void jspDestroy() {
      converter = null;
   }
%>
<html>
<head>
<title>CurrencyConverter</title>
</head>

<body bgcolor="white">
<h1>CurrencyConverter</h1>
<hr>
<p>Enter an amount to convert:</p>
<form method="get">
<input type="text" name="amount" size="25">
<br>
<p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>

<%
         String amount = request.getParameter("amount");
         if ( amount != null &&amount.length() > 0 ) {
BigDecimal d = new BigDecimal(amount);

BigDecimalyenAmount = converter.dollarToYen(d);
      %>
<p>
<%= amount %> dollars are  <%= yenAmount %>  Yen.
<p>
<%
BigDecimaleuroAmount =
converter.yenToEuro(yenAmount);
      %>
<%= amount %> Yen are <%= euroAmount %>  Euro.
<%
         }
      %>
</body>
</html>
```

7.12 BUILDING AND RUNNING WEB APPLICATION

Running the converter Application Client:

When you run the application client, the application client container first injects the resources specified in the client and then runs the client. You can run the application client using either NetBeans IDE or Ant.

Running the converter Application Client Using NetBeans IDE:
Follow these instructions to run the application client using NetBeans IDE.

1. In NetBeans IDE, make sure the converter application is open.

In the Projects tab, right-click the converter project and select Run. You will see the following output in the Output tab:

…
$100.00 is 11258.00 Yen.
11258.00 Yen is 78.81 Euro.

2. …

Running the converter Application Client Using Ant
To run the application client using Ant, perform the following steps.
In a terminal window, go to this directory:
Type the following command:
ant run

appclient -client client-jar/converterClient.jar
In the terminal window, the client displays these lines:
...
$100.00 is 11531.00 Yen.
11531.00 Yen is 81.88 Euro.
1. ...

Running the converter Web Client:
To run the web client, point your browser at the following URL. Replace *host* with the name of the host running the Application Server. If your browser is running on the same host as the Application Server, you can replace *host* with localhost.

http://*host*:8080/converter

Figure 3 Currency Converters Web Client

7.13 SUMMARY

- Enterprise JavaBeans present a way to build components as well as a means to make these components exist in a transactional, secure, and distributed environment.
- However, a single bean represents only one component - and consequently only one part of a complete application.
- EJB provides developers with flexibility in determining how these components should be made to work together.
- There are a number of ways in which Enterprise JavaBeans can be made to work together to form a complete enterprise application.
- Top Link can be integrated into each variety of EJB application architecture to provide both the technology that enables these architectures and the features that add value to them.

REFERENCE FOR FURTHER READING

1. Java EE 6 for Beginners, Sharanam Shah, Vaishali Shah, SPD
2. https://docs.oracle.com/javaee/5/tutorial/doc/bnbnc.html

UNIT END EXERCISES

1. Explain the EJB Architecture?
2. What are the benefits of EJB?
3. What are the different types of enterprise beans?
4. Write a short note on accessing and packaging of beans?

*****

UNIT V

8

HIBERNATE AND STRUTS

Unit Structure

8.1 HIBERNATE: INTRODUCTION

Writing the application, Application development approach, creating database and tables in MySQL, creating a web application, Adding the required library files, creating a java bean class, creating hibernate configuration and mapping file, adding a mapping resource, creatingJSPs.

8.2 HIBERNATE

Hibernate is the latest Open-Source persistence technology. It is available as a free open source [distributed under the GNU Lesser General Public License] Object/Relational Mapping [ORM] library for the Java programming language. It provides a framework for mapping an object-oriented DOMAIN model to a traditional Relational Database.

Hibernate was developed by a team of Java software developers around the world led by Gavin King, JBoss Inc. [now pail of Red Hat].

Hibernate has become the de-facto ORM [Object/Relational Mapping] framework for most of the organizations today.

Hibernate was developed with a goal to relieve the developers from 95% of common data persistence related programming tasks by:

- Making the developer feel as if the database contains plain Java objects, without having them worry about how to get them out of [or back into] databasetables.
- Allowing the developers to focus on the objects and features of the application, without having to worry about how to store them or find them later.

Its primary feature is mapping from:

- Java Classes → DatabaseTables

- Java Data Types → SQL DataTypes

In addition to this, Hibernate also allows querying and retrieving data. It generates all the necessary SQL calls to achieve this and thereby, relieves the developers from manual result set handling and objet conversion.

Hibernate provides transparent persistence that enables the applications [using Hibernate as the ORM] to switch to any database [supported by Hibernate].

To use Hibernate :

- JavaBean classes [POJOs] that represents the table in the database are created.
- The instance variables of the class are mapped to the columns in the database table.

Why Hibernate?:
Because of the following reasons:

- Hibernate is Free under LGPL i.e. Hibernate can be used to develop/package and distribute the applications for free.
- It eliminates the need for repetitiveSQL.
- It allows working with classes and objects instead of queries and result sets which makes the approach more Object Oriented and less Procedural.
- Handles all Create, Read, Update, Delete [CRUD]operations.
- Brings in portability across databases.
- Supports IDEs such as Eclipse, NetBeans by providing a suite of plug-ins.
- Reduces the development time by supporting inheritance, polymorphism, composition and the Java Collection framework.
- Supports the multiple primary key generation including built-in support for identity [Auto increment] columns, sequences, UUID algorithm and HI/LO algorithm. Hibernate also includes support for application assigned identifiers and composite keys.
- Hibernate's Dual Layer Cache Architecture [HDCLCA] delivers thread safeness, nonblocking data access, session level cache, optional second- level cache and optional query cache. Hibernate also works well when other applications have simultaneous access to the database.
- Supports connectionpooling
- Supports wide range of database such as:

– Oracle	–DB2	–Sybase
– MS SQLServer	–PostgreSQL	–MySQL
– HypersonicSQL	–Mckoi SQL	– SAP DB

- Interbase
- FrontBase
- Firebird
- ApacheDerby
- CorelParadox
- Pointbase
- Ingres
- TimesTen
- HP NonStopSQL/MX
- Xbase
- Progress
- Informix
- InterSystemsCache
- MS Access

Architecture of Hibernate:

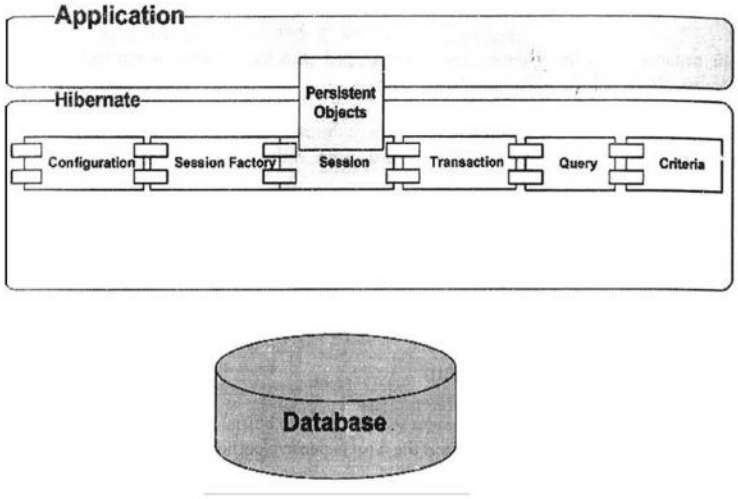


Fig. 1

Configuration Object:

The Configuration Object represents a configuration or properties file for Hibernate. It is usually created once during application initialization.

The Configuration object reads the properties [hibernate.properties/ hibernate.cfg.xml] to establish a database connection. Typically, a Configuration object is spawned to create a SessionFactory.

Session Factory:

The SessionFactory is created with the help of a Configuration object during the application start up. It serves as a factory for spawning Session objects when required.

Typically, it is created once [one SessionFactory object per database] and kept alive for later use. However, for applications that require interacting with multiple databases, multiple SessionFactory objects [each with a separate Configuration object].

Session:

Session objects are lightweight and inexpensive to create. They provide the main interface to perform actual database operations. All the POJOs i.e. persistent objects are saved and retrieved with the help of a Session object. Typically, session objects are created as needed and destroyed when not required.

Transaction:

A Transaction represents a unit of work with the database. Any kind of modifications initiated via the session object are placed with in a transaction. A Session object helps creating a Transaction object. Transaction objects are typically used for a short time and are closed by either committing or rejecting.

Query:

Persistent objects are retrieved using a Query object. Query objects allow using SQL or Hibernate Query Language [HQL] queries to retrieve the actual data from the database and create objects.

Criteria

Persistent objects can also be retrieved using a Criteria object. Criteria uses an object/method based means of constructing and executing a request to retrieve objects.

8.3 STRUCTURE OF HIBERNATE.CFG.XML FILE (HIBERNATE CONFIGURATIONFILE).

