1

JAVA SWING

Unit Structure :

1.0 INTRODUCTION

Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

1.1 DIFFERENCE BETWEEN AWT AND SWING

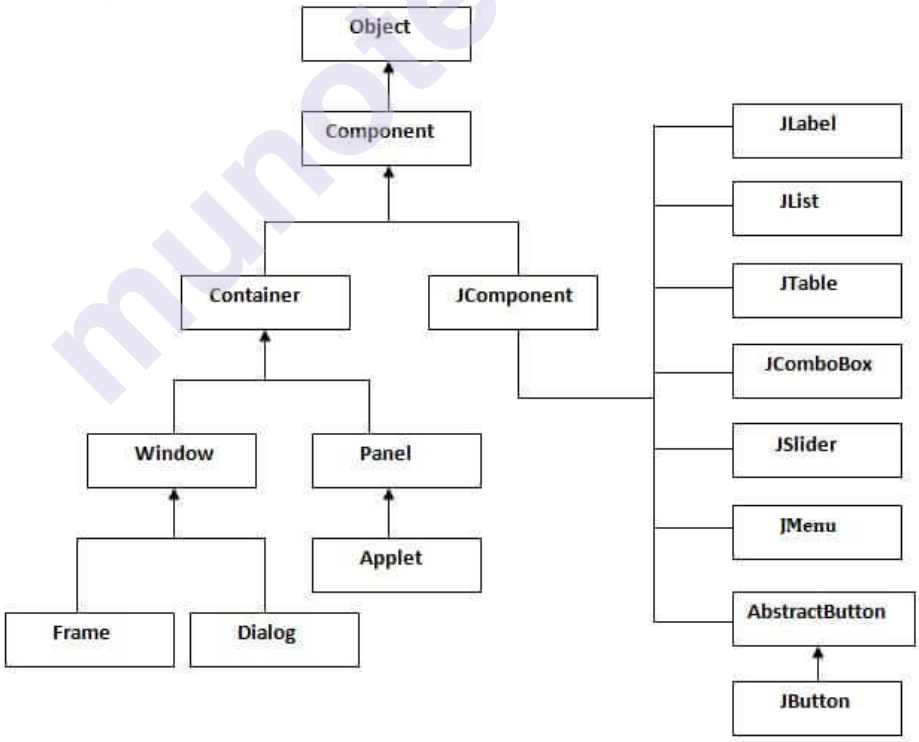
There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.

No.	Java AWT	Java Swing
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



1.2 COMMONLY USED METHODS OF COMPONENT CLASS

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

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

1.2.1. Java Swing Examples

There are two ways to create a frame:

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

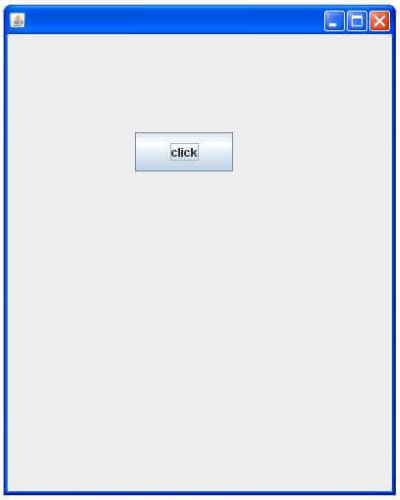
We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

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

File: FirstSwingExample.java

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



Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```
import javax.swing.*;
public class Simple
 {
JFrame f;
Simple()
{
f=new JFrame();//creating instance of JFrame
        JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);
        f.add(b);//adding button in JFrame
        f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
public static void main(String[] args)
{
new Simple();
}
}
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

1.3 SWING COMPONENTS

1.3.1. Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

public class JButton **extends** AbstractButton **implements** Accessible

Commonly used Constructors:

Constructor	**Description**
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

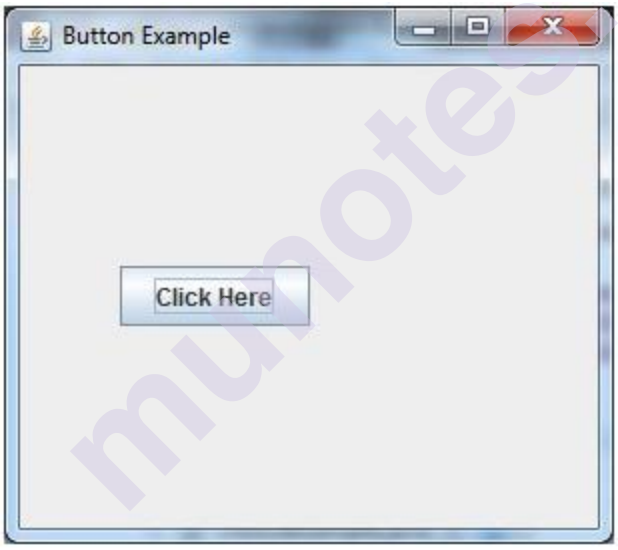
Commonly used Methods of AbstractButton class:

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

Java JButton Example

```
import javax.swing.*;
public class ButtonExample
{
public static void main(String[] args)
 {
        JFrame f=new JFrame("Button Example");
                JButton b=new JButton("Click Here");
                b.setBounds(50,100,95,30);
                f.add(b);
                f.setSize(400,400);
                f.setLayout(null);
                f.setVisible(true);
}
}
```

Output:



1.3.2. Java JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

public class JLabel **extends** JComponent **implements** SwingConstants, A ccessible

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Commonly used Methods:

Methods	Description
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment (int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JLabel Example with ActionListener

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class LabelExample extends Frame implements ActionListener
{
    JTextField tf; JLabel l; JButton b;
    LabelExample(){
    tf=new JTextField();
    tf.setBounds(50,50, 150,20);
    l=new JLabel();
    l.setBounds(50,100, 250,20);
    b=new JButton("Find IP");
    b.setBounds(50,150,95,30);
    b.addActionListener(this);
    add(b);add(tf);add(l);
```

```
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
    {
        try
    {
          String host=tf.getText();
          String ip=java.net.InetAddress.getByName(host).getHostAddress();
            l.setText("IP of "+host+" is: "+ip);
          }
     catch(Exception ex)
    {
System.out.println(ex);
          }
    }
    public static void main(String[] args)
 {
        new LabelExample();
    }
}
```

Output:

1.3.3.Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

public class JTextField **extends** JTextComponent **implements** SwingConstants

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

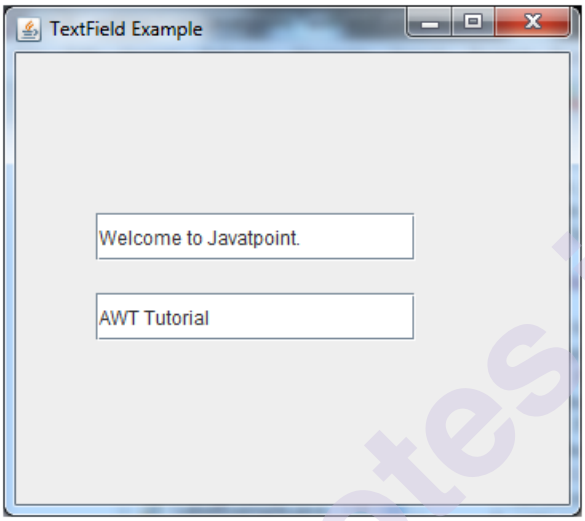
Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

Java JTextField Example

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
        {
   JFrame f= new JFrame("TextField Example");
          JTextField t1,t2;
         t1=new JTextField("Welcome to Java");
         t1.setBounds(50,100, 200,30);
         t2=new JTextField("AWT Tutorial");
```

```
        t2.setBounds(50,150, 200,30);
        f.add(t1);
        f.add(t2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:



1.3.4. Java 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

JTextArea class declaration

Let's see the declaration for javax.swing.JTextArea class.

public class JTextArea **extends** JTextComponent

Commonly used Constructors:

Constructor	Description
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.

Commonly used Methods:

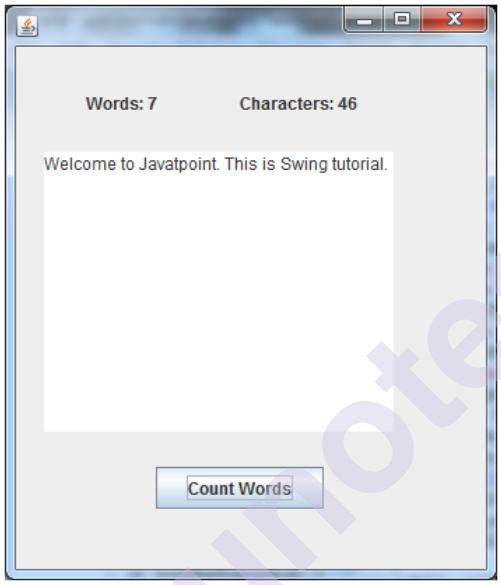
Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

Java JTextArea Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextAreaExample implements ActionListener
{
JLabel l1,l2;
JTextArea area;
JButton b;
TextAreaExample()
{
   JFrame f= new JFrame();
   l1=new JLabel();
   l1.setBounds(50,25,100,30);
   l2=new JLabel();
   l2.setBounds(160,25,100,30);
   area=new JTextArea();
   area.setBounds(20,75,250,200);
   b=new JButton("Count Words");
   b.setBounds(100,300,120,30);
   b.addActionListener(this);
   f.add(l1);
   f.add(l2);
   f.add(area);
   f.add(b);
   f.setSize(450,450);
   f.setLayout(null);
   f.setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
```

```
    String text=area.getText();
    String words[]=text.split("\\s");
    l1.setText("Words: "+words.length);
    l2.setText("Characters: "+text.length());
}
public static void main(String[] args)
{
    new TextAreaExample();
}
}
```

Output:



1.3.5.Java 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.

JPasswordField class declaration

Let's see the declaration for javax.swing.JPasswordField class.

public class JPasswordField **extends** JTextField

Commonly used Constructors:

Constructor	Description
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.

Java JPasswordField Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class PasswordFieldExample
{
   public static void main(String[] args)
{
   JFrame f=new JFrame("Password Field Example");
    final JLabel label = new JLabel();
    label.setBounds(20,150, 200,50);
    final JPasswordField value = new JPasswordField();
    value.setBounds(100,75,100,30);
    JLabel l1=new JLabel("Username:");
      l1.setBounds(20,20, 80,30);
      JLabel l2=new JLabel("Password:");
      l2.setBounds(20,75, 80,30);
      JButton b = new JButton("Login");
      b.setBounds(100,120, 80,30);
      final JTextField text = new JTextField();
      text.setBounds(100,20, 100,30);
            f.add(value); f.add(l1); f.add(label); f.add(l2); f.add(b); f.add(text);
             f.setSize(300,300);
             f.setLayout(null);
             f.setVisible(true);
             b.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent e) {
               String data = "Username " + text.getText();
               data += ", Password: "
               + new String(value.getPassword());
               label.setText(data);
             }
          });
}
}
```

Output:

1.3.6. Java JCheckBox

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

JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

public class JCheckBox **extends** JToggleButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

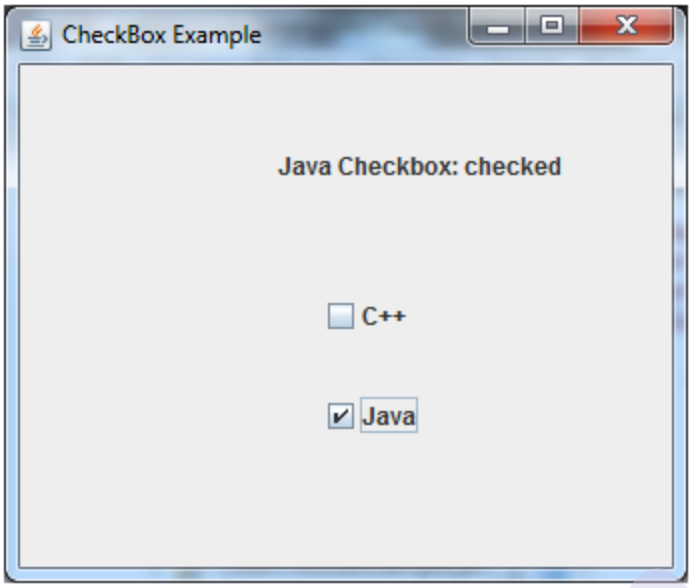
Commonly used Methods:

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

Java JCheckBox Example with ItemListener

```
import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample
{
    CheckBoxExample(){
      JFrame f= new JFrame("CheckBox Example");
      final JLabel label = new JLabel();
      label.setHorizontalAlignment(JLabel.CENTER);
      label.setSize(400,100);
      JCheckBox checkbox1 = new JCheckBox("C++");
      checkbox1.setBounds(150,100, 50,50);
      JCheckBox checkbox2 = new JCheckBox("Java");
      checkbox2.setBounds(150,150, 50,50);
      f.add(checkbox1); f.add(checkbox2); f.add(label);
      checkbox1.addItemListener(new ItemListener() {
          public void itemStateChanged(ItemEvent e) {
            label.setText("C++ Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
          }
        });
      checkbox2.addItemListener(new ItemListener() {
          public void itemStateChanged(ItemEvent e) {
            label.setText("Java Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
          }
        });
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
{
   new CheckBoxExample();
}
}
```

Output:



1.3.7.Java JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

public class JRadioButton **extends** JToggleButton **implements** Accessible

Commonly used Constructors:

Constructor	**Description**
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

Commonly used Methods:

Methods	**Description**
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.

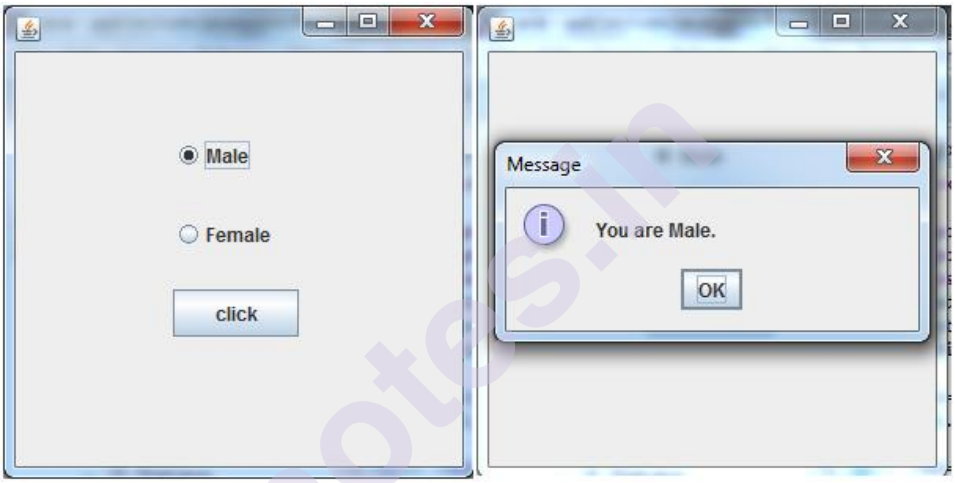
Methods	Description
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JRadioButton Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
class RadioButtonExample extends JFrame implements ActionListener
{
JRadioButton rb1,rb2;
JButton b;
RadioButtonExample(){
rb1=new JRadioButton("Male");
rb1.setBounds(100,50,100,30);
rb2=new JRadioButton("Female");
rb2.setBounds(100,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(rb1);bg.add(rb2);
b=new JButton("click");
b.setBounds(100,150,80,30);
b.addActionListener(this);
add(rb1);add(rb2);add(b);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
if(rb1.isSelected())
{
JOptionPane.showMessageDialog(this,"You are Male.");
}
```

```
if(rb2.isSelected())
{
JOptionPane.showMessageDialog(this,"You are Female.");
}
}
public static void main(String args[])
{
new RadioButtonExample();
}
}
```

Output:



1.3.8. Java JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

JComboBox class declaration

Let's see the declaration for javax.swing.JComboBox class.

public class JComboBox **extends** JComponent **implements** ItemSelectabl e, ListDataListener, ActionListener, Accessible

Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Commonly used Methods:

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

Java JComboBox Example

```
import javax.swing.*;
public class ComboBoxExample
{
JFrame f;
ComboBoxExample()
{
   f=new JFrame("ComboBox Example");
   String country[]={"India","Aus","U.S.A","England","Newzealand"};

   JComboBox cb=new JComboBox(country);
   cb.setBounds(50, 50,90,20);
   f.add(cb);
   f.setLayout(null);
   f.setSize(400,500);
   f.setVisible(true);
}
public static void main(String[] args)
{
   new ComboBoxExample();
}
}
```

Output:

1.3.9.Java JList

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

JList class declaration

Let's see the declaration for javax.swing.JList class.

public class JList **extends** JComponent **implements** Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Commonly used Methods:

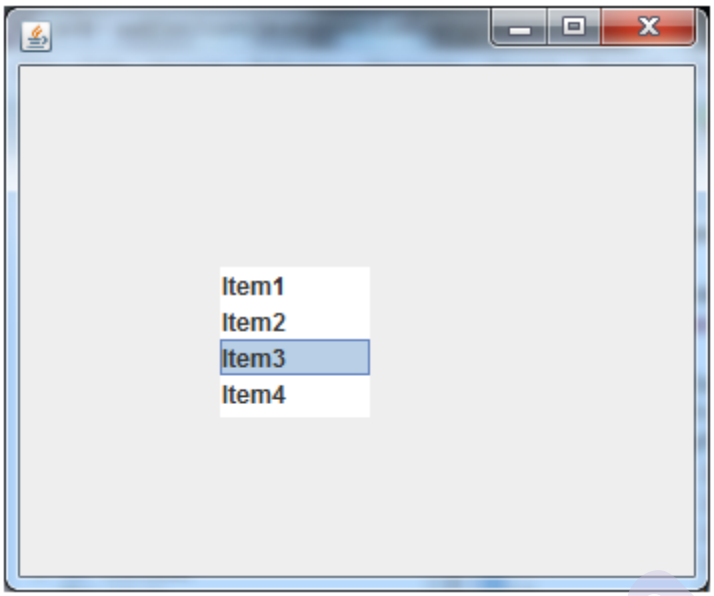
Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.

