.

SYBSC SEMESTER IV **ADVANCED JAVA QUESTION BANK**
UNIT I

1.Difference between Statement,PreparedStatement and CallableStatement Interface in JDBC.

Answer:

Statement	PreparedStatement	CallableStatement
It is used to execute normal SQl queries.	It is used to execute parameterized or dynamic SQL queries.	It is used to call the stored procedures.
Preferred when a SQL query has to be executed only once.	Preferred when a particular SQL query has to be executed multiple times with different values.	It is preferred when stored procedures need to be executed.
You cannot pass parameters to this type of query.	You can pass parameter to this type of query at run-time.	You can pass 3 types of parameters using this interface: IN, OUT and INOUT.
This interface is mainly used to execute DDL statements like CREATE,ALTER,DROP,etc.	This interface is used for any kind of SQL query which needs to be executed multiple times.	It is used to execute stored procedures and functions.
The performance of this interface is very poor.	The performance of this interface is better as compared to the Statement interface when executing the same query multiple times.	The performance of this interface is high.

1. **Design a swing program for generating a login form.**

Answer:

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class LibrarianLoginForm extends JFrame {
   JLabel label1;
   JLabel label2;
   JLabel label3;
   JButton button1;
   JTextField textfield1;
   JPasswordField passwordfield1;
```

```
    LibrarianLoginForm() {
        label1 = new JLabel();
        label1.setBounds(50,25,500,50);+
        label1.setText("Librarian Login Form");
        add(label1);

        label2 = new JLabel();
        label2.setBounds(25,75,100,25);
        label2.setText("Enter Name:");
        add(label2);

        textfield1 = new JTextField();
        textfield1.setBounds(130,75,100,25);
        add(textfield1);

        label3 = new JLabel();
        label3.setBounds(25,125,100,25);
        label3.setText("Enter Password:");
        add(label3);

        passwordfield1 = new JPasswordField();
        passwordfield1.setBounds(130,125,100,25);
        add(passwordfield1);

        button1 = new JButton();
        button1.setBounds(77,175,100,25);
        button1.setText("Login");
        add(button1);

        this.setBounds(50,50,300,275);
        this.setLayout(null);
        this.setVisible(true);
    }

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

2. Design a Radio button in swing for selecting one value from male and female?

Answer:

```
1. import javax.swing.*;
2. public class RadioButtonExample {
3. RadioButtonExample(){
4. f=new JFrame();
5. JRadioButton r1=new JRadioButton("A) Male");
6. JRadioButton r2=new JRadioButton("B) Female");
7. r1.setBounds(75,50,100,30);
8. r2.setBounds(75,100,100,30);
9. ButtonGroup bg=new ButtonGroup();
10. bg.add(r1);bg.add(r2);
11. f.add(r1);f.add(r2);
12. f.setSize(300,300);
13. f.setLayout(null);
14. f.setVisible(true);
15. }
16. public static void main(String[] args) {
17.     new RadioButtonExample();
18. }
19. }
```

3. Explain Panes in Swings? Why they are required?

Answer:

- Swing defined two types of containers. The first are top-level containers: JFrame, JApplet, JWindow and JDialog. These containers do not inherit JComponent and are pretty heavyweight compared to other Swing components.
- Each top-level container defines a set of panes. At the top of the hierarchy is the JRootPane. This is a lightweight container whose purpose is to manage other panes. It also helps to manage the optional menu bar. The panes that comprise the root pane are the glass pane, the content pane and the layered pane.

- The glass pane is the top-level pane. It sits above and covers all other panes. By default, it is a transparent instance of JPanel. The glass pane enables you to manage mouse events that affect the entire container or to print over any other component.
- The layered pane is an instance of JLayeredPane. It allows components to be given a depth value. This value determines which component overlays another.. It holds the content pane and the optional menu bar.
- The pane with which your application interacts the most is the component pane, because this is the pane to which you will add visual components. When you add a component to a top-level container, you will add it to the content pane. The content pane is by default, an opaque instance of JPanel.

4. Methods used for navigation through database records using a ResultSet object.

Answer:

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.

5. Why MVC is the important feature of SWING over AWT.

Answer:

The MVC (Model-View-Controller) architecture is successful because each piece of the design corresponds to an aspect of a component. In MVC technology, the *model* corresponds to the state of the component. The *view* corresponds to how the component is displayed on the screen. The *controller* determines how the component reacts by changing the model to reflect the user's choice, this then results in the view being updated. By separating the component into a model, a view and a controller, the specific implementation of each can be changed without affecting the other two. For example, different view implementations can render the same component in different ways without affecting the model or the controller.