```
<?xml version=”1.0” encoding=”UTF–8”?>

<hibernate–configuration>
<session–factory>

<property name=”hibernate.dialect”>
org.hibernate.dialect.MySQLDialect</property>

<property name=”hibernate.connection.driver_class”>
com.mysql.jdbc.Driver</property>

<property name=”hibernate.connection.url”>
jdbc:mysql://localhost/DBname</property>

<property name=”hibernate.connection.username”> root</property>

<property name=”hibernate.connection.password”> root</property>

<mapping resource=”guestbook.hbm.xml”/>

</session–factory>
```

</hibernate–configuration>

Elements:
hibernate.dialect– It represents the name of the SQL dialect for the database

hibernate.connection.driver_class– It represents the JDBC driver class for the specific database

hibernate.connection.url– It represents the JDBC connection to the database

hibernate.connection.username– It represents the user name which is used to connect to thedatabase

hibernate.connection.password– It represents the password which is used to connect to thedatabase

guestbook.hbm.xml– It represents the name of mapping file

Structure Of Guestbook.Hbm.Xml File (Hibernate Mapping File).

```
<?xml version=”1.0” encoding=”UTF–8”?>
<hibernate–mapping>

<class name=”guestbook” table=”guestbooktable”>

<property name=”name” type=”string”>
<column name=”username” length=”50” />
</property>

<property name=”message” type=”string”>
<column name=”usermessage” length=”100” />
</property>

</class>

</hibernate–mapping>
```

Elements:

<hibernateJmapping> </hibernateJmapping>

It is the base tag which is used to write hibernate mapping file, which is used to map POJO class with database table.

<class> </class>

It represents name of the class and database table which we want to map with each other. It has 2 parameters:
nameJIt represents name of the class

tableJIt represents name of the database table

<property> </property>
It is used to write the property which we want to map with database column. It has 2 parameters:
nameJIt represents name of the property
typeJIt represents type of the property
<column> </column>
It is used to write the database column which we want to map with java class property. It has 2 parameters:
nameJIt represents name of the column
lengthJIt represents maximum length of a column value

8.3 STEPS FOR HIBERNATEPROJECT

I. Create the Database and Table structure

II. Create the Database Service through NetBeans IDE

- Click the Services Tab from the Project Workspace
- Right Click on the Databases option and select the option "New Connection"
- Select the Driver as "MySQL (Connector / J driver)
- Click Next → Type the Database Name (db) → specify the password of MySQL database → select remember password checkbox → click "test connection" to check the successful database connection → Next → Finish

III. Create a Hibernate project

- File -> New Project (Select Java Web and Web Application) → Next (Give the Project Name; change the project location as required; select the checkboxes Dedicated folder for storing libraries) → Next
- Select Glassfish Server → Next → Select the framework Hibernate 3.2.5(select the respective Database Connection) → Finish

IV. Adding the Hibernate Configuration and POJO

1. Adding Reverse Engineering File
   Right click on the project from the workspace → New → other → (Select Hibernate category and Hibernate Reverse Engineering Wizard file type) → Next → File name (hibernate.reveng) → Next → select the table name from the available tables option → click add(select the check box include related tables) → Finish
2. Adding Hibernate Configuration and POJO

3. Right click on the project from the workspace → New → other → (Select Hibernate category and Hibernate mapping files and POJOs from Database file type) → Next → Keep the default configuration file name(hibernate.cfg) and Hibernate Reverse Engineering File(hibernate.reveng) and type the package name(hibernate) → Finish

V. Add the required JSP's Servlet's Files

Example: Develop a Hibernate application to store Feedback of Website Visitor in MySQL Database.

Database commands in MySQL

1) create databasedb;
2) usedb;
3) create table guestbook(no int primary key auto_increment, name varchar(20),

msg varchar(100), dt varchar(40));

Guestbook.java

```
package hibernate;
 public class Guestbook implements
java.io.Serializable { private Integer no;
private String
name; private
String msg;
private String
dt;

public Guestbook() {
}

public Guestbook(String name, String msg,
String dt) { this.name = name;
this.msg =
msg; this.dt
= dt;


}

public Integer
getNo() { return
this.no;
}

public void setNo(Integer
no) { this.no = no;
}
public String
getName() { return
this.name;
}
```

```
public void setName(String
name) { this.name = name;
}
public String
getMsg() {
returnthis.msg;
}

public void setMsg(String
msg) { this.msg =msg;
}
public String
getDt() { return
this.dt;
}
public void
setDt(String dt) {
this.dt = dt;
}
}
```

hibernate.cfg.xml

```
<hibernate-configuration>
<session-factory>
   <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</p
roperty>
   <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</p
roperty>
   <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/db</p
roperty>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">tiger</property>
<mapping resource="hibernate/Guestbook.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Guestbook.hbm.xml

```
<hibernate-mapping>
<class name="hibernate.Guestbook" table="guestbook" catalog="db">
<id name="no" type="java.lang.Integer">
<column name="no" />
<generator class="identity" />
</id>

<property name="name" type="string">
<column name="name" length="20" />
</property>
```

```
<property name="msg" type="string">
<column name="msg" length="100" />
</property>

<property name="dt" type="string">
<column name="dt" length="40" />
</property>
</class>
</hibernate-mapping>
```

index.jsp

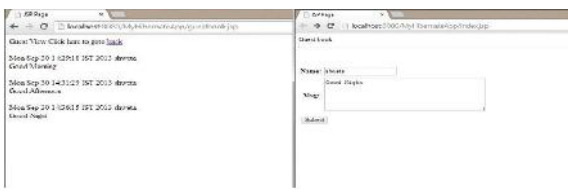
```
<html>
<head>
<title>Guest Book</title>
</head>
<body>
Guest Book <hr><br><br>
<form action="guestbookview.jsp" method="post">
Name <input type="text" name="name" maxlength="20"><br>
        Message          <textarea      rows="5"      cols="40"
                         maxlength="100" name="msg"></textarea>
<br><input type="submit" value="submit">
</form>
</body>
</html>
```

guestbookview.jsp

```
<%@page import="org.hibernate.SessionFactory"%>
<%@page import="org.hibernate.Session"%>
<%@page import="org.hibernate.cfg.Configuration"%>
<%@page import="org.hibernate.Transaction"%>
<%@page import="java.util.List"%>
<%@page import="java.util.Iterator"%>
<%@page import="hibernate.Guestbook"%>
<%!
SessionFactory sf;
org.hibernate.Session ss;
List<hibernate.Guestbook>
gbook;
%>
<%
sf = new
Configuration().configure().buildSessionFactory();
ss= sf.openSession();
Transaction tx=null;
Guestbook gb=new
Guestbook(); try
{
tx=ss.beginTransaction();
String
name=request.getParameter("name");
```

```
String
msg=request.getParameter("msg");
String dt=new java.util.Date().toString();
gb.setName(name);
gb.setMsg(m
sg);
gb.setDt(dt);
ss.save(gb);
tx.commit();
}
catch(Exception e){ out.println("Error"+e.getMessage()); }

try
{
ss.beginTransaction();
gbook=ss.createQuery("from
Guestbook").list();
}
catch(Exception e){ }

%>

<html>
<head>
<title>Guest View</title>
</head>
<body>
Guest View
Click here to go <a href="index.jsp"> BACK </a>
<br><br>
<%
Iterator
it=gbook.iterator();
while(it.hasNext())
{
Guestbook each=(Guestbook)it.next();
out.print(each.getDt()+" ");
out.print(each.getName()+"<br>");
out.print(each.getMsg()+"<br><hr>");
}
%
>
</body>
</html>
```

Output:



9

STRUTS

Unit Structure

9.1 INTRODUCTION TO STRUTS

The Javaworld is very vast. Web application development in this vast world has come a long way with several Integrated Development Environments such as NetBeans, Eclipse and So on which has made creating standard Java based Web applications quite easy.