Methods	Description
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

Java JList Example

```
import javax.swing.*;
public class ListExample
{
    ListExample(){
       JFrame f= new JFrame();
       DefaultListModel<String> l1 = new DefaultListModel<>();
         l1.addElement("Item1");
         l1.addElement("Item2");
         l1.addElement("Item3");
         l1.addElement("Item4");
         JList<String> list = new JList<>(l1);
         list.setBounds(100,100, 75,75);
         f.add(list);
         f.setSize(400,400);
         f.setLayout(null);
         f.setVisible(true);
    }
public static void main(String args[])
   {
  new ListExample();
   }
}
```

Output:



1.4 SUMMARY

In this chapter we learn about the swing to develop the graphical use interface and difference between AWT and Swing. Also we learn about the Swing component like Jtextfield, Jtextarea, Jpasswordfield ,JLabel, etc…

1.5 QUESTIONS

1. Write a short note on swing.
2. What is the difference between AWT and Swing.
3. Explain JTextarea with suitable example.
4. Explain JRadioButton with suitable example.
5. Explain JList with suitable example.
6. Explain JPasswordField with suitable example.

❄❄❄❄❄❄❄

2

JDBC

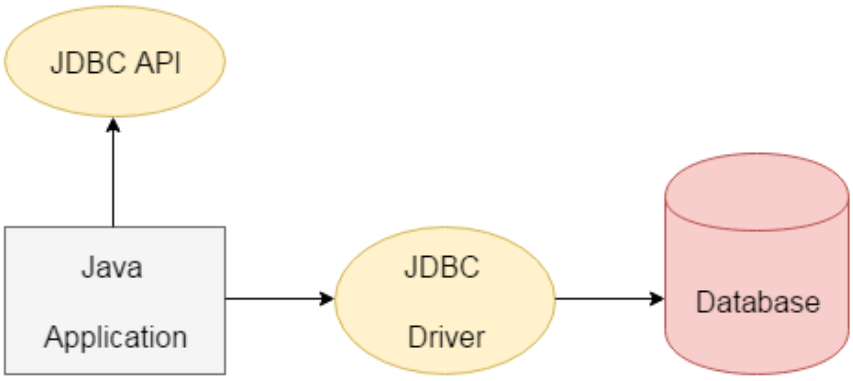
Unit Structure :

2.1 JDBC

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular interfaces of JDBC API are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

2.1.1. Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

What is API?

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

2.2 JDBC DRIVER

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

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

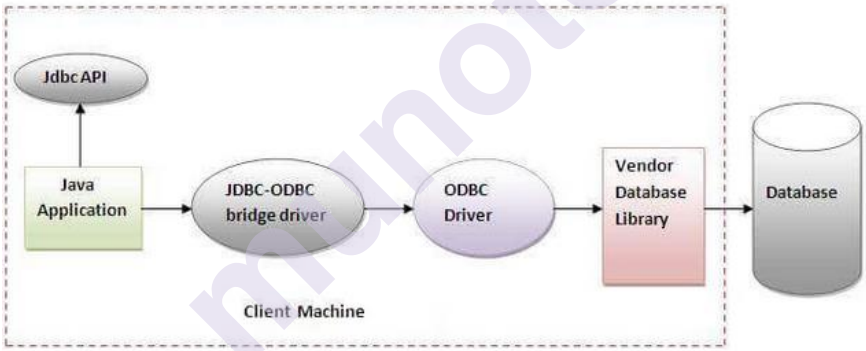


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

Easy to use.

Can be easily connected to any database.

Disadvantages:

Performance degraded because JDBC method call is converted into the ODBC function calls.

The ODBC driver needs to be installed on the client machine.

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

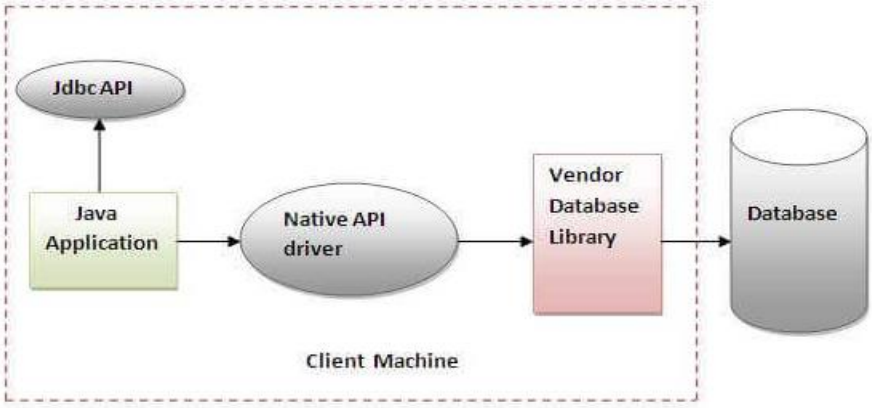


Figure- Native API Driver

Advantage:

Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

The Native driver needs to be installed on the each client machine.

The Vendor client library needs to be installed on client machine.

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

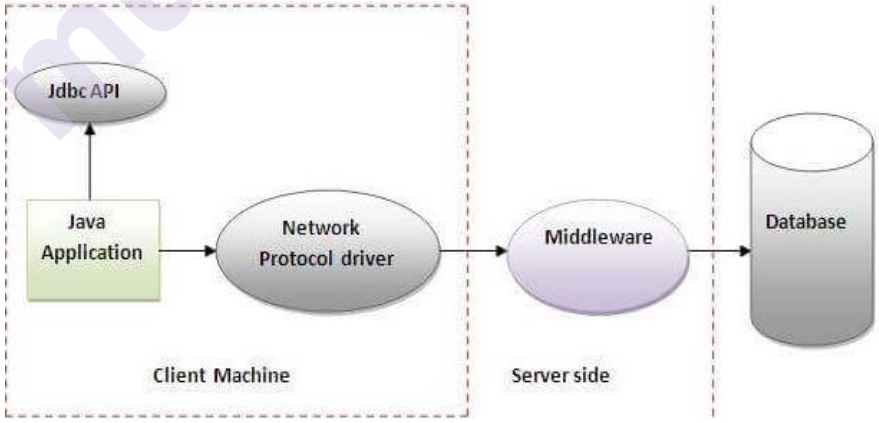


Figure- Network Protocol Driver

Advantage:

No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

Network support is required on client machine.

Requires database-specific coding to be done in the middle tier.

Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) **Thin driver:** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

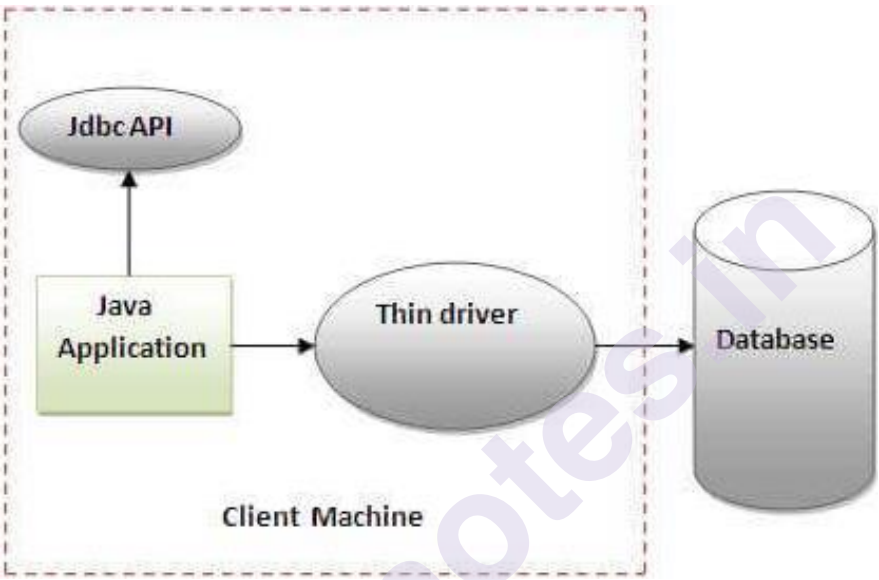


Figure- Thin Driver

Advantage:

Better performance than all other drivers.

No software is required at client side or server side.

Disadvantage:

Drivers depend on the Database.

Java Database Connectivity with 5 Steps

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

1. Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

1) **Register the driver class:** The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

   Syntax of forName() method

   public static void forName(String className)throws ClassNotFound Exception

   Example to register the OracleDriver class

   Here, Java program is loading oracle driver to esteblish database connection.

   Class.forName("oracle.jdbc.driver.OracleDriver");

2) **Create the connection object:** The **getConnection()** method of DriverManager class is used to establish connection with the database.

   Syntax of getConnection() method

   1) public static Connection getConnection(String url)throws SQLEx ception

   2) public static Connection getConnection(String url,String name,Str ing password)

   throws SQLException

   Example to establish connection with the Oracle database

   Connection con=DriverManager.getConnection(

   "jdbc:oracle:thin:@localhost:1521:xe","system","password");

3) **Create the Statement object :** The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

   Syntax of createStatement() method

   public Statement createStatement()throws SQLException

   Example to create the statement object

   Statement stmt=con.createStatement();

4) **Execute the query :** The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

   Syntax of executeQuery() method

   public ResultSet executeQuery(String sql)throws SQLException

Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

5) **Close the connection object :** By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

public void close()throws SQLException

Example to close connection

con.close();

2.3 JAVA DATABASE CONNECTIVITY WITH ORACLE

To connect java application with the oracle database, we need to follow 5 following steps. In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

Driver class: The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.

Connection URL: The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.

Username: The default username for the oracle database is **system**.

Password: It is the password given by the user at the time of installing the oracle database.

Create a Table

Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

```
create table emp(id number(10),name varchar2(40),age number(3));
```

2.3.1. Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from **emp** table. Here, **system** and **oracle** are the username and password of the Oracle database.

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
```

```
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
//step5 close the connection object
con.close();
}
catch(Exception e){ System.out.println(e);}
}
}
```

The above example will fetch all the records of emp table.

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

Two ways to load the jar file:

paste the ojdbc14.jar file in jre/lib/ext folder

set classpath

1) **paste the ojdbc14.jar file in JRE/lib/ext folder:** Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here.

2) **set classpath:** There are two ways to set the classpath:
   - temporary
   - permanent

How to set the temporary classpath:

Firstly, search the ojdbc14.jar file then open command prompt and write:

```
C:>set classpath=c:\folder\ojdbc14.jar;.;
```

How to set the permanent classpath:

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;.; as C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

2.4 DRIVERMANAGER CLASS

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Useful methods of DriverManager class

Method	Description
1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url):	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):	is used to establish the connection with the specified url, username and password.

2.4.1. Connection interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

By default, connection commits the changes after executing queries.

Commonly used methods of Connection interface:

1) **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.

2) **public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) **public void setAutoCommit(boolean status):** is used to set the commit status.By default it is true.

4) **public void commit():** saves the changes made since the previous commit/rollback permanent.

5) **public void rollback():** Drops all changes made since the previous commit/rollback.

6) public void close(): **closes the connection and Releases a JDBC resources immediately.**

2.5 STATEMENT INTERFACE

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

1) **public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.

2) **public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) **public boolean execute(String sql):** is used to execute queries that may return multiple results.

4) **public int[] executeBatch():** is used to execute batch of commands

2.5.1. Example of Statement interface

Let's see the simple example of Statement interface to insert, update and delete the record.

```
import java.sql.*;
class FetchRecord{
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");
Statement stmt=con.createStatement();
//stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
//int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");
int result=stmt.executeUpdate("delete from emp765 where id=33");
System.out.println(result+" records affected");
con.close();
}
}
```

2.6 RESULTSET INTERFACE

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

By default, ResultSet object can be moved forward only and it is not updatable.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatment(int,int) method as well as we can make this object as updatable by:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSEN
SITIVE,
                    ResultSet.CONCUR_UPDATABLE);
```

Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

2.6.1. Example of Scrollable ResultSet

Let's see the simple example of ResultSet interface to retrieve the data of 3rd row.

```
import java.sql.*;
class FetchRecord{
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");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");
  //getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
  con.close();
}
}
```

2.7 PREPAREDSTATEMENT INTERFACE

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

```
String sql="insert into emp values(?,?,?)";
```

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

```
public PreparedStatement prepareStatement(String query)throws SQLException{}
```

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

2.7.1 Example of PreparedStatement interface that inserts the record

First of all create table as given below:

```
create table emp(id number(10),name varchar2(50));
```

Now insert records in this table by the code given below:

```
import java.sql.*;
class InsertPrepared{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localh
ost:1521:xe","system","oracle");

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?
)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");

int i=stmt.executeUpdate();
System.out.println(i+" records inserted");

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

2.7.2 Example of PreparedStatement interface that updates the record

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? w
here id=?");
stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e.
name
stmt.setInt(2,101);
 int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

2.7.3 Example of PreparedStatement interface that deletes the record

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id
=?");
stmt.setInt(1,101);
 int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
```

2.7.4 Example of PreparedStatement interface that retrieve the records of a table

```
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

2.7.5. Example of PreparedStatement to insert records until user press n

```
import java.sql.*;
import java.io.*;
class RS{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localh
ost:1521:xe","system","oracle");
 PreparedStatement ps=con.prepareStatement("insert into emp130 values(
?,?,?)");
BufferedReader br=new BufferedReader(new InputStreamReader(System.
in));
do{
System.out.println("enter id:");
int id=Integer.parseInt(br.readLine());
System.out.println("enter name:");
String name=br.readLine();
System.out.println("enter salary:");
float salary=Float.parseFloat(br.readLine());
```

```
 ps.setInt(1,id);
ps.setString(2,name);
ps.setFloat(3,salary);
int i=ps.executeUpdate();
System.out.println(i+" records affected");