The main Java based technologies that one commonly uses to develop Web applications are Servlet and JavaServer Pages [JSP].

Standard Application Flow

In a standard Java EE Web application:

1) Using Web based data entry form information is submitted to theserver
2) Such information is handed over to a Java Servlet or a JavaServer Page for processing
3) The Java Servlet or theJSP:

- Interacts with thedatabase
- Produces an HTMLresponse

As an application grows in complexity, it becomes more and more difficult to manage the relationship between the JSP pages, the backend business logic and the forms and validations. Developers start finding it increasingly difficult to maintain and add additional functionality to the applications.

Both the technologies [Java Servlets and JSP] mix the application and the business logic with the presentation layer and thus make maintenance very difficult. This is not suitable for large enterprise applications. This means there's something still missing in these technologies which create a gap.

In such scenarios, most experienced developers, split various pieces of an application's functionality into small manageable pieces of code spec.

These small pieces of code spec hold a single piece of functionality and when taken together as a whole forms the basis for an application development framework.

FRAMEWORK

A framework is a collection of services that provide developers with common set of functionality, which can be reused and leveraged across multiple applications.

A framework usually come into existence by:

- Making generalizations about the common tasks and workflow of a specific domain
- Providing a platform upon which applications of that domain can be more quickly built

A framework helps automate all the tedious tasks of the domain and provides an elegant architectural solution to the common workflow of the domain.

A framework allows developers to focus on coding the business logic and the presentation layer of the application and not the overhead jobs such as heavy code spec to capture user input or to generate drop down list boxes.

Nowadays with several frameworks available, application development projects no longer begin with the question: **Should we use a framework?**

Instead, they begin with: **Which framework should we use? Why Struts?**

Struts, a Java based framework, allows a clean separation between the application logic and interacts with a database from the HTML pages that form the response.

It cuts time, out of the development process and makes developers more productive by giving them prebuilt components to assemble Web applications from.

Struts is not a technology, it’s a framework that can be used along with Java based technologies.

Struts makes the development of enterprise Web application development easier by providing a flexible and extensible application architecture, custom tags and a technique to configure common workflows within an application.

The Struts framework is a strong implementation of the widely recognized Model View Controller **design pattern**. The key focus of MVC pattern is separation which is what is desired.

Struts

Struts is an application Framework for building Web based applications in Java using the Java Enterprise Edition platform.

Struts [formerly located under the Apache Jakarta Project] was originally developed by Craig McClanahan and donated to the Apache Foundation in May, 2000. It was formerly known as Jakarta Struts. Struts is maintained as a part of Apache Jakartaproject.

Struts comes with an Open-Source license which means it has no cost and its users have free access to all its internal source code.

Today, the Apache Struts Project offers the two major versions of the Struts framework:

- **Struts 1** Which was recognized as the most popular Web application framework forJava

- **Struts 2** originally known as WebWork 2 is now the best choice which provides elegant solutions to complexproblems.

9.2 STRUTS FRAMEWORK CORE COMPONENTS

After understanding the fundamentals of Struts 2, it's time to learn about the CORE components of Struts 2.

This chapter takes a complete and comprehensive look at each CORE component of the Struts 2 architecture along with an in–depth explanation of the roles these components play in the framework.

These components make up the functionality of the application as well as the framework itself.

In Struts 2 the Model–View–Controller design pattern is recognized with the following CORE components:

- MODEL –Actions
- VIEW – Results and Result Types [ViewTechnologies]
- CONTROLLER – FilterDispatcher
- Interceptors
- Value Stack /OGNL

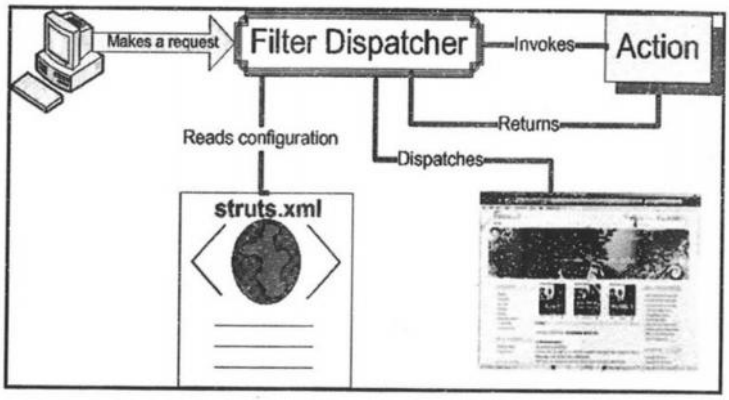


Fig. 2

Figure 2 indicates how the MVC components in Struts 2 interact with each other.

Let's begin with the Controller.

Filter Dispatcher

The controller is the first component that takes charge when processing a request. In Struts 2 the Filter Dispatcher plays the role of the Controller.

Actions

Action [the Action class] is heart and soul of the Struts 2 framework. It processes input and interacts with other layers of the application.

It is associated with a HTTP request received from the user. Each HTTP request [in form of a URL] is mapped to a specific Action. This Action class holds the code spec that helps serve the user request.

Actions are used to:

- Encapsulate the actual work to be done for a given request
- Server as a data carrier from the request to the view
- Assist the framework in choosing the response that has to be sent to the user i.e. view

Interceptors

Interceptors allow developing code spec that can be run before and/or after the execution of an action.

A request usually processed as follows:

- A user requests a resource that maps to an Action
- The Struts 2 framework invokes the appropriate Action to serve the request

If interceptors are written, available and configured, then:

- Before the Action is executed, the invocation could be intercepted by another object
- After the Action executes, the invocation could be intercepted again by another object

Such objects who intercept the invocation are called Interceptors.

Conceptually, Interceptors are very similar to Servlet Filters or the JDKs Proxy class.

Why Interceptors?
Interceptors are one of the best aspects of the Struts 2 framework Interceptors are mainly used to encapsulate common functionality in a reusable form that can be applied to one or more Actions in the application.

Developers can define code spec that can be executed before and/or after the execution of an action. This thus Intercepting can be done:

- Before the action
- After the action

Value Stack/OGNL
Now that the Action and Interceptor components are covered, let's move on to the next component in the Struts 2 framework.

Value Stack
Value Stack is exactly what the name suggests a stack of objects.
The Value Stack is a storage area that holds all of the data associated with the processing of a Request.

Accessing Value Stack
The Value Stack can be accessed by simply using the tags provided for JSP. When the Value Stack is queried for an attribute value, each stack element, in the provided order, is asked whether it holds the queried property.

If it holds the queried property, then the value is returned.
If it does not hold the queried property, then the next element down is queried.

This continues until the last element in the stack is scanned. This is very useful as the developer need not know where the attribute value currently is. Is it available in:

- The Action
- The Model
- The HTTP Request

The developer simply needs to know that such a value exists somewhere and Struts 2 returns it!

OGNL [ObjectJGraph Navigation Language]

OGNL [Object–Graph Navigation Language] is a fully featured expression language (the default expression language) for Retrieving and Setting properties of the Java objects. It helps data transfer and type conversion.

In the Value Stack, searching or evaluating, a particular expression, can be done using OGNL. OGNL provides a mechanism to navigate object graphs using a dot notation and evaluate expressions, including calling methods on the objects being retrieved.

OGNL supports:

- Type conversion
- Collection manipulation
- Projection across collections
- Lambda expressions
- Calling methods
- Generation
- Expression evaluation

RESULTS AND RESULT TYPES [VIEW TECHNOLOGIES]

After the Action completes its job, resulting information needs to be sent back to the user as a Response.

In Struts 2, this task is split into the Result Type and the Result itself.

Results

Results and Result Types come into picture only after the processing of the Action is complete.

Results define what happens next after the Action has been executed.

For example, Results can help determine, if the control is shifted to success view, an error view or back to the date entry input view.

The method of the Action class that processes the Request returns a String as the results. The value of the String is used to select a Result element. This return value is mapped via the configuration file to an implementation of the Result interface.

The XML [struts.xml] configuration:

<action name=”MyFirstStruts2App” class=“book.MyFirstStruts2App”>

<result name =“SUCCESS”>/apps/WelcomeToStruts.jsp</result>

<result name=”ERROR”>/apps/SorryNoEntry.jsp</result>

</action>

View Technologies
The most common way of rendering Results [View Technology] is JavaServer Pages [JSP]. However, JSP is not the only view technology. There are a few others that can replace JSP in a Struts 2 application:

- Velocity Templates
- Freemarker Templates
- XSLT Transformations

Freemarker and Velocity are very similar to JSP. In terms of configuration, the name of the JSP template is simply replaced with then ameof either the Velocity or Freemarker template in the actions configuration file.

For example, the XML [struts.xml] configuration for Fremarker template would be:
<action name="MyFirstStruts2App" class = "book.MyFirstStruts2App">
<result type="freemarker" name=
"SUCCESS">/apps/WelcomeToStruts.ftl
</results>
<result type= "freemarker" name=
"ERROR">/apps/SorryNoEntry.ftl/result>
</action>

The XSLT result is a little different. Instead of replacing the template name with the style sheet name, additional parameters are used.

Result Types
The response that the Result interface generates can vary between different concrete class implementations which are nothing but Result Types.

The Result Type provides the implementation details for the type of information that is returned to the user.

For example, a response could modify the HTTP response codes, generate a byte array for an image or render a JSP and soon.

This completes a comprehensive look at the Core Components. Let's setup the required development environment to start off.

9.3 STRUCTURE OF STRUTS.XML FILE

<?xml version="1.0" encoding="UTF–8"?>
<struts>
<package name="/" extends="struts–default">
<action name="test" class="hello.test">

<result name="SUCCESS">/welcome.jsp</result>
</action
>
</packa
ge>
</struts>

Elements:
<struts> .....</struts>
It represents the base tag for struts.xml file. This file contains all of the routing and configuration information for the Struts application.

<package> .....</package>
It allows separation and modularization of the configuration. This is very useful when you have a large project and project is divided into different modules. It has 2 parameters:

nameJIt represents the package name

extendsJIt represents which package does this package extend from

<action>..... </action>
It represents the action class. It has 2 parameters:

nameJIt represents name of the action

classJIt represents the name of the action class

<result> .....</result>
It represents the name of the result or a view. It has 1 parameter:

nameJIt represent a result for which appropriate view will get displayed

9.4 STEPS FOR HIBERNATE PROJECT

I Create a Struts project

- File -> New Project (Select Java Web and Web Application) → Next (Give the Project Name; change the project location as required; select the checkboxes Dedicated folder for storing libraries) → Next
- Select Glassfish Server → Next → Select the framework Struts 1.3.10(select the check box Add struts TDL) → Finish

II. Creating Views

- To add views Right click on the project -> New -> JSP (Add the required number of JSP pages) eg – (login, success, failure)

III. Writing business login(Model)

A. Design Action bean

- To add StructsActionForm bean, Right click on the project -> other -> select(Struts and StrutsActionFormBean) ->next
- Give the bean class name (bean1) -> select the package name from the list ->Finish
- Include the variables (username and email) into the action bean and insert setter and gettermethod

B. Design Action

- To add Action, Right click on the project -> other -> select(Struts and StrutsAction) ->next
- Give the class name (LoginAction) -> select the package from the list -> type the action path as (/login) ->next
- Select the action form bean(bean1) -> keep input resource blank -> select the scope as request -> uncheck validate ActionForm Bean checkbox ->Finish
- Modify the execute()method.

IV. Configuring controllers and mapping instruts-config.xml

A. Configuration File

- Select the struts-config.xml file from the configuration files
- Right click within the <action>tag
- Select Struts ->Add forward
- Select the forward name (success) and select the resource file(success.jsp)
- Repeat and Add all the forwards
  B. Mapping file
- Select the web.xml file from the configuration files
- Click the page stag
- Select the welcome files(login.jsp)

Example: Develop a simple Struts Application to Demonstrate E-mail Validator <u>Login.jsp:</u>

```
<html>
<head>
<title>LoginPage</title>
</head>
<body>
<form name="login" action="login.do">
<br>Username
```

```
<input type="text" name="username" value="">
<br>Email ID
<input type="text" name="email" value="">
<br>
<input type="submit" name="submit" value="submit">
</form>
</body>
</htmtml> Success.jsp:
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<html>
<head>
<title>Success Page</title>
</head>
<body>
Email Validation is done...
<br>Your entered details are
<br>Username
<bean:write name="bean1" property="username" />
<br>email
<bean:write name="bean1" property="email" />
</body>
</html> Failure.jsp:
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>

<html>
<head>
<title>Failure Page</title>
</head>
<body>
Wrong Email ID..
<br>Your entered details are
<br>Username
<bean:write name="bean1" property="username" />
<br>email
<bean:write name="bean1" property="email" />
</body>
</html>
```

Struts-config.xml

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

<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration
1.3//EN" "http://jakarta.apache.org/struts/dtds/struts-
config_1_3.dtd">

<struts-config>
<form-beans>
<form-bean name="bean1" type="com.myapp.struts.bean1"/>
```

```
</form-beans>
<global-exceptions>

</global-exceptions>

<global-forwards>
<forward name="welcome"path="/Welcome.do"/>
</global-forwards>

<action-mappings>
      <action name="bean1" path="/login"
scope="request"
type="com.myapp.struts.loginaction1"
validate="false">
<forward name="login" path="/login.jsp"/>
<forward name="success" path="/success.jsp"/>
<forward name="failure" path="/failure.jsp"/>


</action>
<action path="/Welcome" forward="/welcomeStruts.jsp"/>
</action-mappings>

<controller
processorClass="org.apache.struts.tiles.TilesRequestProcessor"/>

<message-resources
parameter="com/myapp/struts/ApplicationResource"/>


<plug-in className="org.apache.struts.tiles.TilesPlugin" >
<set-property property="definitions-config" value="/WEB-INF/tiles-
defs.xml"
/>
<set-property property="moduleAware" value="true" />
</plug-in>

<!-- ========================== Validator plugin
================================== -->
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
<set-property
property="pathna
mes"
value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>

</struts-config>
```

Bean1.java(form bean)

```
package com.myapp.struts;
```

```
import javax.servlet.http.HttpServletRequest;

import
org.apache.struts.action.ActionErro
rs; import
org.apache.struts.action.ActionMap
ping; import
org.apache.struts.action.ActionMes
sage;
public class bean1 extends org.apache.struts.action.ActionForm {

   private String
username;
private String
email;

public String
getEmail() {
return email;
}

public void
setEmail(String email) {
this.email = email;
}

public String
getUsername() {
return username;
}

public void setUsername(String
username) { this.username =
username;
}

}
```

Loginaction1.java

```
package com.myapp.struts;

import
javax.servlet.http.HttpServletReque
st; import
javax.servlet.http.HttpServletRespo
nse; import
org.apache.struts.action.ActionFor
m; import
```

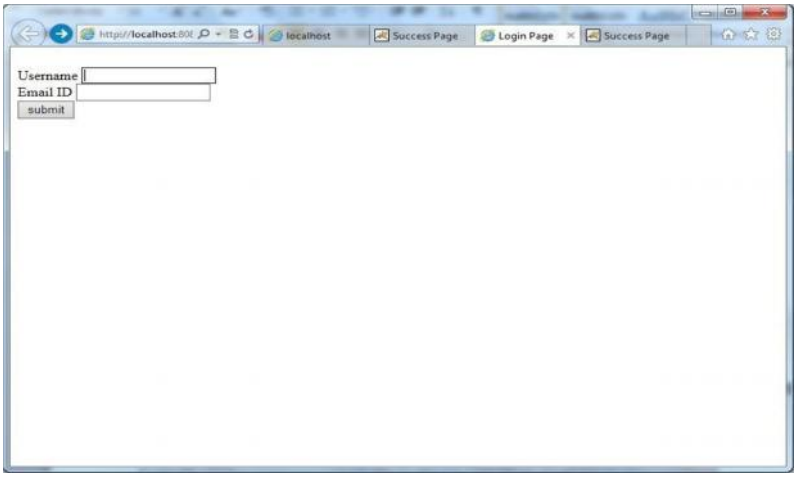
```
org.apache.struts.action.ActionFor
ward; import
org.apache.struts.action.ActionMap
ping;
public class loginaction1 extends
org.apache.struts.action.Action {
private static final String
SUCCESS = "success";
private static final String FAILURE = "FAILURE";