System.out.println("Do you want to continue: y/n");
String s=br.readLine();
if(s.startsWith("n")){
break;
}
}while(true);
con.close();
}
}
```

2.8 SUMMARY

In this chapter we learn the concept of Data Base concept in java using jdbc and its different types of Drivers and how to connect the database like oracle, SQL, etc.. with java program. Also we will discuss about the interfaces like statement, resultset, preparedstatement etc..

2.9 QUESTIONS

1. Write a short note on JDBC.
2. Why Should We Use JDBC.
3. Explain Native-API driver.
4. Write a short note on Resultset.
5. Explain PreparedStatement in detail.
6. Explain JDBC APT components.

❄❄❄❄❄❄❄

3

SERVLETS

Unit Structure:

3.0 INTRODUCTION TO JAVA SERVLETS

Today we all are aware of the need of creating dynamic web pages i.e the ones which have the capability to change the site contents according to the time or are able to generate the contents according to the request received by the client. If you like coding in Java, then you know that using Java there also exists a way to generate dynamic web pages and that way is Java Servlet.

3.1 WEB APPLICATION ARCHITECTURE IN JAVA

Web application architecture is a mechanism that gives us a clarification that how the connection is established between the client and the server. It determines how the components in an application communicate with each other. It doesn't matter what's is the size and the complexity level of the application is, they all follow the same principle only the details may differ.

In technical terms, when a user makes a request on a website, various components of the applications, user interfaces, middleware systems, databases, servers, and the browser interact with each other. Web Application Architecture is a framework that ties up this relation together and maintains the interaction between these components.

Http protocol and http method

For HTTP/1.1, the set of common methods are defined below. This set can be expanded based on the requirements. The name of these methods is case sensitive, and they must be used in uppercase.

i) **GET :** This method retrieves information from the given server using a given URI. GET request can retrieve the data. It can not apply other effects on the data.

ii) **HEAD :** This method is the same as the GET method. It is used to transfer the status line and header section only.

iii) **POST :** The POST request sends the data to the server. For example, file upload, customer information, etc. using the HTML forms.

iv) **PUT :** The PUT method is used to replace all the current representations of the target resource with the uploaded content.

v) **DELETE :** The DELETE method is used to remove all the current representations of the target resource, which is given by URI.

vi) **CONNECT :** This method establishes a tunnel to the server, which is identified by a given URI.

vii) **OPTIONS :** This method describes the options of communication for the target resource.

3.2 WEB SERVER AND WEB CONTAINER

Web Server

A web server can be characterized as software that receives HTTP requests, processes them, and sends back responses.

A web server is a software program that, as its name implies, serves websites to users. It does this by responding to HTTP requests from the user's computer. The response includes HTML content that is sent over the internet and displayed in the user's browser.

Examples of web servers include Apache, Nginx, Microsoft Internet Information Server (IIS). Web Container

A web container, on the other hand, is an application that includes a web server as well as additional components like a servlet container, Enterprise JavaBean (EJB) container, and so forth.

Examples of web containers include Tomcat, Glassfish, JBoss Application Server, or Wildfly.

The benefit of using a web container is that there are fewer applications to maintain and configure. For instance, if your application requires an EJB, you can use the JBoss Application Server instead of Tomcat. If you are using the Spring framework, you can choose between Spring Source to Server or JBoss Application Server to host your Spring apps.

Servlet Interface

Servlet interface provides common behavior to all the servlets. Servlet interface defines methods that all servlets must implement.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

- Genericservlet
- GenericServlet class

- GenericServlet class implements Servlet, ServletConfig and Serializable interfaces. It provides the implementation of all the methods of these interfaces except the service method.
- GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

3.3 METHODS OF GENERICSERVLET CLASS

There are many methods in GenericServlet class. They are as follows:

1. Public void init(ServletConfig config) is used to initialize the servlet.
2. Public abstract void service(ServletRequest request, ServletResponse response) provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. Public void destroy() is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. Public ServletConfig getServletConfig() returns the object of ServletConfig. Public String getServletInfo() returns information about servlet such as writer, copyright, version etc.
5. Public void init() it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
6. Public ServletContext getServletContext() returns the object of ServletContext.
7. Public String getInitParameter(String name) returns the parameter value for the given parameter name.
8. Public Enumeration getInitParameterNames() returns all the parameters defined in the web.xml file.
9. Public String getServletName() returns the name of the servlet object.
10. Public void log(String msg) writes the given message in the servlet log file.
11. Public void log(String msg,Throwable t) writes the explanatory message in the servlet log file and a stack trace.

HttpServlet

Public abstract class HttpServlet extends GenericServlet. Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, usually one of these: doGet , if the servlet supports HTTP GET requests. doPost , for HTTP POST requests.

Servlet Life Cycle

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1) **Servlet class is loaded :** The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) **Servlet instance is created :** The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) **Init method is invoked :** The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

   Public void init(ServletConfig config) throws ServletException

4) **Service method is invoked :** The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

   Public void service(ServletRequest request, ServletResponse response)

   Throws ServletException, IOException

5) Destroy method is invoked : The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

   Public void destroy()

3.4 SERVLETCONFIG

javax.servlet.ServletConfig is an interface as a part of servlet API. For every Servlet class in our application, the web container will create one ServletConfig object and the web container will pass this object as an

argument to the public void init(ServletConfig config) method of our Servlet class object. Some of the important points on ServletConfig are:

ServletConfig is an object containing some initial parameters or configuration information created by Servlet Container and passed to the servlet during initialization.

ServletConfig is for a particular servlet, which means one should store servlet-specific information in web.xml and retrieve them using this object.

ServletContext

Javax.servlet.ServletConfig is an interface as a part of servlet API. For every Servlet class in our application, the web container will create one ServletConfig object and the web container will pass this object as an argument to the public void init(ServletConfig config) method of our Servlet class object. Some of the important points on ServletConfig are:

ServletConfig is an object containing some initial parameters or configuration information created by Servlet Container and passed to the servlet during initialization.

ServletConfig is for a particular servlet, which means one should store servlet-specific information in web.xml and retrieve them using this object.

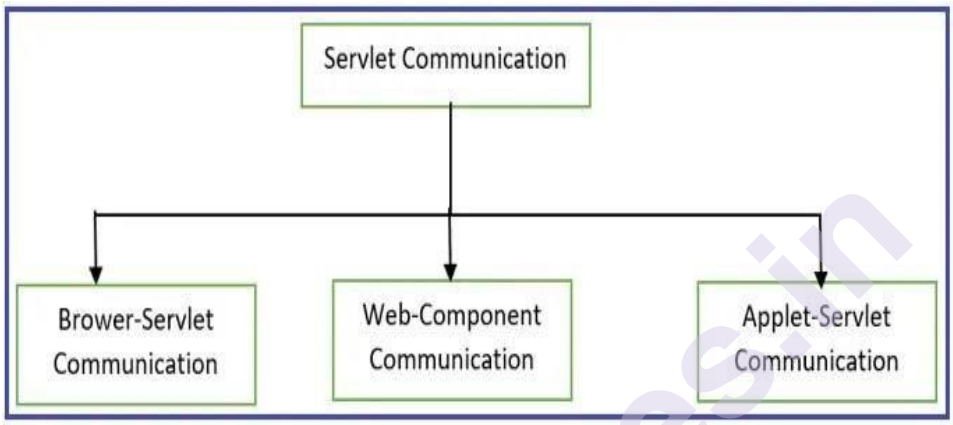
Differentiate between ServletContext and ServletConfig.

Sr. No.	ServletConfig	ServletContext
1	ServletConfig is one per Servlet	ServletContext is one per web application
2	It can be used to pass the deployment time parameters to the servlet using during the servlet initialization like database name, file name, etc	It can be used to access the Web application parameters configured in the deployment descriptor file (WEB.xml)
3	It can be used to access the ServletContext object.	It can be used for the inter application communication between servlets, JSPs and other components of a web application.
4	Can access the initialization parameters for a servlet instance.	Can be used to access the server information about the container and the version of the API it supports.

Servlet Communication

In general, web application deployment is not at all suggestible to provide the complete application logic within a single web resource, it is suggestible to distribute the complete application logic over multiple web resources.

In the above context, to execute the application we must require communication between all the web resources, for this, we have to use Servlet Communication. In the web application, we are able to provide servlet communication in the following three ways:



Browser-Servlet Communication

In web applications, we will use a browser as a client at the client machine, from the browser we will send a request to a servlet available at the server, where the servlet will be executed and generate some response to the client browser.

In the above process, we have provided communication between the client browser and servlet. So that sending a normal request from the client to the server and getting a normal response from the server to the client is an example of Browser-Servlet Communication.

3.5 WEB COMPONENT COMMUNICATION

The process of providing communication between more than one web component available at the server machine is called Web Component Communication.

In general, web-component communication is available in between Servlet-Servlet, Servlet-Jsp, Servlet-HTML, Jsp-Jsp, Jsp-Servlet, Jsp-HTML, and so on. In web applications, we are able to achieve web-component communication in the following 2 ways:

i. Include Mechanism

ii. Forward Mechanism

Applet Servlet Communication

HTML exhibits high performance by taking less time to load in the browser. However, when we use HTML page for important user details, by default, all the parameters that are passed appended in the URL. This compromises with the security. On the other hand, applet takes more time to load but there is no problem with Java security. This is an advantage of this technique.

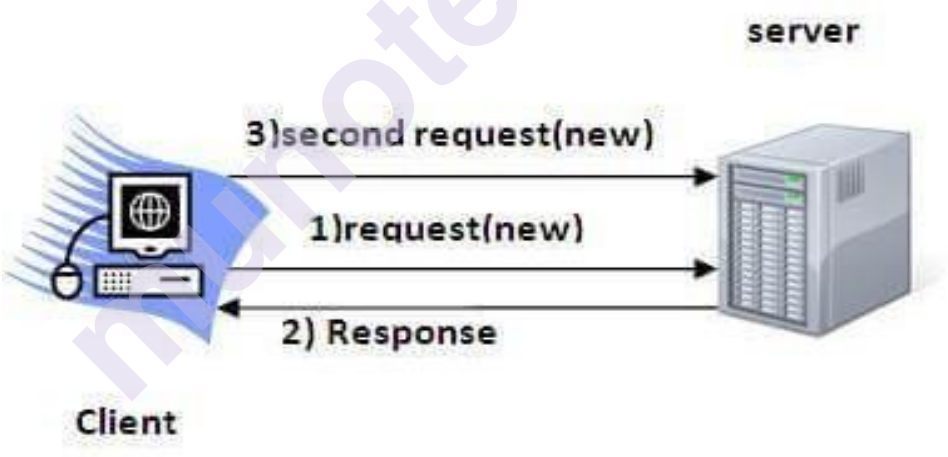
Applet is a compiled Java class that can be sent over the network. Applets are an alternative to HTML form page for developing websites. HTML form gives good performance, takes less time to load but has poor security. Whereas, Applets give poor performance, take time to load but have good security.

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

To recognize the user, it is used to recognize the particular user.

Your session is set to 20 minutes in web.xml.

Session Tracking employs Four Different techniques

A. Cookies

B. Hidden Form Field

C. URL Rewriting

D. HttpSession

A. **Cookies :** Cookies are little pieces of data delivered by the web server in the response header and kept by the browser. Each web client can be assigned a unique session ID by a web server. Cookies are used to keep the session going. Cookies can be turned off by the client.

B. **Hidden Form Field :** The information is inserted into the web pages via the hidden form field, which is then transferred to the server. These fields are hidden from the user's view.

Illustration:

< input type= 'hidden' name='session' value='12345'>

C. **URL Rewriting :** With each request and return, append some more data via URL as request parameters. URL rewriting is a better technique to keep session management and browser operations in sync.

D. **HttpSession :** A user session is represented by the HttpSession object. A session is established between an HTTP client and an HTTP server using the HttpSession interface. A user session is a collection of data about a user that spans many HTTP requests.

3.6 INTRODUCTION TO JSP

JSP (JavaServer Pages) is a technology that enables developers to create dynamic, data-driven web pages using Java. JSP is a server-side technology that generates HTML, XML, or other types of content dynamically based on data that is retrieved from a database or other sources.

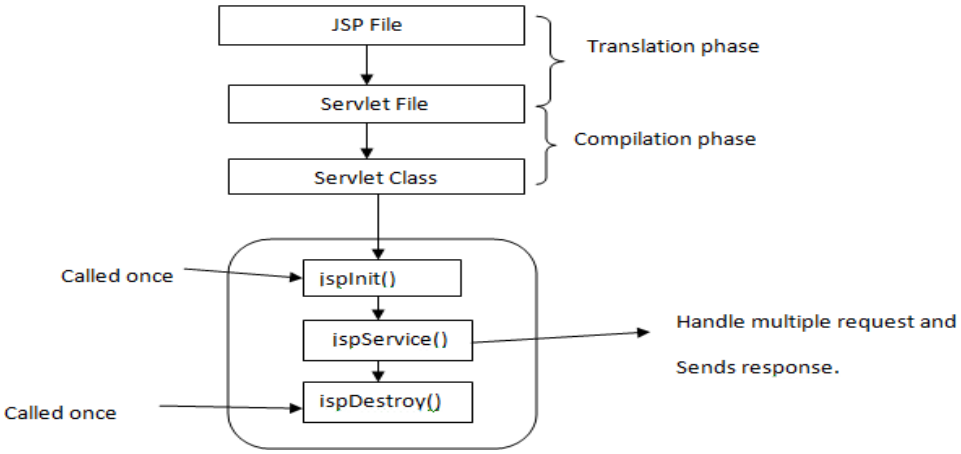
JSP pages are a combination of HTML or XML markup and Java code that is embedded in special tags. The Java code is executed on the server-side and generates dynamic content that is sent to the client browser as HTML or other types of content.

JSP pages can be used to implement a wide range of web applications, from simple forms and pages to complex enterprise applications that require integration with databases, web services, and other technologies.

Life cycle of JSP

A Java Server Page life cycle is defined as the process that started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.

JSP Life Cycle



Translation of JSP page to Servlet
Compilation of JSP page(Compilation of JSP into test.java)
Classloading (test.java to test.class)
Instantiation(Object of the generated Servlet is created)
Initialization(jspInit() method is invoked by the container)
Request processing(_jspService()is invoked by the container)
JSP Cleanup (jspDestroy() method is invoked by the container)

Translation of JSP page to Servlet :

This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP. Here test.jsp file is translated to test.java.

Compilation of JSP page :

Here the generated java servlet file (test.java) is compiled to a class file (test.class).

Classloading :

Servlet class which has been loaded from the JSP source is now loaded into the container.

Instantiation :

Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.

Initialization :

jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

Request processing :

jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.

JSP Cleanup :

In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.

Jsp implicit object scope

Implicit object request has the 'request' scope. '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.

JSP directives

In JSP (JavaServer Pages), directives are special instructions that provide information to the JSP container or web server about how to process the JSP file during translation and execution. Directives are not executed as part of the response to the client's request, but they are used to control the translation and execution of the JSP page.

There are three types of JSP directives:

Page Directive: The page directive is used to define the attributes of the JSP page, such as error handling, scripting language, session management, and buffer size. It is declared at the beginning of the JSP file using the <%@ %> syntax.

Include Directive: The include directive is used to include a file in the current JSP page. It is declared using the <%@ include %> syntax.

Taglib Directive: The taglib directive is used to specify the location of custom tag libraries that are used in the JSP page. It is declared using the <%@ taglib %> syntax.

Directives provide a powerful mechanism to control the behavior of the JSP container and the JSP page during translation and execution. They enable JSP developers to specify various settings and configurations that can affect the performance and behavior of the JSP application.

JSP Scripting elements

JSP (JavaServer Pages) scripting elements are used to include Java code in a JSP page. There are three types of JSP scripting elements:

Expression: The expression element is used to include a Java expression in the JSP page that is evaluated at runtime and the result is included in the response. It is declared using the <%=%> syntax. For example:

The current time is <%= new java.util.Date() %>

In this example, the expression <%= new java.util.Date() %> returns the current time and is included in the response.

Declaration: The declaration element is used to declare variables and methods that can be used in the JSP page. It is declared using the <%! %> syntax. For example:

<%! int count = 0; %>

In this example, a variable named count is declared and initialized to 0.

Scriptlet: The scriptlet element is used to include Java code in the JSP page that is executed at runtime. It is declared using the <% %> syntax. For example:

<% if (count > 0) { %> The count is <%= count %> <% } %>

In this example, the if statement is used to conditionally include a paragraph tag with the value of the count variable in the response.

JSP scripting elements provide a flexible and powerful mechanism to include Java code in JSP pages, which enables developers to dynamically generate content and interact with databases and other resources. However, it is important to use scripting elements judiciously and maintain the separation of concerns between the presentation and business logic layers of the application.

JSP Action Tags

JSP (JavaServer Pages) action tags are special tags that are used to perform specific actions in a JSP page, such as forwarding to another resource, including content from another resource, setting request parameters, and defining custom tag libraries.

There are several JSP action tags available, including:

<jsp:forward>: The <jsp:forward> tag is used to forward the request from the current JSP page to another resource, such as another JSP page or a servlet. It is typically used when a request needs to be processed by multiple resources.

<jsp:include>: The <jsp:include> tag is used to include content from another resource, such as another JSP page or a static HTML file, in the current JSP page. It is typically used to reuse common content across multiple pages.

<jsp:param>: The <jsp:param> tag is used to set request parameters that are passed to the included or forwarded resource. It is typically used to pass data between different components of the application.

<jsp:useBean>: The <jsp:useBean> tag is used to instantiate and initialize a JavaBean component that can be used in the JSP page. It is typically used to encapsulate complex business logic in a separate component.

<jsp:setProperty> and <jsp:getProperty>: The <jsp:setProperty> and

<jsp:getProperty> tags are used to set and get properties of a JavaBean component that is used in the JSP page. They are typically used to access and modify data stored in the JavaBean.

Custom Tag Libraries: JSP also allows you to define custom tag libraries, which are sets of custom tags that can be used in JSP pages. Custom tag libraries are typically used to encapsulate complex functionality and provide a higher-level abstraction for common tasks.

JSP action tags provide a powerful mechanism for controlling the behavior of JSP pages and interacting with other components of the application. They enable developers to create reusable components, encapsulate business logic, and manage the flow of requests and responses between different resources.

Custom Tags in JSP

Custom tags are a powerful feature of JSP (JavaServer Pages) that enable developers to define their own tags that can be used in JSP pages. Custom tags can encapsulate complex functionality and provide a higher-level abstraction for common tasks, which can improve the clarity, maintainability, and reusability of JSP pages.

Custom tags can be defined using the JSP Standard Tag Library (JSTL) or by creating a custom tag library. To create a custom tag library, you define a set of tag handlers that implement the behavior of the custom tags. A tag handler is a Java class that extends a JSP tag handler base class and overrides its methods to implement the behavior of the custom tag. The JSP container invokes the methods of the tag handler during the processing of the JSP page.

There are two types of custom tags:

Simple Custom Tags: Simple custom tags are implemented using a single tag handler class that implements the behavior of the custom tag. Simple custom tags are used to encapsulate simple functionality, such as formatting or validation.

Composite Custom Tags: Composite custom tags are implemented using multiple tag handler classes that work together to implement the behavior of the custom tag. Composite custom tags are used to encapsulate complex functionality, such as a data grid or a chart.

To use a custom tag in a JSP page, you first declare the custom tag library using the <%@taglib%> directive, which specifies the URI of the custom tag library and an alias for the library. You can then use the custom tags in the JSP page using the tag name and any attributes that are defined for the tag.

Custom tags provide a powerful mechanism for creating reusable components in JSP pages, which can improve the maintainability, scalability, and performance of web applications. However, it is important to use custom tags judiciously and maintain the separation of concerns between the presentation and business logic layers of the application.

UNIT END EXERCISES

1. What are servlets? Explain the request/response paradigm of servlet
2. Explain different phases of Servlet Life Cycle.
3. Write a short note on GenericServlet Class.
4. Differentiate between ServletConfig and ServletContext.
5. Differentiate between JSP and Servlet
6. Differentiate between JSP Include directives and JSP Include action.
7. What is page directives? Explain page directives and its attributes.

❄❄❄❄❄❄❄

4

JAVA BEANS

Unit Structure:

4.1. OBJECTIVE

- To understand the concept of java bean.
- To understand the properties of java bean with example.

4.2. INTRODUCTION

Java Bean:-

- A Java Bean is a software component that has been designed to be reusable in a variety of different environments.
- There is no restriction on the capability of a Bean. It may perform a simple function, such as checking the spelling of a document, or a complex function, such as forecasting the performance of a stock portfolio.
- A Bean may be visible to an end user. One example of this is a button on a graphical user interface. A Bean may also be invisible to a user. Software to decode a stream of multimedia information in real time.

Advantages of Java Beans:-

- Since java bean is a reusable component, it implies Java's "write-once, run-anywhere" paradigm. It is used in distributed environments.
- The properties, events, and methods of a Bean that are exposed to an application builder tool can be controlled.
- A Bean may be designed to operate correctly in different environments.
- Auxiliary software can be provided to help a person configure a Bean. This software is only needed when the design-time parameters for that

component are being set. It does not need to be included in the run-time environment.

- The configuration settings of a Bean can be saved in persistent storage and restored at a later time.
- A Bean may register to receive events from other objects and can generate events that are sent to other objects.

JAR Files:-

- A JAR file is a Java archive file, which allows you to efficiently deploy a set of classes and their associated resources.
- For example, A multimedia application may uses various sound and image files. A set of Beans can control how and when this information is presented. All of these pieces can be placed into one JAR file.
- The elements in a JAR file are compressed, which makes downloading a JAR file much faster than separately downloading several uncompressed files.
- Digital signatures may also be associated with the individual elements in a JAR file, which allows a consumer to be sure that these elements were produced by a specific organization.
- To generate jar file, the syntax is:- **j**ar options files

Creating a JAR File

- The following command creates a JAR file named Xyz.jar that contains all of the .class and .gif files in the current directory:

  jar cf Xyz.jar *.class *.gif

  Jar Option are:-

  c:- A new archive is to be created.

  C:- Change directories during command execution.

  f:- The first element in the file list is the name of the archive that is to be created or accessed.

  m:-The second element in the file list is the name of the external manifest file.

  M:-Manifest file not created.

  t:-The archive contents should be tabulated.

  u:-Update existing JAR file.

  v:-Verbose output should be provided by the utility as it executes.

  x:-Files are to be extracted from the archive.

  0:-Do not use compression.

- If a manifest file such as **Yxz.mf** is available, it can be used with the following command:

  jar cfm Xyz.jar Yxz.mf *.class *.gif

- The following command lists the contents of **Xyz.jar**:

  jar tf Xyz.jar

- The following command extracts the contents of **Xyz.jar** and places those files in the current directory:

  jar xf Xyz.jar

- The following command adds the file **file1.class** to **Xyz.jar**:

  jar -uf Xyz.jar file1.class

- The following command adds all files below **directoryX** to **Xyz.jar**:

  jar -uf Xyz.jar -C directoryX *

Manifest Files:-

- A manifest file indicates which of the components in a JAR file are Java Beans.
- An example of a manifest file is provided in the following listing. It defines a JAR file that contains four .gif files and one .class file. The last entry is a Bean.

  Name: sunw/demo/slides/slide0.gif

  Name: sunw/demo/slides/slide1.gif

  Name: sunw/demo/slides/slide2.gif

  Name: sunw/demo/slides/slide3.gif

  Name: sunw/demo/slides/Slides.class

Java-Bean: True

- A manifest file may reference several .class files. If a .class file is a Java Bean, its entry must be immediately followed by the line "**Java-Bean: True**".

Introspection:-

- Introspection is the process of analyzing a Bean to determine its capabilities.
- It allows an application builder tool to present information about a component to a software designer.
- There are two ways in which the developer of a Bean can indicate which of its properties, events, and methods should be exposed by an application builder tool.
  - In the first method, simple naming conventions are used. These allow the introspection mechanisms to infer information about a Bean.
  - In the second way, an additional class is provided that explicitly supplies this information.
- The following sections indicate the design patterns for properties and events that enable the functionality of a Bean to be determined.

4.3 JAVABEANS PROPERTIES

Design Patterns for Properties:-

- Java Bean consists of properties, methods and events, which is called as design patterns.
- A property is a subset of a Bean's state. The values assigned to the properties determine the behavior and appearance of that component.
- There are three types of properties: **simple, Boolean, and indexed.**

1. Simple Properties

- A simple property has a single value.
- Design pattern for simple property is:

  public T getN( );

  public void setN(T arg);
- where N is the name of the property and T is its type.
- A read/write property has both of these methods to access its values.
- A read-only property has only a get method. A write-only property has only a set method.

The following listing shows a class that has two read/write simple properties:

```
public class Student
{
        private int rollno,marks;
        public int getRollno( )
        {
                return rollno;
        }
        public void setRollno(int d)
        {
                rollno = d;
         }
        public int getMarks( )
        {
                return marks;
        }
        public void setMarks(int h)
        {
                marks = h;
         }
}
```

2. Boolean Properties:-

- A Boolean property has a value of **true** or **false**.

  Design patterns are:-

```
public boolean isN( );
public boolean getN( );
public void setN(boolean value);
Where N is the name of the property:
```

- Either the first or second pattern can be used to retrieve the value of a Boolean property.

The following listing shows a class that has one Boolean property:

```
public class Line
{
        private boolean dotted = false;
        public boolean isDotted( )
        {
                return dotted;
         }
        public void setDotted(boolean dotted)
        {
                this.dotted = dotted;
        }
}
```

3. Indexed Properties:-

- An indexed property consists of multiple values.
- Design patterns are:-
  public T getN(int index);
  public void setN(int index, T value);
  public T[ ] getN( );
  public void setN(T values[ ]);
- Where N is the name of the property and T is its type.

The following listing shows a class that has one read/write indexed property:

```
public class PieChart
{
        private double data[ ];
        public double getData(int index)
        {
                return data[index];
        }
        public void setData(int index, double value)
        {
                data[index] = value;
        }
        public double[ ] getData( )
        {
                return data;
        }
        public void setData(double[ ] values)
        {
                data = new double[values.length];
                System.arraycopy(values, 0, data, 0, values.length);
        }
}
```

4. Design Patterns for Events:-

- Beans use the delegation event model.
- Beans can generate events and send them to other objects.
- Design patterns are:-

  public void addTListener(TListener eventListener);

  public void addTListener(TListener eventListener) throws TooManyListeners;

  public void removeTListener(TListener eventListener);

- Where T is the type of the event.
- These methods are used by event listeners to register an interest in events of a specific type.
- The first pattern indicates that a Bean can multicast an event to multiple listeners.
- The second pattern indicates that a Bean can unicast an event to only one listener.
- The third pattern is used by a listener when it no longer wishes to receive a specific type of event notification from a Bean.

The following listing outlines a class that notifies other objects when a temperature value moves outside a specific range.

```
public class Thermometer
{
        public void addTemperatureListener(TemperatureListener tl)
        {
                        ...
    }
    public void removeTemperatureListener(TemperatureListener tl)
    {
                        ...
    }
}
```

5. Bound Properties:-

- A Bean that has a bound property generates an event when the property is changed.
- The event is of type PropertyChangeEvent and is sent to objects that previously registered an interest in receiving such notifications.
- Sometimes when a Bean property changes, another object might need to be notified of the change, and react to the change. Whenever a bound property changes, notification of the change is sent to interested listeners.
- The accessor methods for a bound property are defined in the same way as those for simple properties. However, you also need to provide the event listener registration methods forPropertyChangeListener classes and fire a PropertyChangeEvent event to the PropertyChangeListener objects by calling their propertyChange methods.
- **Class:-**
    - PropertyChangeEvent class:- The PropertyChangeEvent class encapsulates property change information, and is sent from the property change event source to each object in the property change listener list with the propertyChange method.
- **Methods:-**
    - public void **addPropertyChangeListener**(PropertyChangeListener listener):-
        - Add a PropertyChangeListener to the listener list. The listener is registered for all properties.

- public void **firePropertyChange**(String propertyName, Object oldValue, Object newValue):- Report a bound property update to any registered listeners. No event is fired if old and new are equal and non-null.
- public void **removePropertyChangeListener**(PropertyChangeListener listener):-Remove a PropertyChangeListener from the listener list. This removes a PropertyChangeListener that was registered for all properties.

6. Constrained Properties:-

- The java bean component allows for the possibility that one or more listener objects might not allow certain changes to the value of a property. This is known as Constrained properties.
- A Bean that has a constrained property generates an event when an attempt is made to change its value. The event is of type **PropertyChangeEvent**. It is sent to objects that previously registered an interest in receiving such notifications. Those other objects have the ability to veto the proposed change.
- Constrained properties are more complicated than bound properties because they also support property change listeners which happen to be vetoers.
- The following operations in the setXXX method for the constrained property must be implemented in this order:
  - Save the old value in case the change is vetoed.
  - Notify listeners of the new proposed value, allowing them to veto the change.
  - If no listener vetoes the change (no exception is thrown), set the property to the new value.
- **Class:-**
  - **VetoableChangeSupport**:-This class can be used by beans that support constrained properties. You can use an instance of this class as a member field of your bean and delegate various work to it.
- **Methods:-**
  - public void **addVetoableChangeListener**(VetoableChangeListener listener):-Add a VetoableListener to the listener list. The listener is registered for all properties.

- public void **fireVetoableChange**(String propertyName, Object oldValue, Object newValue) throws PropertyVetoException:- Report a vetoable property update to any registered listeners. If anyone vetos the change, then fire a new event reverting everyone to the old value and then rethrow the PropertyVetoException. No event is fired if old and new are equal and non-null.
- public void **removeVetoableChangeListener**(VetoableChangeListener listener):-Remove a VetoableChangeListener from the listener list. This removes a VetoableChangeListener that was registered for all properties.

7. BeanInfo Interface:-

- A bean implementor who wishes to provide explicit information about their bean may provide a BeanInfo class that implements this BeanInfo interface and provides explicit information about the methods, properties, events, etc, of their bean.
- This interface is used to determine design patterns.
- This interface defines several methods, including these:
  - **PropertyDescriptor[ ] getPropertyDescriptors( ):-** returns an array of PropertyDescriptors describing the editable properties supported by this bean.
  - **EventSetDescriptor[ ] getEventSetDescriptors( ):-** returns an array of EventSetDescriptors describing the kinds of events fired by this bean.
  - **MethodDescriptor[ ] getMethodDescriptors( ):-** returns an array of MethodDescriptors describing the externally visible methods supported by this bean.
- By implementing these methods, a developer can designate exactly what is presented to a user.
- **SimpleBeanInfo** is a class that provides default implementations of the **BeanInfo** interface, including the three methods just shown.
  - You may extend this class and
  - override one or more of them.
- Example**:-**The following example shows how this is done for the **Colors**
- Bean. This bean displays a colored ellipse which changes its color randomly and shape to rectangle. **ColorsBeanInfo** is a subclass of **SimpleBeanInfo**.

- It overrides **getPropertyDescriptors( )** in order to designate which properties are presented to a Bean user. This method creates a **PropertyDescriptor** object for the **rectangular** property. The **PropertyDescriptor** constructor that is used is shown here:

- **PropertyDescriptor(String property, Class beanCls) throws IntrospectionException**.

- Here, the first argument is the name of the property, and the second argument is the class of the Bean.

```
// A Bean information class.

package sunw.demo.colors;
import java.beans.*;

public class ColorsBeanInfo extends SimpleBeanInfo
{
        public PropertyDescriptor[] getPropertyDescriptors( )
        {
                try
                {
                        PropertyDescriptor rectangular = new
                        PropertyDescriptor("rectangular",
Colors.class);
                        PropertyDescriptor pd[] = {rectangular};
                        return pd;
                }
                catch(Exception e)
                 {
                 }
                return null;
         }
}
```

- You must compile this file from the BDK\\demo directory or set CLASSPATH so that it includes c:\\bdk\\demo. If you don't, the compiler won't find the Colors.class file properly.

- After this file is successfully compiled, the colors.mft file can be updated, as shown here:

  Name: sunw/demo/colors/ColorsBeanInfo.class

  Name: sunw/demo/colors/Colors.class

  Java-Bean: True

8. Persistence:-

- Persistence is the ability to save a Bean to nonvolatile storage and retrieve it at a later time.

- In the first case, we can manually save the bean by selecting File/Save menu from the menu bar of the bean box. This allows you to specify the name of a file to which the beans and their configuration parameters should be saved. To restore, go to the menu bar and select File/Load.

- **Example:-**

    - Let us first see how the BDK allows you to save a set of Beans that have been configured and connected together to form an application.Suppose we have two beans colors and TickTock. The rectangular property of the Colors Bean was changed to true, and the interval property of the TickTock Bean was changed to one second.

    - To save the application, go to the menu bar of the BeanBox and select File Save. A dialog box should appear, allowing you to specify the name of a file to which the Beans and their configuration parameters should be saved. Supply a filename and click the OK button on that dialog box. Exit from the BDK.

    - Start the BDK again. To restore the application, go to the menu bar of the BeanBox and select File | Load. A dialog box should appear, allowing you to specify the name of the file from which an application should be restored. Supply the name of the file in which the application was saved, and click the OK button.

    - Your application should now be functioning.

- In the second case, object serialization is used. To do this, the bean class must implement a java.io.serializable interface.

- The object serialization capabilities provided by the Java class libraries are used to provide persistence for Beans.

- If a Bean inherits directly or indirectly from **java.awt.Component**, it is automatically serializable, because that class implements the **java.io.Serializable** interface. If a Bean does not inherit an implementation of the **Serializable** interface, you must provide this yourself. Otherwise, containers cannot save the configuration of your component.

- The **transient** keyword can be used to designate data members of a Bean that should not be serialized.

9. The Java Beans API:-

- The Java Beans functionality is provided by a set of classes and interfaces in the **java.beans** package.

The Interfaces Defined in java.beans:

- **AppletInitializer**:- Methods in this interface are used to initialize Beans that are also applets.
- **BeanInfo:-** This interface allows a designer to specify information about the properties, events, and methods of a Bean.
- **Customizer:-**This interface allows a designer to provide a graphical user interface through which a Bean may be configured.
- **DesignMode**:- Methods in this interface determine if a Bean is executing in design mode.
- **PropertyChangeListener**:-A method in this interface is invoked when a bound property is changed.
- **PropertyEditor:-** Objects that implement this interface allow designers to change and display property values.
- **VetoableChangeListener**:-A method in this interface is invoked when a constrained property is changed.
- **Visibility**:-Methods in this interface allow a Bean to execute in environments where a graphical user interface is not available.

10. The Classes Defined in java.beans:-

- **BeanDescriptor:-**This class provides information about a Bean. It also allows you to associate a customizer with a Bean.
- **Beans:-**This class is used to obtain information about a Bean.
- **EventSetDescriptor:-**Instances of this class describe an event that can be generated by a Bean.
- **IndexedPropertyDescriptor:-** Instances of this class describe an indexed property of a Bean.
- **FeatureDescriptor:-**This is the superclass of the PropertyDescriptor,EventSetDescriptor, and MethodDescriptor classes.
- **IntrospectionException:-**An exception of this type is generated if a problem occurs when analyzing a Bean.
- **Introspector:-** This class analyzes a Bean and constructs a BeanInfo object that describes the component.
- **MethodDescriptor**:-Instances of this class describe a method of a Bean.

- **ParameterDescriptor:-**Instances of this class describe a method parameter.
- **PropertyChangeEvent:-**This event is generated when bound or constrained properties are changed. It is sent to objects that registered an interest in these events and implement either the PropertyChangeListener or VetoableChangeListener interfaces.
- Example:- The following program illustrates the **Introspector**, **BeanDescriptor**,
    - **PropertyDescriptor**, and **EventSetDescriptor** classes and the **BeanInfo** interface. It Lists the properties and events of the **Colors** Bean.