@Override
public ActionForward execute(ActionMapping mapping,
ActionForm form, HttpServletRequest request,
HttpServletResponse response)
throws Exception {
      bean1
b1=(bean1)form;
String
m=b1.getEmail();
if (m == null || m.equals("") || m.indexOf("@")
== -1) { return
mapping.findForward(FAILURE);
}
else{

return mapping.findForward(SUCCESS);
}

}
}
```

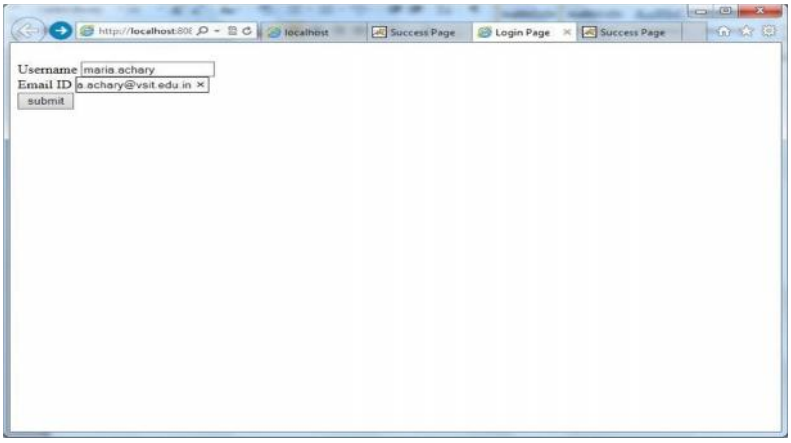
Web.xml

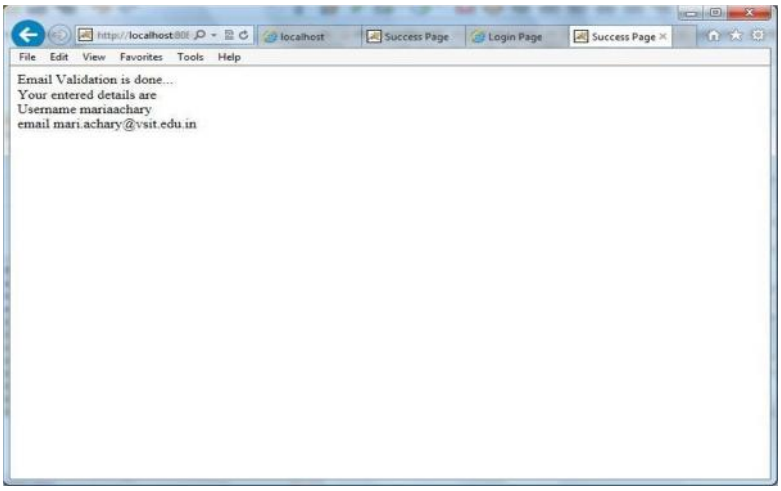
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app              version="3.0"
                      xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaeehttp://java.sun.co
m/xml/ns/javaee/web-app_3_0.xsd">
<servlet>
<servlet-name>action</servlet-name>
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
<init-param>
<param-name>config</param-name>
<param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
<init-param>
<param-name>debug</param-name>
<param-value>2</param-value>
</init-param>
<init-param>
```

```
<param-name>detail</param-name>
<param-value>2</param-value>
</init-param>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>action</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
<session-config>
<session-timeout> 30
</session-timeout>
</session-config>
<welcome-file-list>
<welcome-file>login.jsp</welcome-file>
</welcome-file-list>
<jsp-config>
 <ta
glib
  >
<taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/WEB-INF/struts-nested.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
</ta
glib
  >
 </jsp-
config
    >
</web-app>
```

Output:


Success Page
Login Page
Success Page
Username
Email ID
submit


Success Page
Login Page
Success Page
Username maria.achary
Email ID a.achary@vsit.edu.in
submit


localhost
Success Page
Login Page
Success Page
File Edit View Favorites Tools Help
Email Validation is done...
Your entered details are
Username mariaachary
email mari.achary@vsit.edu.in

*****

UNIT VI

10

WEBSERVICES, JAVAMAIL AND JNDI

Unit Structure

10.1 OBJECTIVES

At the end of this unit learner will be able to

- Describe the concept of webservice
- Demonstrate the concept of JAX-WS
- Illustrate the mean of SOAP
- Explain the concept of Java Mail and JNDI

10.2 INTRODUCTION

1 Web service is a technology to communicate one programming language with another. For example, java programming language can interact with PHP and .Net by using web services. In other words, web service provides a way to achieve interoperability.

2. The Javamail is an API that is used to compose, write and read electronic messages(emails).

3. The Javamail API provides protocol-independent and platform-independent framework for sending and receiving mails.
4. A fundamental element in every application is the capability to find and locate components and services.
5. The component name correspond to the actual name, typically much more difficult to remember and manage. A well-known naming service is provided on Internet by DNS.

10.3 WEB SERVICES

10.3.1 SOAP:

1. There are three major web service components

1. SOAP

2.WSDL

3.UDDI

10.1 SOAP:

1. SOAP is an acronym for simple object Access protocol.
2. It is a XML-based protocol for accessing web services.
3. It is a W3C recommendation for communication between applications.
4. It is XML based, so it is platform independent and language independent. In other words, it can be used with java, .NET or PHP language or any platform.

5. Advantages of SOAP Web services:

1. WS Security- It defines its own security known as a WS security.
2. Language and platform independent- SOAP webservices can be written in any programming language and executed in any platform.

6. Disadvantages of SOAP Web services:

1. Slow- SOAP uses XML format that must be parsed to be read. It defines many standards that must be followed while developing the SOAP applications. So it is slow and consumes more bandwidth and resource.
2. WSDL dependent- SOAP uses WSDL and doesn't have any other mechanism to discover the service.

7. SOAP Building blocks:

1. The SOAP specification defines something known as a "SOAP message" which is what is sent to the web service and the client application.

2. The below diagram of SOAP architecture shows the various blocks of a SOAP message

Fig 1 SOAP Message Building Blocks

3. The SOAP message is nothing but a mere XML document which has the below components.
4. An Envelope element that identifies the XML document as a SOAP message-This is the containing part of the SOAP message and is used to encapsulate all the details in the SOAP message. This is the root element in the SOAP message.
5. A Header element that contains header information-The header element can contain information such as authentication credentials which can be used by the calling application.
6. A Simple SOAP service example of a complex type is shown below

Suppose we want to send a structured data type which had a combination of “tutorial name” and “tutorial description”, then we would define the complex type as shown below

The complex type is defined by the element tag <xsd:complexType>. All of the required elements of the structure along with their respective data types are then defined in the complex type collection.

```
<xsd:complexType>
<xsd:sequence>
      <xsd:element name="Tutorial Name" type="string"/>
      <xsd:element        name="Tutorial        Description"
type="string"/>
</xsd:sequence>
</xsd:complexType>
```

6.1 A body element that contains call and response information- This element is what contains the actual data which needs to be sent between the web service and the calling application.

```
<soap:Body>
<GetTutorialInfo>
            <TutorialName>Web Services</TutorialName>
```

```
            <TutorialDescription>All        about        web
services</TutorialDescription>
</GetTutorialInfo>
</soap:Body>
```

7. SOAP Message Structure:

1. SOAP messages are normally auto-generated by the web service when it is called.
2. Whenever a client application calls a method in the web service, the web service will automatically generate a SOAP message which will have the necessary details of the data which will be sent from the web service to the client application.
3. A Simple SOAP message has the following elements-
    1. The Envelope element
    2. The header element and
    3. The body element
    4. The fault element(optional)

8. SOAP Envelope element:

1. The first bit of the building block is the SOAP envelope. The SOAP envelope is used to encapsulate all of the necessary details of the SOAP messages, which are exchanged between the web service and the client application.
2. The SOAP envelope element is used to indicate the beginning and to end of a SOAP message. This enables the client application which calls the web service to know when the SOAP message ends.
3. The following points can be noted on the SOAP envelope element
    1. Every SOAP message needs to have a root envelope element. It is absolutely mandatory for SOAP message to have an envelope element.
    2. Every envelope element needs to have atleast one SOAP body element.
    3. If an envelope element contains a header element, it must contain no more than one, and it must appear as the first child of the envelope before the body element.
    4. The envelope changes when SOAP version change.
    5. A V1.1 complaint SOAP processor generates a fault upon receiving a message containing the v1.2 envelope namespace.
    6. A v1.2-compliant SOAP processor generates a Version Mismatch fault if it receives a message that does not include the v1.2 envelope namespace.

9. The Fault Message:

When a request is made to a SOAP web service, the response returned can be of either 2 forms which are a successful response or an

error response. When a success is generated, the response from the server will always be a SOAP message. But if SOAP faults are generated, they are returned as "HTTP 500" errors.

10. SOAP Communication Model:

1. All communication by SOAP is done via the HTTP protocol. Prior to SOAP, a lot of web services used the standard RPC (Remote Procedure Call) style for communication.
2. SOAP would then use the below communication model

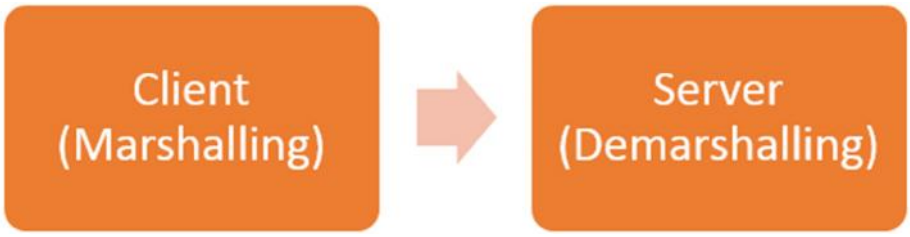


Fig 2 Client Marshalling and Demarshalling

2.1 The client would format the information regarding the procedure call and any arguments into a SOAP message and sends it to the server as part of an HTTP request. This process of encapsulating the data into a SOAP message was known as **Marshalling.**

2.2 The server would then unwrap the message sent by the client, see what the client requested for and then send the appropriate response back to the client as a SOAP message. The practice of unwrapping a request sent by the client is known as **Demarshalling.**

3.2 Building a web services using JAX-WS:

1. JAX-WS tutorial is provides concepts and examples of JAX-WS API. There are two ways to develop JAX-WS example

1.1 Through RPC Style

1.2 Document Style

2. Difference between RPC and Document web services

1. RPC Style

1.1 RPC style web services use method name and parameters to generate XML structure.

1.2 The generated WSDL is difficult to be validated against schema.

1.3 In RPC style, SOAP message is sent as many elements.

1.4 RPC style message is tightly coupled.

1.5 In RPC style, SOAP message keeps the operation name.

1.6 In RPC style, parameters are sent as discrete values.

2. Document Style

1.7 Document style web service can be validated against predefined schema

1.8 In document style, SOAP message is sent as a single document.

1.9 Document style message is loosely coupled

1.10 In document style, SOAP message loose the operation name.

1.11 In document style, parameters are sent in XML format.

3. Creating JAX-WS example is a easy task because it requires no extra configuration settings.

1 JAX-WS API is in built in JDK, So you don't need to load any extra jar file for it. Lets see a simple example of JAX-WS example in RPC style

2 There are created 4 files for hello world JAX-WS example

1.3 Helloworld.java

1.4 Helloworldimpl.java

1.5 Publisher.java

1.6 HelloWorldClient.java

The first 3 files are created for server side and 1 application for client side
JAX-WS Server code

File Helloworld.java

```
1.  package com.javatpoint;
2.  import javax.jws.WebMethod;
3.  import javax.jws.WebService;
4.  import javax.jws.soap.SOAPBinding;
5.  import javax.jws.soap.SOAPBinding.Style;
6.  //Service Endpoint Interface
7.  @WebService
8.  @SOAPBinding(style = Style.RPC)
9.  public interface HelloWorld{
10.    @WebMethod String getHelloWorldAsString(String name);
11. }
```

File HelloWorldImpl.java

```
1.  package com.javatpoint;
2.  import javax.jws.WebService;
3.  //Service Implementation
4.  @WebService(endpointInterface = "com.javatpoint.HelloWorld")
5.  public class HelloWorldImpl implements HelloWorld{
6.    @Override
```

```
7.          public String getHelloWorldAsString(String name) {
8.             return "Hello World JAX-WS " + name;
9.          }
10.     }
```

File Publisher.java

```
1.      package com.javatpoint;
2.      import javax.xml.ws.Endpoint;
3.      //Endpoint publisher
4.      public class HelloWorldPublisher{
5.          public static void main(String[] args) {
6.            Endpoint.publish("http://localhost:7779/ws/hello", new Hello
WorldImpl());
7.             }
8.      }
```

After running the publisher code, we can see the generated WSDL file by visiting the URL

http://localhost:7779/ws/hello?wsdl

JAX-WS Client Code

File HelloWorldClient.java

```
• package com.javatpoint;
• import java.net.URL;
• import javax.xml.namespace.QName;
• import javax.xml.ws.Service;
• public class HelloWorldClient{
•    public static void main(String[] args) throws Exception {
•    URL url = new URL("http://localhost:7779/ws/hello?wsdl");
•
•       //1st argument service URI, refer to wsdl document above
•    //2nd argument is service name, refer to wsdl document above
•
     QName qname = new QName("http://javatpoint.com/", "HelloWorldI
mplService");
•       Service service = Service.create(url, qname);
•       HelloWorld hello = service.getPort(HelloWorld.class);
•
     System.out.println(hello.getHelloWorldAsString("javatpoint rpc"));
•     }
}
```

Output

HelloWorld JAX-WS Javatpoint rpc

2. JAVA MAIL API

1 The JavaMail is an API that is used to compose, write and read electronic messages (emails).

2 The JavaMail API provides protocol-independent and platform-independent framework for sending and receiving mails.

2 The javax.mail and javax.mail.activation packages contains the core classes of javamail API.

3 The javamail facility can be applied to many events. It can be be used at the time of registering the user (sending notification such as thanks for registering), forgot password, sending notifications for important updates etc.

4.1 Mail Protocols:

1. SMTP

1. SMTP is an acronym for Simple Mail Transfer Protocol. It provides a mechanism to deliver the email.
2. We can use Apache James server, Postcast server, cmail server etc. as an SMTP server. But if we purchase the host space, an SMTP server is by default provided by the host provider.
3. For example, my smtp server is mail.javatpoint.com. If we use the SMTP server provided by the host provider, authentication is required for sending and receiving emails.

2 POP:

1. POP is an acronym for Post Office Protocol, also known as POP3. It provides a mechanism to receive the email.
2. It provides support for single mail box for each user. We can use Apache James server, cmail server etc. as an POP server.
3. But if we purchase the host space, an POP server is by default provided by the host provider. For example, the pop server provided by the host provider for my site is mail.javatpoint.com. This protocol is defined in RFC 1939.

3. IMAP

1 IMAP is an acronym for Internet Message Access Protocol. IMAP is an advanced protocol for receiving messages.

2 It provides support for multiple mail box for each user, in addition to, mailbox can be shared by multiple users. It is defined in RFC 2060.

4. MIME:

1 Multiple Internet Mail Extension (MIME) tells the browser what is being sent e.g. attachment, format of the messages etc. It is not known as mail transfer protocol but it is used by your mail program.