```
//Show properties and events.
package sunw.demo.colors;
import java.awt.*;
import java.beans.*;

    public class IntrospectorDemo
        {
        public static void main(String args[ ])
            {
            try {
Class c = Class.forName("sunw.demo.colors.Colors");
BeanInfo beanInfo = Introspector.getBeanInfo(c);
BeanDescriptor beanDescriptor =beanInfo.getBeanDescriptor( );
System.out.println("Bean name = " +beanDescriptor.getName( ));
System.out.println("Properties:");
PropertyDescriptor propertyDescriptor[ ]=
        beanInfo.getPropertyDescriptors();

    for(int i = 0; i < propertyDescriptor.length; i++)
        {
        System.out.println("\\t" +propertyDescriptor[i].getName( ));
        }
```

```
            System.out.println("Events:");
            EventSetDescriptor eventSetDescriptor[] =
                  beanInfo.getEventSetDescriptors();

      for(int i = 0; i < eventSetDescriptor.length; i++)
            {
            System.out.println("\\t" +eventSetDescriptor[i].getName());
            }
                        }
                  catch(Exception e) {
                  System.out.println("Exception caught. " + e);
                  }
            }
      }
```

The output from this program is the following:

Bean name = Colors

Properties:

rectangular

Events:

propertyChange

mouseMotion

focus , mouse, inputMethod, key, component.

4.4. SUMMARY

- JavaBeans is a portable, platform-independent model written in Java Programming Language. Its components are referred to as beans.
- A JavaBean property can be accessed by the user of the object.
- JavaBeans components are built purely in Java, hence are fully portable to any platform
- JavaBeans is pretty compatible, there isn't any new complicated mechanism for registering components with the run-time system.

4.5. REFERENCE FOR FURTHER READING

1. Herbert Schildt, Java2: The Complete Reference, Tata McGraw-Hill,5th Edition

2. Joe Wigglesworth and Paula McMillan, Java Programming: Advanced Topics, Thomson Course Technology (SPD) ,3rd Edition

4.6. UNIT END EXERCISES

1. What are Java Beans? What are the advantages of using Java Beans?

2. What are JAR files? What are the options available with the JAR files? List any five options.

3. What is meant by the Design Pattern of the Beans? What are the property types that support design patterns?

4. Explain Beanlnfo Interface briefly. And what are the different Descriptors that contain in Beanlnfo classes

5. Define manifest file of Beans & explain the concept of Introspection of the Bean.

❄❄❄❄❄❄❄

5

STRUTS 2 PART-I

Unit Structure:

5.1 OBJECTIVE

1. Building applications on the web using struts 2.
2. Using web frameworks.
3. To understand and explore The Struts 2 framework.

5.2 INTRODUCTION

- Struts is an application development framework that is designed for and used with the popular J2EE platform.
- It cuts time out of the development process and makes developers more productive by providing them a series of tools and components to build applications with.
- Struts falls under the Jakarta subproject of the Apache Software Foundation and comes with an Open Source license.
- The struts framework is an open source framework for creating well-structured web based applications.
- The struts framework is based on the **Model View Controller** (MVC) paradigm which distinctly separates all the three layers - Model (state of the application), View (presentation) and Controller (controlling the application flow).
- This makes struts different from conventional JSP applications where sometimes logic, flow and UI are mingled in a Java Server Page.

- The struts framework is a complete **web framework** as it provides complete web form components, validators, error handling, internationalization, tiles and more. Struts framework provides its own Controller component.
- Struts can integrate well with Java Server Pages (JSP), Java Server Faces (JSF), JSTL, Velocity templates and many other presentation technologies for View.
- For Model, Struts works great with data access technologies like JDBC, Hibernate, EJB.
- Struts: a collection of Java code designed to help you build solid applications while saving time. It provides the basic skeleton and plumbing;
- Second, the benefit of using a framework is that it allows your code to be highly platform independent. For example, the same Struts code should work under Tomcat on an old Windows machine as runs using Weblogic on Linux or Solaris in production.

5.3 BASIC MVC ARCHITECTURE,

- Struts is based on the time-proven Model-View-Controller (MVC) design pattern.
- The MVC pattern is widely recognized as being among the most well-developed and mature design patterns in use.
- By using the MVC design pattern, processing is broken into three distinct sections aptly named the Model, the View, and the Controller.

These are described in the following subsections:

A) Model Components

- Model components provide a "model" of the business logic or data behind a Struts program. For example, in a Struts application that manages customer information, it may be appropriate to have a "Customer" Model component that provides program access to information about customers.
- It's very common for Model components to provide interfaces to databases or back-end systems. For example, if a Struts application needs to access employee information that is kept in an enterprise HR information system, it might be appropriate to design an "Employee" Model component that acts as an interface between the Struts application and the HR information system.
- Model components are generally standard Java classes. There is no specifically required format for a Model component, so it may be possible to reuse Java code written for other projects.

B) View Components

- View components are those pieces of an application that present information to users and accept input. In Struts applications, these correspond to Web pages.

- View components are used to display the information provided by Model components. For example, the "Customer" Model component needs a View component to display its information. Usually, there will be one or more View components for each Web page in a Struts application.
- View components are generally built using JavaServer Page files. Struts provides a large number of "**JSP Custom Tags**" which extend the normal capabilities of JSP and simplify the development of View components.

C) Controller Components

- Controller components coordinate activities in the application.
- This may mean taking data from the user and updating a database through a Model component, or it may mean detecting an error condition with a back-end system and directing the user through special error processing.
- Controller components accept data from the users, decide which Model components need to be updated, and then decide which View component needs to be called to display the results.
- One of the major contributions of Controller components is that they allow the developer to remove much of the error handling logic from the JSP pages in their application.
- This can significantly simplify the logic in the pages and make them easier to develop and maintain.
- Controller components in Struts are Java classes and must be built using specific rules. They are usually referred to as "Action classes."

Architecture Overview

- All incoming requests are intercepted by the Struts servlet controller.
- The Struts Configuration file struts-config.xml is used by the controller to determine the routing of the flow.
- This flows consists of an alternation between two transitions:

From View to Action	A user clicks on a link or submits a form on an HTML or JSP page. The controller receives the request, looks up the mapping for this request, and forwards it to an action. The action in turn calls a Model layer (Business layer) service or function.
From Action to View	After the call to an underlying function or service returns to the action class, the action forwards to a resource in the View layer and a page is displayed in a web browser.

5.4 STRUTS 2 FRAMEWORK FEATURES

1. Simple and Easy
2. Simplified Design
3. Tag support
4. Easy to modify tags
5. Easy Plugins
6. Stateful Checkboxes
7. Configurable MVC components
8. POJO based actions
9. AJAX support
10. Integration support
11. Various Result Types
12. Various Tag support
13. Theme and Template support

Simplified Design

- Programming abstract classes instead of interfaces is one of the design problems of struts1 framework that has been resolved in the struts 2 framework.
- Most of the Struts 2 classes are based on interfaces and most of its core interfaces are HTTP independent.
- Struts 2 Action classes are framework independent and are simplified to look as simple POJOs.
- Framework components are tried to keep loosely coupled.

Tag support

- Struts2 has improved the form tags and the new tags allow the developers to write less code.

Easy to modify tags

- Tag markups in Struts2 can be tweaked using Freemarker templates. This does not require JSP or java knowledge. Basic HTML, XML and CSS knowledge is enough to modify the tags.

Easy Plugins

- Struts 2 extensions can be added by dropping in a JAR. No manual configuration is required

AJAX Support

- The AJAX theme gives interactive applications a significant boost. The framework provides a set of tags to help you ajaxify your applications, even on Dojo.

Stateful Checkboxes

- Struts 2 checkboxes do not require special handling for false values.

QuickStart

- Many changes can be made on the fly without restarting a web container.

Customizing Controller

- Struts 1 lets you customize the request processor per module, Struts 2 lets you customize the request handling per action, if desired.

Easy Spring Integration

- Struts 2 Actions are Spring-aware. Just need to add Spring beans.

5.5 STRUTS 2 MVC PATTERN

- Struts2 is a pull-MVC framework.
- The Model-View-Controller pattern in Struts2 is implemented with the following five core components
  - Actions
  - Interceptors
  - Value Stack / OGNL
  - Results / Result types
  - View technologies
- Struts 2 is slightly different from a traditional MVC framework, where the action takes the role of the model rather than the controller, although there is some overlap.
- Struts 2 follows the Model-View-Controller (MVC) design patterns. The figure shown below shows how Struts 2 framework implements MVC components.
  - Action - model
  - Result - view
  - FilterDispatcher - controller
- The role each module plays Controller's job is to map incoming HTTP requests to actions. Those mapping are defined by using XML-based configuration(struts.xml) or Java annotations.

- The Model in Struts 2 is actions. So each action is defined and implemented by following the framework defined contract, an example consisting of an execute() method.
- Model components consist of the data storage and business logic. Each action is an encapsulation of requests and is placed in ValueStack.
- View is the presentation component of MVC pattern.

5.6 REQUEST LIFE CYCLE

- In accordance with Fig. 1, We need to understand the workflow by way of user's request life cycle in Struts 2 as shown below:

1. User sends a request to the server requesting for some resources /pages.
2. The Filter Dispatcher looks at the request and then regulates the appropriate Action.
3. Configured interceptor functionalities apply such as validation, file upload etc.
4. Selected action is performed based on the requested operation.
5. Once more, configured interceptors are applied to do any post processing if required.
6. Lastly, the result is composed by the view and returns the result to the user.

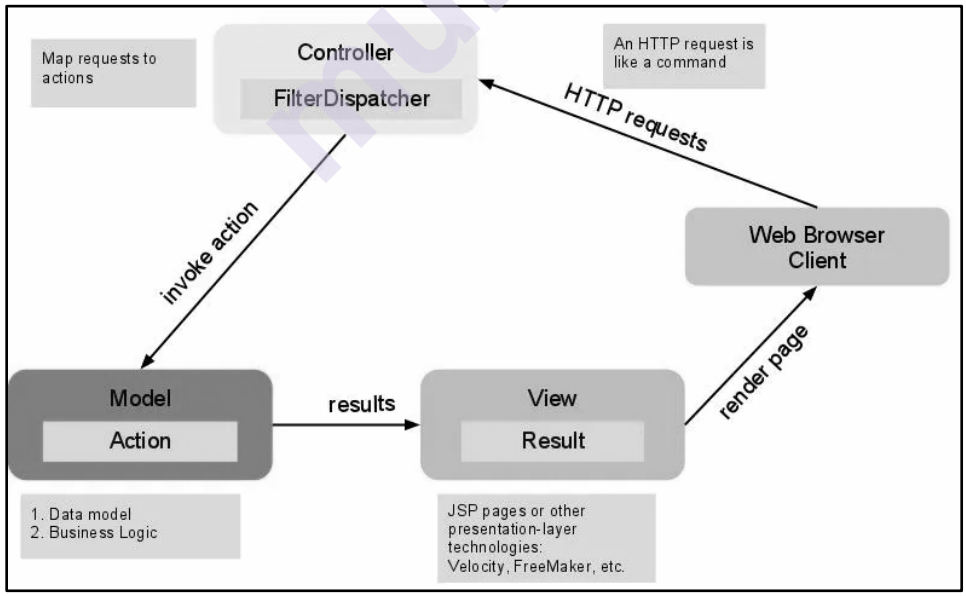
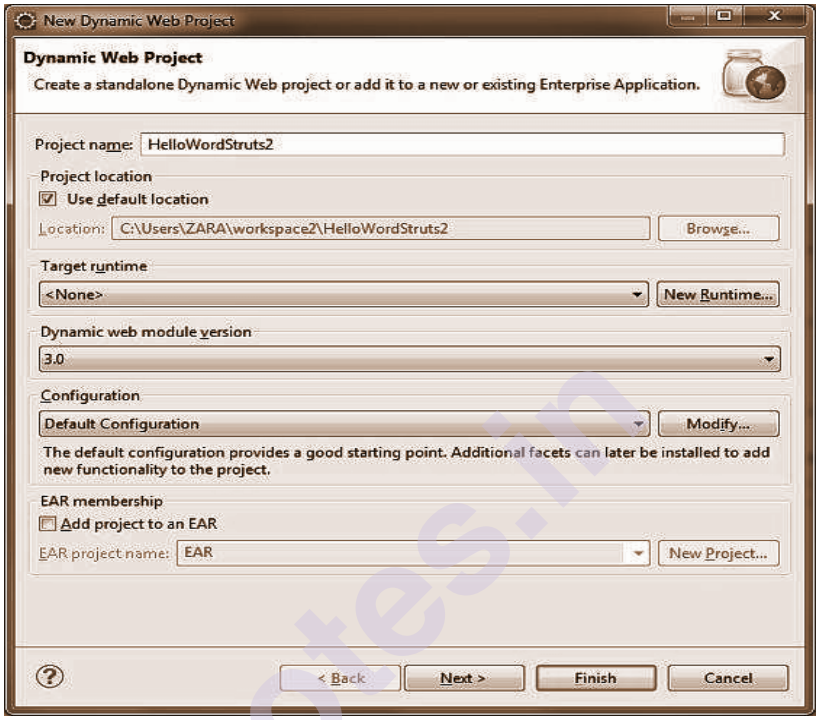


Fig. 1 Interactions among each MVC module

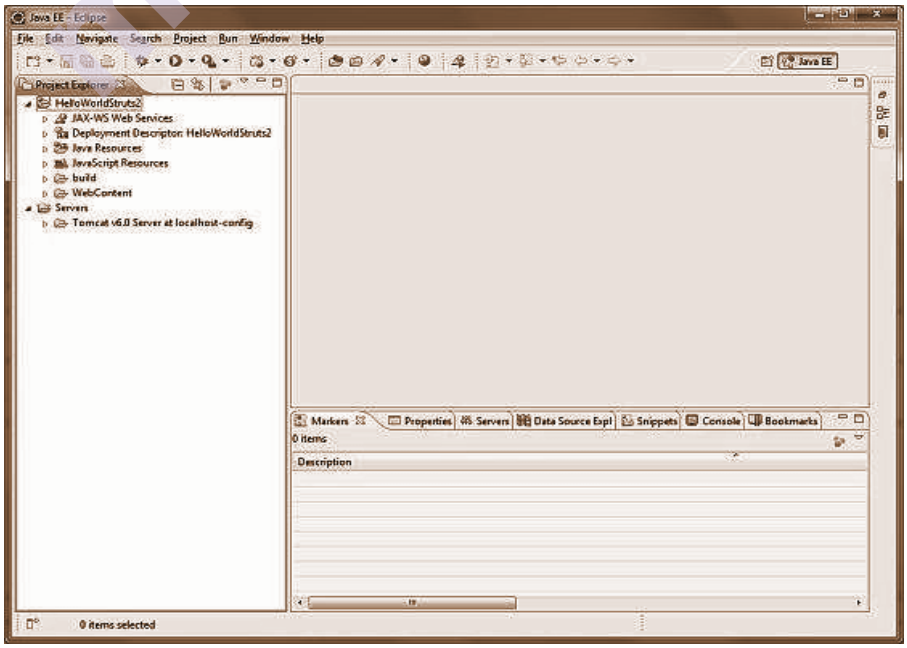
Example:

Creating a Dynamic Web Project

Start your Eclipse and then go with File > New > Dynamic Web Project and enter project name as HelloWorldStruts2 and set rest of the options as given in the following screen



Select all the default options in the next screens and finally check Generate Web.xml deployment descriptor option. This will create a dynamic web project for you in Eclipse. Now go with Windows > Show View > Project Explorer, and you will see your project window something as below:



Now copy the following files from struts 2 lib folder C:\struts-2.2.3\lib to our project's WEB-INF\lib folder. To do this, you can simply drag and drop all the following files into the WEB-INF\lib folder.

- commons-fileupload-x.y.z.jar
- commons-io-x.y.z.jar
- commons-lang-x.y.jar
- commons-logging-x.y.z.jar
- commons-logging-api-x.y.jar
- freemarker-x.y.z.jar
- javassist-.xy.z.GA
- ognl-x.y.z.jar
- struts2-core-x.y.z.jar
- xwork-core.x.y.z.jar

Create Action Class

Create a java file HelloWAction.java under Java Resources > src with a package name com.idoluniversity.struts2 with the contents given below.

The Action class responds to a user action when the user clicks a URL. One or more of the Action class's methods are executed and a String result is returned. Based on the value of the result, a specific JSP page is rendered.

Example:

```
package com.idoluniversity.struts2;
public class HelloWAction {
  private String name;
  public String execute() throws Exception {
    return "success";
  }

  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
}
```

Create a View

- Create the below jsp file HelloWorld.jsp in the WebContent folder in your eclipse project. To do this, right click on the WebContent folder in the project explorer and select New >JSP File.

```
<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>
  <head>
    <title>Hello World</title>
  </head>

  <body>
    Hello World, <s:property value = "name"/>
  </body>
</html>
```

- The taglib directive tells the Servlet container that this page will be using the Struts 2 tags and that these tags will be preceded by s.
- The s:property tag displays the value of action class property "name> which is returned by the method getName() of the HelloWAction class.

Create Main Page

- Create index.jsp in the WebContent folder. This file will serve as the initial action URL where a user can click to tell the Struts 2 framework to call a defined method of the HelloWAction class and render the HelloWorld.jsp view.

```
<%@ page language = "java" contentType = "text/html; charset = ISO-8859-1"
   pageEncoding = "ISO-8859-1"%>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Hello World</title>
  </head>
    <body>
    <h1>Hello World From Struts2</h1>
```

```
    <form action = "hello">
      <label for = "name">Please enter your name</label><br/>
      <input type = "text" name = "name"/>
      <input type = "submit" value = "Say Hello"/>
    </form>
  </body>
</html>
```

- Create a file called struts.xml. Since Struts 2 requires struts.xml to be present in the classes folder. Hence, create struts.xml file under the WebContent/WEB-INF/classes folder. Eclipse does not create the "classes" folder by default. Right click on the WEB-INF folder in the project explorer and select New > Folder. Your struts.xml should look like.

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />

  <package name = "helloworld" extends = "struts-default">
    <action name = "hello"
      class = "com.idoluniversity.struts2.HelloWAction"
      method = "execute">
      <result name = "success">/HelloWorld.jsp</result>
    </action>
  </package>
</struts>
```


- Creating a package is useful when you want to group your actions together. In our example, we named our action as "hello" which is corresponding to the URL /hello.action and is backed up by theHelloWAction.class. The execute method of HelloWAction.class is the method that is run when the URL /hello.action is invoked. If the outcome of the execute method returns "success", then we take the user to HelloWorld.jsp.

- Then create a web.xml file which is an entry point for any request to Struts 2. The entry point of Struts2 application will be a filter defined in deployment descriptor (web.xml).

- Hence, we will define an entry of org.apache.struts2.dispatcher.FilterDispatcher class in web.xml. The web.xml file needs to be created under the WEB-INF folder under WebContent. Eclipse had already created a skeleton web.xml file for you when you created the project. So, let's just modify it as follows −

Example:

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<web-app xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xmlns = "http://java.sun.com/xml/ns/javaee"
  xmlns:web = "http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id = "WebApp_ID" version = "3.0">

  <display-name>Struts 2</display-name>

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

  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

To Enable Detailed Log

- Creating logging.properties file under WEB-INF/classes folder. Keep the following two lines in your property file

  org.apache.catalina.core.ContainerBase.[Catalina].level = INFO

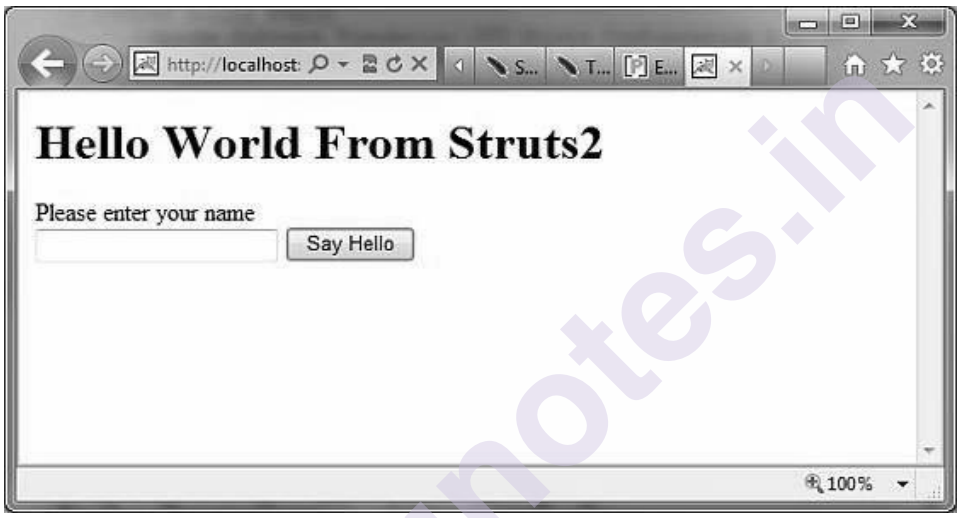
  org.apache.catalina.core.ContainerBase.[Catalina].handlers = \

  java.util.logging.ConsoleHandler

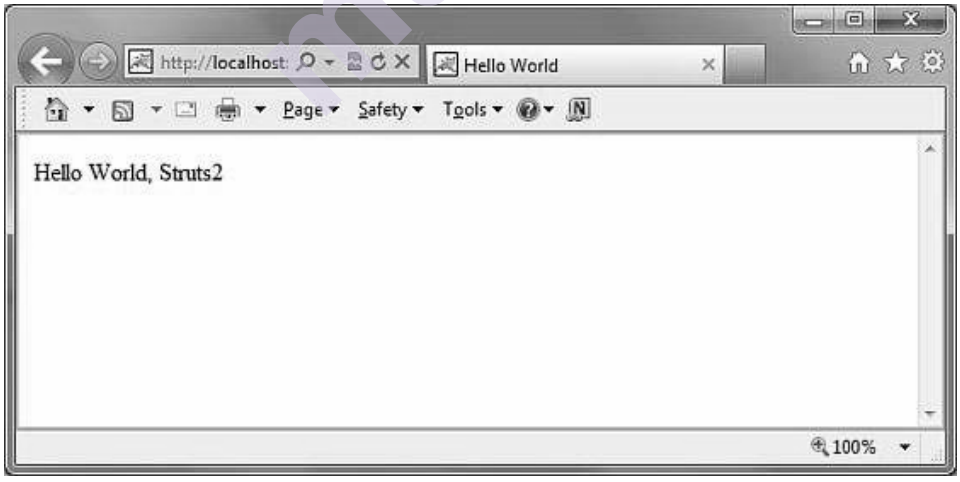
- The default logging.properties specifies a ConsoleHandler for routing logging to stdout and also a FileHandler. A handler's log level threshold can be set using SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST or ALL.

Procedure for Executing the Application

- Right click on the project name and click Export > WAR File to create a War file.
- Then deploy this WAR in the Tomcat's webapps directory.
- Finally, start the Tomcat server and try to access the URL http://localhost:8080/HelloWorldStruts2/index.jsp.
- This will give you following screen



Enter the value "Struts2" and submit the page.



5.7 CONFIGURATION FILES

- Configuration files like web.xml, struts.xml, strutsconfig.xml and struts.properties

The web.xml File

- The web.xml configuration file is a J2EE configuration file that determines how elements of the HTTP request are processed by the servlet container. It is not strictly a Struts2 configuration file, but it is a file that needs to be configured for Struts2 to work.
- The web.xml file needs to be created under the folder WebContent/WEB-INF.
- This is the first configuration file you will need to configure if you are starting without the aid of a template or tool that generates it.

Following is the content of the web.xml file which we used in our previous example.

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<web-app xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xmlns = "http://java.sun.com/xml/ns/javaee"
  xmlns:web = "http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id = "WebApp_ID" version = "3.0">
   <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
   <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

The Struts.xml File

- The struts.xml file contains the configuration information that you will be modifying as actions are developed. This file can be used to override default settings for an application, for example *struts.devMode = false* and other settings which are defined in the property file. This file can be created under the folder WEB-INF/classes.

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />
  <package name = "helloworld" extends = "struts-default">
    <action name = "hello"
      class = "com.idoluniversity.struts2.HelloWAction"
      method = "execute">
      <result name = "success">/HelloWorld.jsp</result>
    </action>
    <-- more actions can be listed here -->
  </package>
  <-- more packages can be listed here -->
</struts>
```

The Struts-config.xml File

- The struts-config.xml configuration file is a link between the View and Model components in the Web Client but you would not have to touch these settings for 99.99% of your projects.

The configuration file basically contains following main elements −

Sr.No	Interceptor & Description
1	struts-config This is the root node of the configuration file.
2	form-beans This is where you map your ActionForm subclass to a name. You use this name as an alias for your ActionForm throughout the rest of the strutsconfig.xml file, and even on your JSP pages.
3	global forwards This section maps a page on your webapp to a name. You can use this name to refer to the actual page. This avoids hardcoding URLs on your web pages.
4	action-mappings This is where you declare form handlers and they are also known as action mappings.
5	controller This section configures Struts internals and rarely used in practical situations.
6	plug-in This section tells Struts where to find your properties files, which contain prompts and error messages

Following is the sample struts-config.xml file

```
<?xml version = "1.0" Encoding = "ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
  "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
<struts-config>
  <!-- ========== Form Bean Definitions ============ -->
  <form-beans>
    <form-bean name = "login" type = "test.struts.LoginForm" />
  </form-beans>
  <!-- ========== Global Forward Definitions ========= -->
  <global-forwards>
  </global-forwards>
  <!-- ========== Action Mapping Definitions ======== -->
  <action-mappings>
    <action
      path = "/login"
      type = "test.struts.LoginAction" >
      <forward name = "valid" path = "/jsp/MainMenu.jsp" />
      <forward name = "invalid" path = "/jsp/LoginView.jsp" />
    </action>
  </action-mappings>
  <!-- ========== Controller Definitions ======== -->
  <controller contentType = "text/html;charset = UTF-8"
    debug = "3" maxFileSize = "1.618M" locale = "true" nocache =
"true"/>
</struts-config>
```


The Struts.properties File

- This configuration file provides a mechanism to change the default behavior of the framework.
- The values configured in this file will override the default values configured in default.properties which is contained in the struts2-core-x.y.z.jar distribution.

There are a couple of properties that you might consider changing using the struts.properties file

```
### When set to true, Struts will act much more friendly for developers
struts.devMode = true
### Enables reloading of internationalization files
```

```
struts.i18n.reload = true
### Enables reloading of XML configuration files
struts.configuration.xml.reload = true
### Sets the port that the server is run on
struts.url.http.port = 8080
```

5.8 ACTIONS

- Actions are the core of the Struts2 framework, as they are for any MVC (Model View Controller) framework. Each URL is mapped to a specific action, which provides the processing logic which is necessary to service the request from the user.
- Action also serves in two other important capacities.
    - Firstly, the action plays an important role in the transfer of data from the request through to the view, whether it's a JSP or other type of result.
    - Secondly, the action must assist the framework in determining which result should render the view that will be returned in the response to the request.

Create Action

- The only requirement for actions in Struts2 is that there must be one no argument method that returns either a String or Result object and must be a POJO. If the no-argument method is not specified, the default behavior is to use the execute() method.
- ActionSupport class which implements six interfaces including Action interface. The Action interface is as follows

```
public interface Action {
   public static final String SUCCESS = "success";
   public static final String NONE = "none";
   public static final String ERROR = "error";
   public static final String INPUT = "input";
   public static final String LOGIN = "login";
   public String execute() throws Exception;
}
```

- Action method in the Hello World example

```
package com.idoluniversity.struts2;
public class HelloWAction {
   private String name;
```

```
  public String execute() throws Exception {
    return "success";
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
}
```

- Following change to the execute method and extend the class ActionSupport as follows

```
package com.idoluniversity.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class HelloWAction extends ActionSupport {
  private String name;
  public String execute() throws Exception {
    if ("SECRET".equals(name)) {
      return SUCCESS;
    } else {
      return ERROR;
    }
  }
    public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
}
```

- Then modify our struts.xml file as follows

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration
2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />
  <package name = "helloworld" extends = "struts-default">
    <action name = "hello"
```

```
        class = "com.idoluniversity.struts2.HelloWAction"
        method = "execute">
        <result name = "success">/HelloWorld.jsp</result>
        <result name = "error">/AccessDenied.jsp</result>
      </action>
   </package>
</struts>
```

Create a View

- Create the below jsp file HelloWorld.jsp in the WebContent folder in your eclipse project. To do this, right click on the WebContent folder in the project explorer and select New >JSP File.

  Then defined in Action interface

```
<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>
   <head>
      <title>Hello World</title>
   </head>
  <body>
      Hello World, <s:property value = "name"/>
   </body>
</html>
```

- Following is the code which will be invoked by the framework in case the action result is ERROR which is equal to String constant "error". Following is the content of AccessDenied.jsp

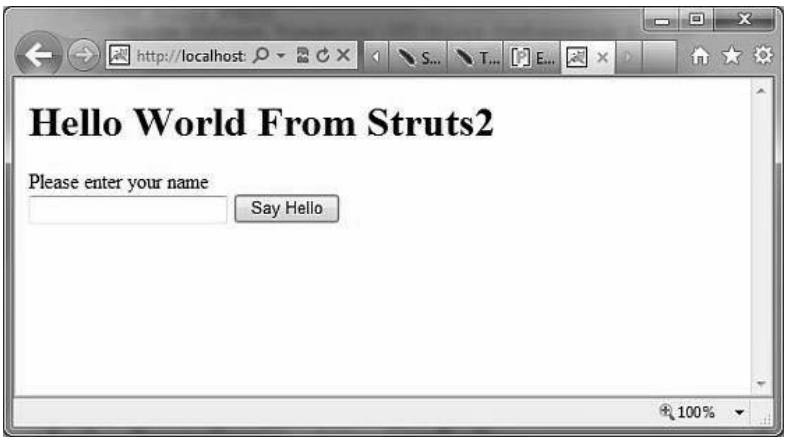
```
<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>
   <head>
      <title>Access Denied</title>
   </head>

   <body>
      You are not authorized to view this page.
   </body>
</html>
```

- We also need to create index.jsp in the WebContent folder. This file will serve as the initial action URL where the user can click to tell the Struts 2 framework to call the executemethod of the HelloWAction class and render the HelloWorld.jsp view.

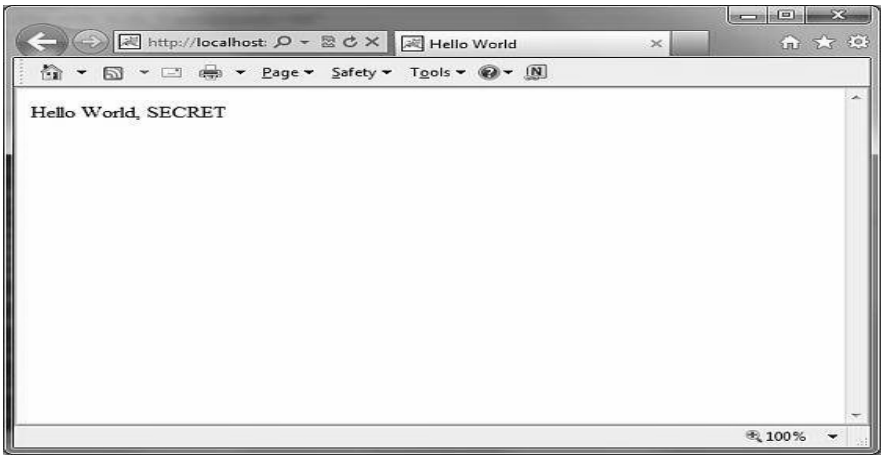
```
<%@ page language = "java" contentType = "text/html; charset = ISO-8859-1"
  pageEncoding = "ISO-8859-1"%>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Hello World</title>
  </head>

  <body>
    <h1>Hello World From Struts2</h1>
    <form action = "hello">
      <label for = "name">Please enter your name</label><br/>
      <input type = "text" name = "name"/>
      <input type = "submit" value = "Say Hello"/>
    </form>
  </body>
</html>
```

Execute the Application

- Right click on the project name and click Export > WAR File to create a War file. Then deploy this WAR in the Tomcat's webapps directory. Finally, start the Tomcat server and try to access the URL http://localhost:8080/HelloWorldStruts2/index.jsp.



- Let us enter a word as "SECRET" and you should see the following page



5.9 SUMMARY

- Apache Struts 2 is an open-source web application framework for developing Java EE web applications.
- It uses and extends the Java Servlet API to encourage developers to adopt a model view controller (MVC) architecture.

5.10. REFERENCE FOR FURTHER READING

1. Herbert Schildt, Java2: The Complete Reference, Tata McGraw-Hill,5th Edition
2. Joe Wigglesworth and Paula McMillan, Java Programming: Advanced Topics, Thomson Course Technology (SPD) ,3rd Edition

5.11 UNIT END EXERCISES

1. Explain the Struts 2 Basic MVC Architecture?
2. What are the Struts 2 framework features?
3. Explain the Request life cycle with the help of an example?
4. Write a short note on:
   a. Configuration Files
   b. Actions
5. Differentiate between struts.xml and web.xml

❄❄❄❄❄❄❄

6

STRUTS 2 PART-II

Unit Structure:

6.1 OBJECTIVE

1. Introducing interceptors and the ValueStack.
2. Bundling actions into packages & Implementing actions.
3. To understand the role of a view in the Struts2 MVC framework.
4. To learn Object-Graph Navigation Language.

6.2 INTRODUCTION

- The tasks that are done by the Struts 2 framework before and after an Action is executed are done by Struts 2 interceptors. Interceptors are standard Java classes included in the Struts 2 core jar which are executed in a specific order.
- In our example application there is a package node in struts.xml. The package node has an attribute of extends with a value of "struts-default."
- The value "struts-default" identifies to the framework the specific stack of interceptors that will be executed before and after the Actions in that package.

6.3 INTERCEPTORS

- Struts 2 framework provides a list of out-of-the-box interceptors that come pre-configured and ready to use. Few of the important interceptors are listed below.

Interceptor	Description
alias	Allows parameters to have different name aliases across requests.
checkbox	Assists in managing checkboxes by adding a parameter value of false for check boxes that are not checked.
conversionError	Places error information from converting strings to parameter types into the action's field errors.
createSession	Automatically creates an HTTP session if one does not already exist.
debugging	Provides several different debugging screens to the developer.
execAndWait	Sends the user to an intermediary waiting page while the action executes in the background.
exception	Maps exceptions that are thrown from an action to a result, allowing automatic exception handling via redirection.
fileUpload	Facilitates easy file uploading.
logger	Provides simple logging by outputting the name of the action being executed.
params	Sets the request parameters on the action.
prepare	This is typically used to do pre-processing work, such as setup database connections.
profile	Allows simple profiling information to be logged for actions.
scope	Stores and retrieves the action's state in the session or application scope.
ServletConfig	Provides the action with access to various servlet-based information.
timer	Provides simple profiling information in the form of how long the action takes to execute.
token	Checks the action for a valid token to prevent duplicate form submission.
validation	Provides validation support for actions

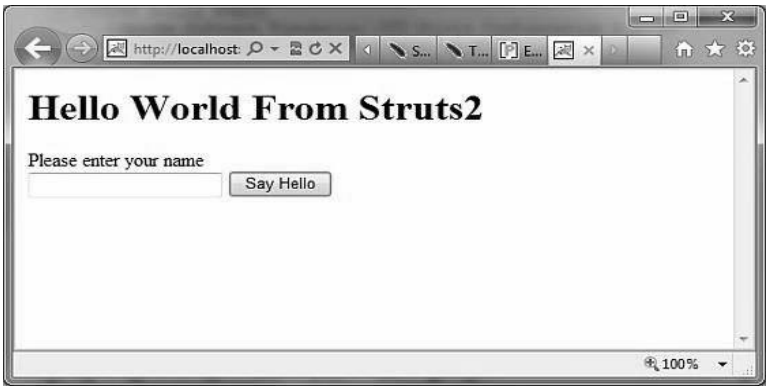
Use of Interceptors

- Let us consider the existing interceptor to our "Hello World" program.