5. NNTP and Others:

1 There are many protocols that are provided by third-party providers. Some of them are Network News Transfer Protocol (NNTP), Secure Multipurpose internet mail extensions (S/MIME) etc.

4.2 Components of the Java Mail API

1 The java application uses JavaMail API to compose, send and receive mails. The Java Mail API uses SPI(service provider Interfaces) that provides the intermediatory services to the java application to deal with the different protocols. Let's understand it with the figure given below

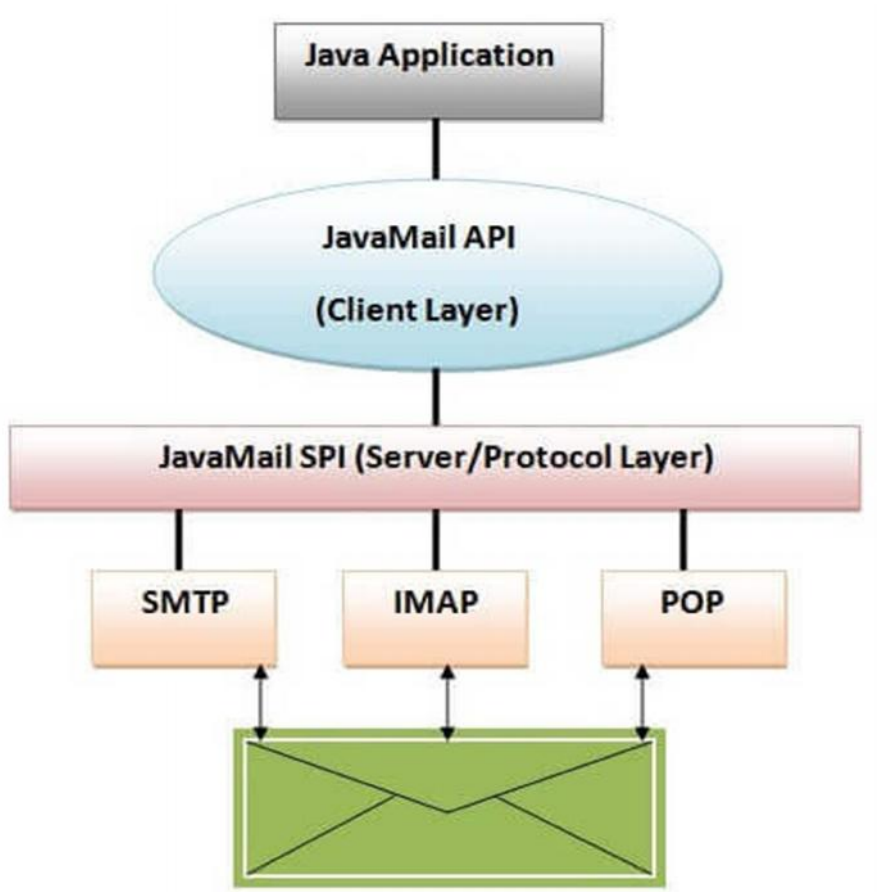


Fig 3 Architecture cum Components of Java Mail API

2 JavaMail API Core Classes:

There are two packages that are used in Java Mail API. Javax.mail and javax.mail.internet package. These packages contains many classes for java mail API They are

- 1 javax.mail.Session class
- javax.mail.Message class
- javax.mail.internet.MimeMessage class
- javax.mail.Address class
- javax.mail.internet.InternetAddress class
- javax.mail.Authenticator class
- javax.mail.PasswordAuthentication class
- javax.mail.Transport class
- javax.mail.Store class
- javax.mail.Folder class etc.

4.3 Starting JavaMail API:

1 There are various ways to send email using JavaMail API. For this purpose, one must have SMTP server that is responsible to send mails.

2 One can use one of the following techniques to get the SMTP server

2.1 Install and use any SMTP server such as Postcast server, Apache James server, Cmail server etc.

2.2 Use the SMTP server provided by the host provider eg My SMTP server is mail.javatpoint.com(or)

2.3 Use the SMTP server provided by other companies egg mail etc.

3 Steps to send email using JavaMail API

3.1 There are following three steps to send mail using Javamail. They are as follows

1 Get the Session object- That stores all the information of host like host name, username, password etc. The javax.mail.Session class provides two methods to get the object of session, Session.getDefaultInstance() method and Session.getInstance() method. We can use any method to get the session object

1.1 Methods of Session class

Sr. No.	Method	Description
1	public static Session getDefaultInstance(Properties props)	returns the default session.
2	public static Session getDefaultInstance(Properties props,Authenticator auth)	returns the default session.
3	public static Session getInstance(Properties props)	returns the new session.
4	public static Session getInstance(Properties props,Authenticator auth)	returns the new session.

2. Compose the message- The javax.mail.Message class provides methods to compose the message. But it is an abstract class so its subclass javax.mail.internet.MimeMessage class is mostly used.

1.2 To create the message we need to pass session object in MimeMessage class constructor. For example MimeMessage message=new MimeMessage(session). Now message object has been created but to store information in this object MimeMessage class provides many methods.

1.3 Commonly used methods of MimeMessage class

Sr.No	Methods	Description
1	public void setFrom(Address address)	is used to set the from header field.
2	publicvoid addRecipient(Message.RecipientType type, Address address)	is used to add the given address to the recipient type.
3	public void addRecipients(Message.RecipientType type, Address[] addresses)	is used to add the given addresses to the recipient type.
4	public void setSubject(String subject)	is used to set the subject header field.
5	public void setText(String textmessage)	is used to set the text as the message content using text/plain MIME type.
6	public void setText(String textmessage)	is used to set the text as the message content using text/plain MIME type.
7	public void setContent(Object msg, String contentType)	is used to set the content as the message content using given MIME type.

2. Send the message- The javax.mail.Transport class provides method to send the message. Commonly used methods of transport class

Sr.No	Method	Description
1	public static void send(Message message)	is used send the message.
2	public static void send(Message message, Address[] address)	is used send the message to the given addresses.

Simple example of sending email in Java

For sending the email using JavaMail API, you need to load two jar files Mail.jar and activation.jar

```
1.  import java.util.*;
2.  import javax.mail.*;
3.  import javax.mail.internet.*;
4.  import javax.activation.*;
5.
6.  public class SendEmail
7.  {
8.   public static void main(String [] args){
9.       String to = "sonoojaiswal1988@gmail.com";//change accordi
         gly
10.      String from = "sonoojaiswal1987@gmail.com";change accordi
         ngly
11.      String host = "localhost";//or IP address
12.
13.     //Get the session object
14.      Properties properties = System.getProperties();
15.      properties.setProperty("mail.smtp.host", host);
16.      Session session = Session.getDefaultInstance(properties);
17.
18.     //compose the message
19.      try{
20.        MimeMessage message = new MimeMessage(session);
21.        message.setFrom(new InternetAddress(from));
22.        message.addRecipient(Message.RecipientType.TO,new Inte
           netAddress(to));
23.        message.setSubject("Ping");
24.        message.setText("Hello, this is example of sending email  ");

25.
26.        // Send message
27.        Transport.send(message);
28.        System.out.println("message sent successfully....");
29.
30.      }catch (MessagingException mex) {mex.printStackTrace();}
31.    }
32. }
```

To run above code we need to load two jar files. There are 4 ways to load the jar file. One of the way is set classpath. Let's see how to run this example

Load the Jar file	**set classpath=mail.jar;activation.jar;.;**
Compile the source file	**javac SendEmail.java**
Run by	**java SendEmail**

5. JNDI:

1. The java Naming and Directory Interface(JNDI) is an application programming interface(API) that provides naming and directory functionality to applications written using the Java programming language.
2. It is defined to be independent of any specific directory service implementation. Thus a variety of directories- new, emerging and already deployed can be accessed in a common way.
3. Architecture

1 The JNDI architecture consists of an API and a service provider interface(SPI). Java applications use the JNDI API to access a variety of naming and directory services. The SPI enables a variety of naming and directory services to be plugged in transparently, thereby allowing the java application using the JNDI API to access their services, as given in following figure.