- Let's use the timer interceptor whose purpose is to measure how long it took to execute an action method.
- Then use a params interceptor whose purpose is to send the request parameters to the action.
- Then we find that the name property is not being set because the parameter is not able to reach the action.
- Let us modify the struts.xml file to add an interceptor as follows

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />

  <package name = "helloworld" extends = "struts-default">
    <action name = "hello"
      class = "com.idoluniversity.struts2.HelloWorldAction"
      method = "execute">
      <interceptor-ref name = "params"/>
      <interceptor-ref name = "timer" />
      <result name = "success">/HelloWorld.jsp</result>
    </action>
  </package>
</struts>
```

- Right click on the project name and click Export > WAR File to create a War file. Then deploy this WAR in the Tomcat's webapps directory. Finally, start the Tomcat server and try to access the URL http://localhost:8080/HelloWorldStruts2/index.jsp. This will produce the following screen −



- Now enter any word in the given text box and click the Say Hello button to execute the defined action. Now if you will check the log generated.

```
INFO: Server startup in 3539 ms
27/08/2022 8:40:53 PM
com.opensymphony.xwork2.util.logging.commons.CommonsLog
ger info
INFO: Executed action [//hello!execute] took 109 ms.
```

Create Custom Interceptors

- Using custom interceptors the application is an elegant way to provide crosscutting application features. Creating a custom interceptor is easy, the interface that needs to be extended is the following Interceptor interface

```
public interface Interceptor extends Serializable {
   void destroy();
   void init();
   String intercept(ActionInvocation invocation)
   throws Exception;
}
```

- As the names suggest, the init() method provides a way to initialize the interceptor, and the destroy() method provides a facility for interceptor cleanup.
- Unlike actions, interceptors are reused across requests and need to be thread safe, especially the intercept() method.

Create Interceptor Class

- Let us create the following MyInterceptor.java in Java Resources > src folder

```
package com.idoluniversity.struts2;
import java.util.*;
import com.opensymphony.xwork2.ActionInvocation;
import
com.opensymphony.xwork2.interceptor.AbstractInterceptor;
public class MyInterceptor extends AbstractInterceptor {
   public String intercept(ActionInvocation invocation)throws
Exception {
```

```
    /* let us do some pre-processing */
    String output = "Pre-Processing";
    System.out.println(output);
    /* let us call action or next interceptor */
    String result = invocation.invoke();
    /* let us do some post-processing */
    output = "Post-Processing";
    System.out.println(output);
    return result;
  }
}
```

- The framework itself starts the process by making the first call to the ActionInvocation object's invoke(). Each time invoke() is called, ActionInvocation consults its state and executes whichever interceptor comes next.
- When all of the configured interceptors have been invoked, the invoke() method will cause the action itself to be executed.

The following figure shows the same concept through a request flow

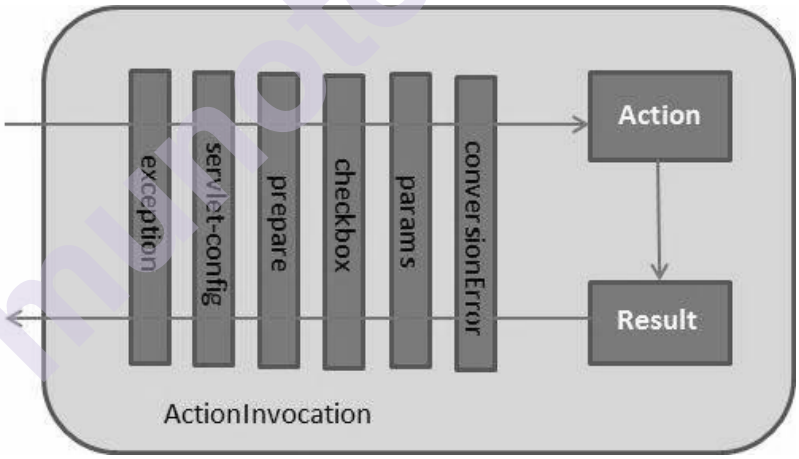


Fig.1. Request Flow

Create Action Class

- Create a java file HelloWorldAction.java under Java Resources > src with a package name com.idoluniversity.struts2 with the contents given below.

```
package com.idoluniversity.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class HelloWorldAction extends ActionSupport {
  private String name;
```

```
  public String execute() throws Exception {
    System.out.println("Inside action....");
    return "success";
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
}
```

Create a View

- Create the below jsp file HelloWorld.jsp in the WebContent folder in your eclipse project.

```
<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>
  <head>
    <title>Hello World</title>
  </head>
   <body>
    Hello World, <s:property value = "name"/>
  </body>
</html>
```

Create Main Page

- Also create index.jsp in the WebContent folder. This file will serve as the initial action URL where a user can click to tell the Struts 2 framework to call the a defined method of the HelloWorldAction class and render the HelloWorld.jsp view.

```
<%@ page language = "java" contentType = "text/html; charset =
ISO-8859-1"
  pageEncoding = "ISO-8859-1"%>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
```

```
    <title>Hello World</title>
  </head>
  <body>
    <h1>Hello World From Struts2</h1>
    <form action = "hello">
      <label for = "name">Please enter your name</label><br/>
      <input type = "text" name = "name"/>
      <input type = "submit" value = "Say Hello"/>
    </form>
  </body>
</html>
```

Configuration Files

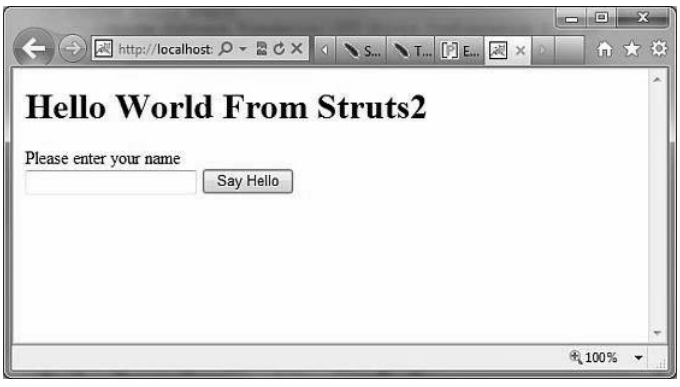
- Register an interceptor and then call it as we had called the default interceptor in the previous program. To register a newly defined interceptor, the <interceptors>...</interceptors> tags are placed directly under the <package> tag insstruts.xml file.
- We can skip this step for a default interceptors as we did in our previous example. But here let us register and use it as follows

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration
2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />
  <package name = "helloworld" extends = "struts-default">
    <interceptors>
      <interceptor name = "myinterceptorone"
        class = "com.idoluniversity.struts2.MyInterceptor" />
    </interceptors>
    <action name = "hello"
      class = "com.idoluniversity.struts2.HelloWorldAction"
      method = "execute">
      <interceptor-ref name = "params"/>
      <interceptor-ref name = "myinterceptorone" />
      <result name = "success">/HelloWorld.jsp</result>
    </action>
  </package>
</struts>
```

- The web.xml file needs to be created under the WEB-INF folder under WebContent as follows

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<web-app xmlns:xsi = "http://www.w3.org/2022/XMLSchema-instance"
  xmlns = "http://java.sun.com/xml/ns/javaee"
  xmlns:web = "http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id = "WebApp_ID" version = "3.0">
  <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

- Right click on the project name and click Export > WAR File to create a War file. Then deploy this WAR in the Tomcat's webapps directory. Finally, start the Tomcat server and try to access the URL http://localhost:8080/HelloWorldStruts2/index.jsp. This will produce the following screen



- Now enter any word in the given text box and click the Say Hello button to execute the defined action. Now if you will check the log generated, you will find the following text at the bottom

  Pre-Processing

  Inside action....

  Post-Processing

6.4 RESULTS & RESULT TYPES

- The <results> tag plays the role of a view in the Struts2 MVC framework. The action is responsible for executing the business logic. The next step after executing the business logic is to display the view using the <results> tag.
- There are some navigation rules attached with the results.
- For example, if the action method is to authenticate a user, there are three possible outcomes.
- Successful Login
- Unsuccessful Login - Incorrect username or password
- Account Locked

The Dispatcher Result Type

- The dispatcher result type is the default type, and is used if no other result type is specified. It's used to forward to a servlet, JSP, HTML page, and so on, on the server. It uses the RequestDispatcher.forward() method.
- Provided a JSP path as the body of the result tag.

```
<result name = "success">
  /HelloWorld.jsp
</result>
```

- Specify the JSP file using a <param name = "location"> tag within the <result...> element as follows

```
<result name = "success" type = "dispatcher">
  <param name = "location">
    /HelloWorld.jsp
  </param >
</result>
```

The FreeMaker Result Type

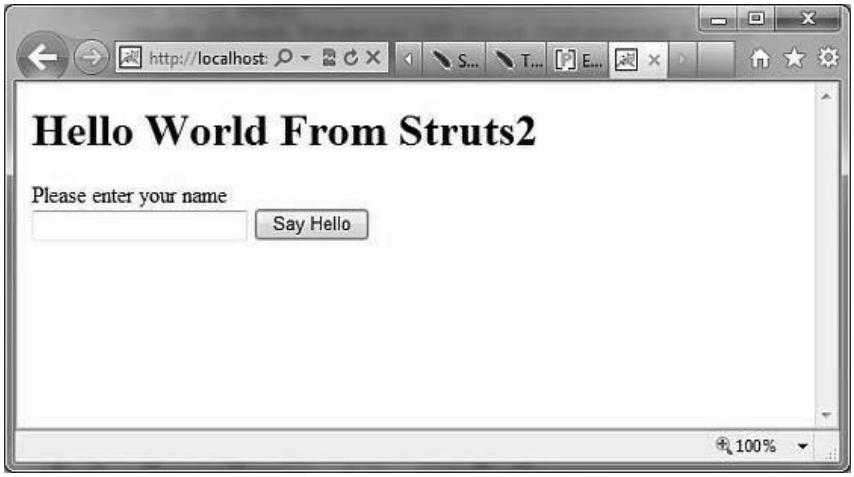
- Freemaker is a popular templating engine that is used to generate output using predefined templates.
- Let us now create a Freemaker template file called hello.fm with the following contents

Hello World ${name}

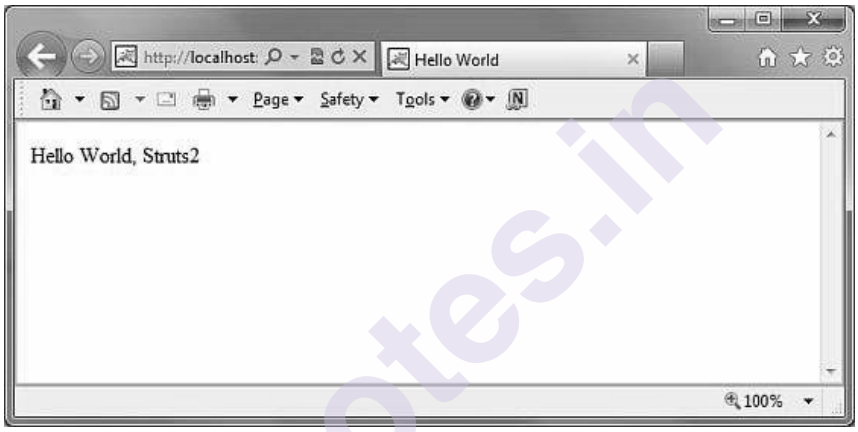
- The above file is a template where name is a parameter which will be passed from outside using the defined action. keep this file in CLASSPATH.
- Next, let us modify the struts.xml to specify the result as follows

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />
  <package name = "helloworld" extends = "struts-default">
    <action name = "hello"
      class = "com.idoluniversity.struts2.HelloWorldAction"
      method = "execute">
      <result name = "success" type = "freemarker">
        <param name = "location">/hello.fm</param>
      </result>
    </action>
  </package>
</struts>
```

- Let us keep HelloWorldAction.java, HelloWorldAction.jsp and index.jsp files
- Now Right click on the project name and click Export > WAR File to create a War file.
- Then deploy this WAR in the Tomcat's webapps directory. Finally, start the Tomcat server and try to access the URL http://localhost:8080/HelloWorldStruts2/index.jsp. This will produce the following screen .



Enter the value "Struts2" and submit the page. You should see the next page.



The Redirect Result Type

- The redirect result type calls the standard response.sendRedirect() method, causing the browser to create a new request to the given location.
- Provide the location either in the body of the <result...> element or as a <param name = "location"> element. Redirect also supports the parse parameter. Here's an example configured using XML

```
<action name = "hello"
  class = "com.idoluniversity.struts2.HelloWorldAction"
  method = "execute">
  <result name = "success" type = "redirect">
    <param name = "location">
      /NewWorld.jsp
    </param >
  </result>
</action>
```

6.5 VALUE STACK/OGNL

The Value Stack

- The value stack is a set of several objects which keeps the following objects in the provided order

Sr.No	Objects & Description
1	Temporary Objects There are various temporary objects which are created during execution of a page. For example the current iteration value for a collection being looped over in a JSP tag.
2	The Model Object If you are using model objects in your struts application, the current model object is placed before the action on the value stack.
3	The Action Object This will be the current action object which is being executed.
4	Named Objects These objects include #application, #session, #request, #attr and #parameters and refer to the corresponding servlet scopes.

- The value stack can be accessed via the tags provided for JSP, Velocity or Freemarker.
- get valueStack object inside your action as follows

  ActionContext.getContext().getValueStack()
- Once you have a ValueStack object, you can use the following methods to manipulate that object

Sr.No	ValueStack Methods & Description
1	Object findValue(String expr) Find a value by evaluating the given expression against the stack in the default search order.
2	CompoundRoot getRoot() Get the CompoundRoot which holds the objects pushed onto the stack.
3	Object peek() Get the object on the top of the stack without changing the stack.

4	Object pop() Get the object on the top of the stack and remove it from the stack.
5	void push(Object o) Put this object onto the top of the stack.
6	void set(String key, Object o) Sets an object on the stack with the given key so it is retrievable by findValue(key,...)
7	void setDefaultType(Class defaultType) Sets the default type to convert to if no type is provided when getting a value.
8	void setValue(String expr, Object value) Attempts to set a property on a bean in the stack with the given expression using the default search order.
9	int size() Get the number of objects in the stack.

The OGNL

- The Object-Graph Navigation Language (OGNL) is a powerful expression language that is used to reference and manipulate data on the ValueStack.
- OGNL also helps in data transfer and type conversion.
- The OGNL is very similar to the JSP Expression Language.
- OGNL is based on the idea of having a root or default object within the context.
- The properties of the default or root object can be referenced using the markup notation, which is the pound symbol.
- OGNL is based on a context and Struts builds an ActionContext map for use with OGNL. The ActionContext map consists of the following

  Application − Application scoped variables

  Session − Session scoped variables

  Root / value stack − All your action variables are stored here

  Request − Request scoped variables

  Parameters − Request parameters

  Attributes − The attributes stored in page, request, session and application scope

- It is important to understand that the Action object is always available in the value stack. So, therefore if an Action object has properties “x” and “y” they are readily available to use.
- Objects in the ActionContext are referred to using the pound symbol, however, the objects in the value stack can be directly referenced.
- For example, if employee is a property of an action class, then it can be referenced as follows

  <s:property value = "name"/>

  instead of

  <s:property value = "#name"/>
- If you have an attribute in session called "login" you can retrieve it as follows

  <s:property value = "#session.login"/>
- OGNL also supports dealing with collections - namely Map, List and Set. For example to display a dropdown list of colors, you could do

  <s:select name = "color" list = "{'red','yellow','green'}" />
- The OGNL expression is clever to interpret the "red","yellow","green" as colors and build a list based on that.

ValueStack/OGNL Example

Create Action

- Let us consider the following action class where we are accessing valueStack and then setting a few keys which we will access using OGNL in our view, i.e., JSP page.

```
package com.idoluniversity.struts2;
import java.util.*;
import com.opensymphony.xwork2.util.ValueStack;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class HelloWorldAction extends ActionSupport {
  private String name;
  public String execute() throws Exception {
    ValueStack stack = ActionContext.getContext().getValueStack();
    Map<String, Object> context = new HashMap<String, Object>();
    context.put("key1", new String("This is key1"));
    context.put("key2", new String("This is key2"));
```

```
    stack.push(context);
    System.out.println("Size of the valueStack: " + stack.size());
    return "success";
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
}
```

Create Views

- Let us create the below jsp file HelloWorld.jsp in the WebContent folder in your eclipse project. This view will be displayed in case action returns success

```
<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>
  <head>
    <title>Hello World</title>
  </head>

  <body>
    Entered value : <s:property value = "name"/><br/>
    Value of key 1 : <s:property value = "key1" /><br/>
    Value of key 2 : <s:property value = "key2" /> <br/>
  </body>
</html>
```

- Also create index.jsp in the WebContent folder whose content is as follows

```
<%@ page language = "java" contentType = "text/html; charset =
ISO-8859-1"
  pageEncoding = "ISO-8859-1"%>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <h1>Hello World From Struts2</h1>
    <form action = "hello">
      <label for = "name">Please enter your name</label><br/>
      <input type = "text" name = "name"/>
      <input type = "submit" value = "Say Hello"/>
    </form>
  </body>
</html>
```

Configuration Files

Following is the content of struts.xml file

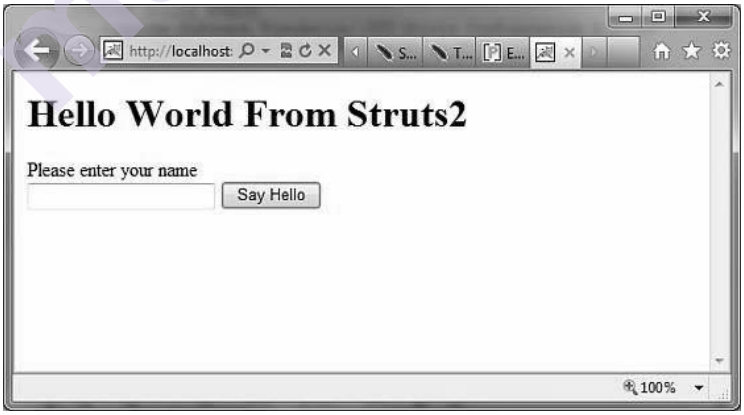
```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />
  <package name = "helloworld" extends = "struts-default">
    <action name = "hello"
      class = "com.idoluniversity.struts2.HelloWorldAction"
      method = "execute">
      <result name = "success">/HelloWorld.jsp</result>
    </action>
  </package>
</struts>
```

- Following is the content of web.xml file

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<web-app xmlns:xsi = "http://www.w3.org/2022/XMLSchema-instance"
  xmlns = "http://java.sun.com/xml/ns/javaee"
```

```
  xmlns:web = "http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id = "WebApp_ID" version = "3.0">
  <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

- Right click on the project name and click Export > WAR File to create a War file. Then deploy this WAR in the Tomcat's webapps directory. Finally, start the Tomcat server and try to access the URL http://localhost:8080/HelloWorldStruts2/index.jsp. This will produce the following screen



- Now enter any word in the given text box and click the "Say Hello" button to execute the defined action. Now, if you will check the log generated, you will find the following text at the bottom

  Size of the valueStack: 3

6.6 SUMMARY

- Struts 2 framework provides a list of out-of-the-box interceptors that come pre-configured and ready to use.
- The <results> tag plays the role of a view in the Struts2 MVC framework. The action is responsible for executing the business logic.
- The value stack is a set of several objects.
- The Object-Graph Navigation Language (OGNL) is a powerful expression language that is used to reference and manipulate data on the ValueStack.

6.7 REFERENCE FOR FURTHER READING

1. Herbert Schildt, Java2: The Complete Reference, Tata McGraw-Hill,5th Edition
2. Joe Wigglesworth and Paula McMillan, Java Programming: Advanced Topics, Thomson Course Technology (SPD) ,3rd Edition

6.8 UNIT END EXERCISES

1. What are interceptors?
2. Explain the Results & Result Types?
3. Write a short note on Value Stack & OGNL.
4. What is value stack in struts? State and explain the execution flow of value stack.
5. What is OGNL? What does it consist of? How its contents can be accessed?

❄❄❄❄❄❄❄

7

JSON

Unit Structure:

7.1 OBJECTIVE

- To study the overview of JSON.
- To understand the JSON syntax, DataTypes & Objects.
- To learn the use of annotating and validating JSON documents.
- To understand the difference between JSON & XML and JSON with Java.

7.2 INTRODUCTION

- JSON is an acronym for JavaScript Object Notation, is an open standard format, which is lightweight and text-based, designed explicitly for human-readable data interchange.
- It is a language-independent data format.
- It supports almost every kind of language, framework, and library.
- JSON is an open standard for exchanging data on the web. It supports data structures like objects and arrays. So, it is easy to write and read data from JSON.
- In JSON, data is represented in key-value pairs and curly braces hold objects, where a colon is followed after each name. The comma is used to separate key-value pairs. Square brackets are used to hold arrays, where each value is comma separated.

7.3 OVERVIEW

- JSON stands for JavaScript Object Notation.
- JSON is an open standard data-interchange format.
- JSON is lightweight and self-describing.
- JSON originated from JavaScript.
- JSON is easy to read and write.
- JSON is language independent.
- JSON supports data structures such as arrays and objects.

Features of JSON

- Simplicity
- Openness
- Self-Describing
- Internationalization
- Extensibility
- Interoperability

Advantages of JSON:

- **Less Verbose**: In contrast to XML, JSON follows a compact style to improve its users' readability. While working with a complex system, JSON tends to make substantial enhancements.
- **Faster**: The JSON parsing process is faster than that of the XML because the DOM manipulation library in XML requires extra memory for handling large XML files. However, JSON requires less data that ultimately results in reducing the cost and increasing the parsing speed.
- **Readable**: The JSON structure is easily readable and straightforward. Regardless of the programming language that you are using, you can easily map the domain objects.
- **Structured Data**: In JSON, a map data structure is used, whereas XML follows a tree structure. The key-value pairs limit the task but facilitate the predictive and easily understandable model.

7.4 SYNTAX

- JSON structure is based on the JavaScript object literal syntax; they share a number of similarities.

- The core elements of JSON syntax:
  - Data is presented in key/value pairs.
  - Data elements are separated by commas.
  - Curly brackets {} determine objects.
  - Square brackets [] designate arrays.
- As a result, JSON object literal syntax looks like this:
  {“key”:“value”,“key”:“value”,“key”:“value”.}

Example:

```
{
  "book": [
    {
      "id": "01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt"
    },
    {
      "id": "07",
      "language": "C++",
      "edition": "second",
      "author": "E.Balagurusamy"
    }
  ]
}
```


7.5 DATATYPES

JSON supports the following data types

Sr.No.	Type & Description
1	**Number** double- precision floating-point format in JavaScript
2	**String** double-quoted Unicode with backslash escaping
3	**Boolean** true or false
4	**Array** an ordered sequence of values

Sr.No.	Type & Description
5	**Value** it can be a string, a number, true or false, null etc
6	**Object** an unordered collection of key:value pairs
7	**Whitespace** can be used between any pair of tokens
8	**null** empty

1. Number

- It is a double precision floating-point format in JavaScript and it depends on implementation.
- Octal and hexadecimal formats are not used.
- No NaN or Infinity is used in Number.

The following table shows the number types −

Sr.No.	Type & Description
1	**Integer** Digits 1-9, 0 and positive or negative
2	**Fraction** Fractions like .3, .9
3	**Exponent** Exponent like e, e+, e-, E, E+, E-

Syntax & Example:

var json-object-name = { string : number_value, ...}

var obj = {marks: 97}

2. String

- It is a sequence of zero or more double quoted Unicode characters with backslash escaping.
- Character is a single character string i.e. a string with length 1.

Sr.No.	Type & Description
1	**"** double quotation
2	\ backslash
3	/ forward slash

Sr.No.	Type & Description
4	**b** backspace
5	**f** form feed
6	**n** new line
7	**r** carriage return
8	**t** horizontal tab
9	**u** four hexadecimal digits

Syntax & Example

var json-object-name = { string : "string value", ...}

var obj = {name: 'Amit'}

3. Boolean

- It includes true or false values.

Syntax & Example

var json-object-name = { string : true/false, ...}

var obj = {name: 'Amit', marks: 97, distinction: true}

4. Array

- It is an ordered collection of values.
- These are enclosed in square brackets which means that array begins with .[. and ends with .]..
- The values are separated by , (comma).
- Array indexing can be started at 0 or 1.
- Arrays should be used when the key names are sequential integers.

Syntax & Example

[value, ...]

```
{
  "books": [
        { "language":"Java" , "edition":"second" },
        { "language":"C++" , "lastName":"fifth" },
        { "language":"C" , "lastName":"third" }
  ]
}
```

5. Object

- It is an unordered set of name/value pairs.
- Objects are enclosed in curly braces, that is, it starts with '{' and ends with '}'.
- Each name is followed by ':'(colon) and the key/value pairs are separated by , (comma).
- The keys must be strings and should be different from each other.
- Objects should be used when the key names are arbitrary strings.

Syntax & Example:

{ string : value, ...

```
{
  "id": "011A",
  "language": "JAVA",
  "price": 500,
}
```

6. Whitespace

- It can be inserted between any pair of tokens. It can be added to make a code more readable. Example shows declaration with and without whitespace –

  Syntax & Example:

  {string:" ",....}

```
var obj1 = {"name": "Sachin Tendulkar"}
var obj2 = {"name": "SauravGanguly"}
```

7. Null

- It means empty type.

  Syntax & Example:

  Null

```
var i = null
if(i == 1) {
   document.write("<h1>value is 1</h1>");
}
else
{
 document.write("<h1>value is null</h1>");
}
```

8. Value

- It includes
  - number (integer or floating point)
  - String
  - Boolean
  - Array
  - Object
  - Null

Syntax & Example

String | Number | Object | Array | TRUE | FALSE | NULL

```
var i = 1;

var j = "sachin";

var k = null;
```

7.6 OBJECTS

- JSON objects can be created with JavaScript. Let us see the various ways of creating JSON objects using JavaScript.
- Creation of an empty Object

  var JSONObj = {};
- Creation of a new Object

  var JSONObj = new Object();
- Creation of an object with attribute bookname with value in string, attribute price with numeric value. Attributes are accessed by using '.' Operator −

  var JSONObj = { "bookname ":"VB BLACK BOOK", "price":500 };

Example that shows creation of an object in javascript using JSON, save the below code as json_object.htm

```
<html>
  <head>
      <title>Creating Object JSON with JavaScript</title>
      <script language = "javascript" >
      var JSONObj = { "name" : "mumbaiuniversity.com", "year"  :
2005 };
```

```
        document.write("<h1>JSON with JavaScript example</h1>");
        document.write("<br>");
        document.write("<h3>Website Name =
"+JSONObj.name+"</h3>");
        document.write("<h3>Year = "+JSONObj.year+"</h3>");
        </script>
   </head>
 <body>
   </body>
</html>
```

7.7 SCHEMA

- JSON Schema is a specification for JSON based format for defining the structure of JSON data.
- Describe your existing data format.
    - Clear, human- and machine-readable documentation.
    - Complete structural validation, useful for automated testing.
    - Complete structural validation, validating client-submitted data.
- JSON Schema Validation Libraries

  There are several validators currently available for different programming languages. Currently the most complete and compliant JSON Schema validator available is JSV.

Languages	Libraries
C	WJElement (LGPLv3)
Java	json-schema-validator (LGPLv3)
.NET	Json.NET (MIT)
ActionScript 3	Frigga (MIT)
Haskell	aeson-schema (MIT)
Python	Jsonschema
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
PHP	php-json-schema (MIT). json-schema (Berkeley)
JavaScript	Orderly (BSD); JSV; json-schema; Matic (MIT); Dojo; Persevere (modified BSD or AFL 2.0); schema.js.

Example

Given below is a basic JSON schema, which covers a classical product catalog description −

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {

      "id": {
      "description": "The unique identifier for a product",
      "type": "integer"
      },

      "name": {
      "description": "Name of the product",
      "type": "string"
      },

      "price": {
         "type": "number",
      "minimum": 0,
      "exclusiveMinimum": true
      }
  },

  "required": ["id", "name", "price"]
}
```

- Various important keywords that can be used in this schema −

Sr.No.	Keyword & Description
1	**$schema** The $schema keyword states that this schema is written according to the draft v4 specification.
2	**title** You will use this to give a title to your schema.
3	**description** A little description of the schema.
4	**type** The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
5	**properties** Defines various keys and their value types, minimum and maximum values to be used in JSON file.
6	**required**

Sr.No.	Keyword & Description
	This keeps a list of required properties.
7	**minimum** This is the constraint to be put on the value and represents minimum acceptable value.
8	**exclusiveMinimum** If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".
9	**maximum** This is the constraint to be put on the value and represents maximum acceptable value.
10	**exclusiveMaximum** If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".
11	**multipleOf** A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
12	**maxLength** The length of a string instance is defined as the maximum number of its characters.
13	**minLength** The length of a string instance is defined as the minimum number of its characters.
14	**pattern** A string instance is considered valid if the regular expression matches the instance successfully.

7.8 COMPARISON WITH XML

Similarities between the JSON and XML.

- **Self-describing:** Both json and xml are self-describing as both xml data and json data are human-readable text.
- **Hierarchical:** Both json and xml support hierarchical structure. Here hierarchical means that the values within values.
- **Data interchange format:** JSON and XML can be used as data interchange formats by many different programming languages.
- **Parse:** Both the formats can be easily parsed.
- **Retrieve:** Both formats can be retrieved by using HTTP requests. The methods used for retrieving the data are GET, PUT, POST.

Differences between the JSON and XML

JSON	XML
JSON stands for javascript object notation.	XML stands for an extensible markup language.
The extension of json file is .json.	The extension of xml file is .xml.
The internet media type is application/json.	The internet media type is application/xml or text/xml.
The type of format in JSON is data interchange.	The type of format in XML is a markup language.
It is extended from javascript.	It is extended from SGML.
It is open source means that we do not have to pay anything to use JSON.	It is also open source.
The object created in JSON has some type.	XML data does not have any type.
The data types supported by JSON are strings, numbers, Booleans, null, array.	XML data is in a string format.
It does not have any capacity to display the data.	XML is a markup language, so it has the capacity to display the content.
JSON has no tags.	XML data is represented in tags, i.e., start tag and end tag.
	XML file is larger. If we want to represent the data in XML then it would create a larger file as compared to JSON.
JSON is quicker to read and write.	XML file takes time to read and write because the learning curve is higher.
JSON can use arrays to represent the data.	XML does not contain the concept of arrays.
It can be parsed by a standard javascript function. It has to be parsed before use.	XML data which is used to interchange the data, must be parsed with respective to their programming language to use that.
It can be easily parsed and little bit code is required to parse the data.	It is difficult to parse.
File size is smaller as compared to XML.	File size is larger.

JSON	XML
JSON is data-oriented.	XML is document-oriented.
It is less secure than XML.	It is more secure than JSON.

7.9 JSON WITH JAVA

- Encode and decode JSON objects using Java programming language.

Environment

- Install any of the JSON modules available.
- For this download and installed JSON.simple and have added the location of **json-simple-1.1.1.jar** file to the environment variable CLASSPATH.

Mapping between JSON and Java entities

- JSON.simple maps entities from the left side to the right side while decoding or parsing, and maps entities from the right to the left while encoding.

JSON	Java
string	java.lang.String
number	java.lang.Number
true|false	java.lang.Boolean
null	null
array	java.util.List
object	java.util.Map

Encoding JSON in Java

- To encode a JSON object using Java JSONObject which is a subclass of java.util.HashMap. No ordering is provided. If you need the strict ordering of elements, use JSONValue.toJSONString (map) method with ordered map implementation such as java.util.LinkedHashMap.

```
import org.json.simple.JSONObject;
 class JsonEncodeDemo {
   public static void main(String[] args) {
        JSONObject obj = new JSONObject();

       obj.put("name", "foo");
       obj.put("num", new Integer(100));
       obj.put("balance", new Double(1000.21));
```

```
        obj.put("is_vip", new Boolean(true));

         System.out.print(obj);
   }
}
Output:
{"balance": 1000.21, "num":100, "is_vip":true, "name":"foo"}
```

Decoding JSON in Java

- The following example makes use of **JSONObject** and **JSONArray** where JSONObject is a java.util.Map and JSONArray is a java.util.List, so you can access them with standard operations of Map or List.

```
import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.parser.ParseException;
import org.json.simple.parser.JSONParser;

class JsonDecodeDemo {

   public static void main(String[] args) {

         JSONParser parser = new JSONParser();
         String s =
"[0,{\"1\":{\"2\":{\"3\":{\"4\":[5,{\"6\":7}]}}}}]";

         try{
         Object obj = parser.parse(s);
           JSONArray array = (JSONArray)obj;

         System.out.println("The 2nd element of array");
         System.out.println(array.get(1));
         System.out.println();
         JSONObject obj2 = (JSONObject)array.get(1);
         System.out.println("Field \"1\"");
         System.out.println(obj2.get("1"));
        s = "{}";
        obj = parser.parse(s);
         System.out.println(obj);
        s = "[5,]";
```

```
        obj = parser.parse(s);
         System.out.println(obj);
        s = "[5,,2]";
        obj = parser.parse(s);
         System.out.println(obj);
         }catch(ParseException pe) {
         System.out.println("position: " + pe.getPosition());
         System.out.println(pe);
         }
   }
}
Output:
The 2nd element of array
{"1":{"2":{"3":{"4":[5,{"6":7}]}}}}
Field "1"
{"2":{"3":{"4":[5,{"6":7}]}}}
{}
[5]
[5,2]
```

7.10 SUMMARY

1. JSON is an acronym for JavaScript Object Notation, is an open standard format, which is lightweight and text-based, designed explicitly for human-readable data interchange
2. JSON stands for JavaScript Object Notation.
3. JSON is language independent.
4. JSON structure is based on the JavaScript object literal syntax, they share a number of similarities.
5. JSON objects can be created with JavaScript
6. JSON Schema is a specification for JSON based format for defining the structure of JSON data.

7.11 REFERENCE FOR FURTHER READING

1. Herbert Schildt, Java2: The Complete Reference, Tata McGraw-Hill,5th Edition
2. Joe Wigglesworth and Paula McMillan, Java Programming: Advanced Topics, Thomson Course Technology (SPD) , 3rd Edition

7.12 UNIT END EXERCISES

1. Mention what is the rule for JSON syntax rules? Give an example of JSON object?
2. Explain the difference between JSON & XML.
3. Explain the difference between JSON & Java.
4. What are the data types supported by JSON?
5. What are the advantages of JSON?
6. Explain the role of JSON object in detail.
7. Explain what do you mean by JSON Schema? Explain it with suitable example?
8. Explain the steps involved in executing JSON with java.

❄❄❄❄❄❄❄