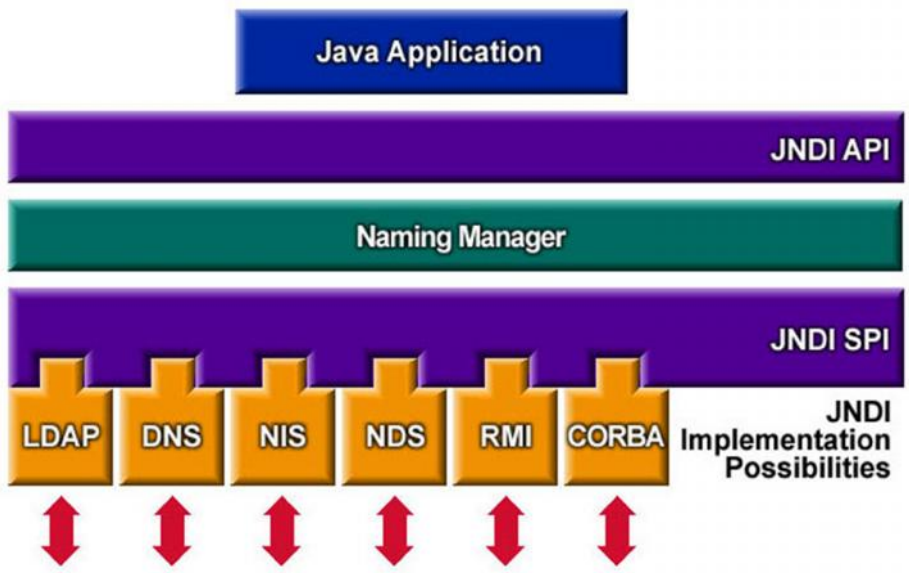


Fig 4 Architecture of JNDI

5. Naming and Directory Service:

1 A client looking for a component/service usually knows its name, but not its physical location.

2 For application, a naming/ directory service is the way to get a reference to a required service(e.g. a JDBC data source, a JMS connection factory, an EJB home interface)

3 A ***naming service*** is an application that contains a set of objects, or references to objects, with corresponding names, Such correspondances are called ***bindings***.

A ***directory service*** allows for the association of attributes to a binding.
Some popular directory implementations:

- Lightweight Directory Access Protocol (LDAP)
- Network Directory Service (NDS)

Network Information Servis Plus (NIS+)
4JNDI (Java Naming Directory interface) provides java clients with the capability to access naming and directory services.
JNDI is subdivided in to the following package
Javax.naming
Javax.naming.directory
Javax.naming.event
Javax.naming.ldap
Javax.naming.spi

4 JNDI configuration could be a quite difficult task – Whenever we use a EJB server, JNDI is started automatically at the same time of the server itself. Such a service is usually already configured for the specific server.

5 Whenever we use an EJB server, JNDI is started automatically at the same time of the server itself. Such a service is usually already configured for the specific server.

6 Also the client applications using JNDI must be configured and this task is up to the programmer / assembler / deployer.

10.2 JNDI and Resources of JNDI:

1. JNDI is the acronym for the Java Naming and Directory Interface API. By making calls to this API, applications locate resources and other program objects.
2. A resource is a program object that provides connections to sytems, such as database servers and messaging systems. (A JDBC resource is sometimes referred to as a data source.)
3. Each resource object is identified by a unique, people-friendly name, called the JNDI name. A resource object and its JNDI name are bound together by the naming and directory service, which is included with the Application Server.
4. To create a new resource, a new name-object binding is entered into the JNDI.
5. Using Custom Services

5.1 A custom resource accesses a local JNDI repository and an external resource accesses an external JNDI repository. Both types of resources

need user-specified factory class elements, JNDI name attributes, etc. In this section, we will discuss how to configure JNDI connection factory resources, for J2EE resources, and how to access these resources.

5.2 Within Application Server, you can create, delete and list resources as well as list-jndi-enttites.

5.3 Creating Custom Resources

1. In the left pane of the Admin Console, open the Sun Java System Application Server instance for the JNDI configuration to be modified.
2. Open the JNDI tab and click Custom Resources. If any custom resources have been created already, they are listed in the right pane. To create a new custom resource, click New. Open the JNDI tab and click New. A page for adding a new custom resource appears.
3. In the JNDI Name field, enter the name to use to access the resource. This name will be registered in the JNDI naming service.
4. In the Resource Type field, enter a fully qualified type definition, as shown in the example above. The Resource Type definition follows the format, xxx.xxx.
5. In the Factory Class field, enter a factory class name for the custom resource to be created. The Factory Class is the user-specified name for the factory class. This class implements the javax.naming.spi.ObjectFactory interface.
6. In the Description field, enter a description for the resource to be creating. This description is a string value and can include a maximum of 250 characters.
7. Mark the Custom Resource Enabled checkbox, to enable the custom resource.
8. Click OK to save your custom resource.

5.4 Editing Custom Resources:

1. In the left pane of the Admin Console, open the Sun Java System Application Server instance for the JNDI configuration to be modified.
2. Open JNDI and select Custom Resources. If any custom resources have been created already, they are listed in the right pane. To edit a custom resource, click on the file name in the right pane.
3. Edit the Resource Type field, the Factory Class field, or the Description field.
4. Mark the Custom Resource Enabled checkbox, to enable the custom resource.
5. Click Save to save the changes to the custom resource.

5.5 Deleting Custom Resource:

1. In the left pane of the Admin Console, open the JNDI tab.
2. Click Custom Resources. If any custom resources have been created already, they are listed in the right pane. To delete a custom resource, click in the box next to the name of the resource to be deleted.
3. Click Delete. The custom resource is deleted.

6. Creating External Resource:

1. In the left pane of the Admin Console, open the Sun Java System Application Server instance for the JNDI configuration to be modified.
2. Open JNDI and select External Resources. If any external resources have been created already, they are listed in the right pane. To create a new external resource, click New.
3. In the JNDI Name field, enter the name that is to be used to access the resource. This name is registered in the JNDI naming service.
4. In the Resource Type field, enter a fully qualified type definition, as shown in the example above. The Resource Type definition follows the format, `xxx.xxx`.
5. In the JNDI Lookup field, enter the JNDI value to look up in the external repository. For example, when creating an external resource to connect to an external repository, to test a bean class, theJNDI Lookup can look like this; `cn=`*testmybean*.
6. In the Factory Class field, enter a JNDI factory class external repository, for example, `com.sun.jndi.ldap`. This class implements the javax.naming.spi.`ObjectFactory` interface.
7. In the Description field, enter a description for the resource to be created. This description is a string value and can include a maximum of 250 characters.
8. Mark the External Resource Enabled checkbox, to enable the external resource.
9. Click OK to save the external resource.
   asadmin command equivalent: `create-jndi-resource` .

7. Editing External Resource:

1. In the left pane of the Admin Console, open the Sun Java System Application Server instance for the JNDI configuration to be modified.
2. Open JNDI and select External Resources. If any external resources have been created already, they are listed in the right pane. To edit an external resource, click on the file name in the right pane.
3. Edit the Resource Type field, the JNDI Lookup field, the Factory Class field, or the Description field.

4. Mark the External Resource Enabled checkbox, to enable the external resource.
5. Click Save to save the changes to the external resource.

8 Deleting External Resource:

To delete an external resource:

1. In the left pane of the Admin Console, open the JNDI tab.
2. Click External Resources. If any external resources have been created already, they are listed in the right pane. To delete an external resource, click the box next to the name of the resource to be deleted.
3. Click Delete. The external resource is deleted.
   asadmin command equivalent delete-jndi-resource.

SUMMARY

In this Unit, overall architecture of webservices, JNDI and SOA is discussed along with API used for mailing services

REFERENCES

[1] The Complete Reference of Java by Herbert Scheildt, 7th Edition published by Indian Edition

QUESTIONS

1. Explain the architecture of JNDI
2. Explain the concept of web services
3. Describe SOAP in your own terms?
4. List down the steps for creation JAX-WS web service
5. Illustrate the Java Mail API methods

*****