Although the MVC architecture and the principles behind it are conceptually sound, the high level of separation between the view and the controller is not beneficial for SWING components. Instead, SWING uses a modified version of MVC which combines the view and the controller into a single logical entity called the *UI delegate*. For this reason, SWING's architecture is called the *Model-Delegate* architecture of the *Separable-Model* architecture. SWING's Model-Delegate architecture gives it a pluggable look and feel. Because the view (look) and controller (feel) are separate from the model, the look and feel can be changed without affecting the way the component is used in the program. Conversely, it is possible to change how the component is used in the program without affecting how it is displayed on the screen.

6. Write JDBC steps for connecting a java program to a 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 ClassNotFoundException

Example to register the OracleDriver class:

Here, Java program is loading oracle driver to establish 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 SQLException

2) public static Connection getConnection(String url,String name,String password) throws SQLException

Example to establish connection with the MySql database

```
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/test",”root","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");

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();

7. Explain Swing components.(from syllabus)

Answer:

JLabel	JLabel is used to display text and/or an icon. *Constructors:* JLabel(Icon *icon*) JLabel(String *str*) JLabel(String *str*, Icon *icon*, int *align*) *Methods:* The icon and text can be obtained by the following methods: Icon getIcon() String getText() The icon and text can be set by the following methods: void setIcon(Icon *icon*) void setText(String *str*)
JTextField	JTextField allows you to edit one line of text.

	Constructors: JTextField(int *cols*) JTextField(String *str*, int *cols*) JTextField(String *str*) *Method:* To obtain the text currently in the text field: String getText()
JPasswordField	JPasswordField is a component that allows editing of a single line of text where the view indicates that something was typed by does not show the actual characters. *Constructors:* JPasswordFields(int *cols*) JPasswordFields(String *str*, int *cols*) JPasswordFields(String *str*) *Methods:* char getEchoChar() void setEchoChar(Char *c*) String getPassword() String getText()
JTextArea	SWING provides the JTextArea class to allow the editing of multiple line text. *Constructors:* JTextArea(String *str*) JTextArea(int *rows*, int *cols*) JTextArea(String *str*, int *rows*, int *cols*) *Methods:* void setRows(int *rows*) void setColumns(int *cols*) void setFont(Font *f*) void insert(String *str*, int *position*) void append(String *str*)
JButton	JButton provides the functionalist of a push button. It can be associated with an icon, a string or both. *Constructors:* JButton(Icon *icon*) JButton(String *str*, Icon *icon*) JButton(String *str*) *Methods:* You can obtain the action command on the event object using:

	String getActionCommand()
JCheckBox	JCheckBox provides the functionality of a simple check box. *Constructor:* JCheckBox(String *str*) JCheckBox(String *str*, boolean *selected*) *Methods:* String getText() - to retrieve the textual content of a checkbox boolean isSelected() - to retrieve a boolean value that tells us whether the checkbox has been selected or not
JRadioButton	Radio Buttons are a group of mutually exclusive buttons, in which only one button can be selected at any one time. In order for their mutually exclusive nature to be activated, radio buttons must be added configured into a group. A button group is created by the ButtonGroup class, elements are added to the ButtonGroup by calling the method: void add(AbstractButton *ab*), here *ab* is the reference to the Radio Button to be added to a group. *Constructors:* JRadioButton(String *str*) JRadioButton(String *str*, boolean *selected*) *Methods:* To get/set the icon or the text of a radio button: void setText(String *str*) void setIcon(Icon *icon*) String getText() Icon getIcon() To enable/disable the button: void setEnabled(Boolean *b*)
JComboBox	SWING provides a combination of a text-field and a drop-down list through the JComboBox class. It normally displays one entry, but it will also display a drop-down list that allows users to select a different entry. *Constructor:* JComboBox(E[] *items*) - here E represents the type of items in the combo box. *Methods:* void addItem(E *obj*) - items can be dynamically added to the list using this method. Object getSelectedItem() - returns a reference to the selected entry from the drop-down list. You will need to cast the returned

	value to the type of element stored in the list
JList	In SWING, the basic list is implemented using the JList class. It supports the selection of one or more items from a list. *Constructor:* JList(E[] *items*) - this creates a JList containing the items in the array specified by *items*, and E represents the type of items in the list. *Methods:* void setSelectionMode(int *mode*) - here *mode* can take one of three values: ● SINGLE_SELECTION - allows the user to select only ONE item from the list. ● SINGLE_INTERVAL_SELECTION - allows the user to select one range of values from the list. ● MULTIPLE_INTERVAL_SELECTION - allows the user to select multiple ranges of items from the list. int getSelectedIndex() - returns the index of the first item selected. E getSelectedValue() - returns a reference to the first selected value instead of its index, if no value has been selected it returns **null**.

8. WAP to store a file in database.

Answer:

- First, we establish a connection between MySQL database and JAVA file with the help of various APIs, interfaces and methods.
- Then, we create an instance of the File class and then implement saving of the file in the database.
    - *Connection* : This interface specifies connection with specific databases like: MySQL, Ms-Access, Oracle etc and java files. The SQL statements are executed within the context of this interface.
    - *Class.forName(String driver)* : It loads the driver.
    - *DriverManager* : This class controls a set of JDBC drivers. Each driver has to be register with this class.
    - *getConnection(String url, String userName, String password)* : This method establishes a connection to specified database url. It is having three arguments:
        - url - Database url where stored or created your database
        - username - Username of MySQL
        - password - Password of MySQL
    - *PreparedStatement* : This interface is slightly more powerful version of Statement which is used for executing the SQL statement.

- Here is the code:

```
import java.io.*;
import java.sql.*;
public class FileToDatabase {
        public static void main(String[] args) throws Exception {
                String fileName = "C:/input.txt";
                File file = new File(fileName);
                FileInputStream fis = new FileInputStream(file);
                Class.forName("com.mysql.jdbc.Driver");
                Connection conn = DriverManager.getConnection(
                                "jdbc:mysql://localhost:3306/test", "root", "root");
                PreparedStatement pstmt = conn
                                .prepareStatement("insert into file(file, file_data) values (?, ?)");
                pstmt.setString(1, file.getName());
                pstmt.setBinaryStream(2, fis, (int) file.length());
                pstmt.executeUpdate();,
        }
}
```

9. Design a Simple Calculator using Swing.

Ans :

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


class Calc implements ActionListener
{
       JFrame f;
       JTextField t;
       JButton b1,b2,b3,b4,b5,b6,b7,b8,b9,b0,bdiv,bmul,bsub,badd,bdec,beq,bdel,bclr;

       static double a=0,b=0,result=0;
       static int operator=0;

       Calc()
       {
              f=new JFrame("Calculator");
              t=new JTextField();
              b1=new JButton("1");
              b2=new JButton("2");
              b3=new JButton("3");
              b4=new JButton("4");
              b5=new JButton("5");
              b6=new JButton("6");
              b7=new JButton("7");
              b8=new JButton("8");
              b9=new JButton("9");
```

```
b0=new JButton("0");
bdiv=new JButton("/");
bmul=new JButton("*");
bsub=new JButton("-");
badd=new JButton("+");
bdec=new JButton(".");
beq=new JButton("=");
bdel=new JButton("Delete");
bclr=new JButton("Clear");

t.setBounds(30,40,280,30);
b7.setBounds(40,100,50,40);
b8.setBounds(110,100,50,40);
b9.setBounds(180,100,50,40);
bdiv.setBounds(250,100,50,40);

b4.setBounds(40,170,50,40);
b5.setBounds(110,170,50,40);
b6.setBounds(180,170,50,40);
bmul.setBounds(250,170,50,40);

b1.setBounds(40,240,50,40);
b2.setBounds(110,240,50,40);
b3.setBounds(180,240,50,40);
bsub.setBounds(250,240,50,40);

bdec.setBounds(40,310,50,40);
b0.setBounds(110,310,50,40);
beq.setBounds(180,310,50,40);
badd.setBounds(250,310,50,40);

bdel.setBounds(60,380,100,40);
bclr.setBounds(180,380,100,40);

f.add(t);
f.add(b7);
f.add(b8);
f.add(b9);
f.add(bdiv);
f.add(b4);
f.add(b5);
f.add(b6);
f.add(bmul);
f.add(b1);
f.add(b2);
f.add(b3);
f.add(bsub);
f.add(bdec);
f.add(b0);
f.add(beq);
f.add(badd);
f.add(bdel);
f.add(bclr);

f.setLayout(null);
```

```
            f.setVisible(true);
            f.setSize(350,500);
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.setResizable(false);

            b1.addActionListener(this);
            b2.addActionListener(this);
            b3.addActionListener(this);
            b4.addActionListener(this);
            b5.addActionListener(this);
            b6.addActionListener(this);
            b7.addActionListener(this);
            b8.addActionListener(this);
            b9.addActionListener(this);
            b0.addActionListener(this);
            badd.addActionListener(this);
            bdiv.addActionListener(this);
            bmul.addActionListener(this);
            bsub.addActionListener(this);
            bdec.addActionListener(this);
            beq.addActionListener(this);
            bdel.addActionListener(this);
            bclr.addActionListener(this);
      }

      public void actionPerformed(ActionEvent e)
      {
            if(e.getSource()==b1)
                  t.setText(t.getText().concat("1"));

            if(e.getSource()==b2)
                  t.setText(t.getText().concat("2"));

            if(e.getSource()==b3)
                  t.setText(t.getText().concat("3"));

            if(e.getSource()==b4)
                  t.setText(t.getText().concat("4"));

            if(e.getSource()==b5)
                  t.setText(t.getText().concat("5"));

            if(e.getSource()==b6)
                  t.setText(t.getText().concat("6"));

            if(e.getSource()==b7)
                  t.setText(t.getText().concat("7"));

            if(e.getSource()==b8)
                  t.setText(t.getText().concat("8"));

            if(e.getSource()==b9)
                  t.setText(t.getText().concat("9"));

            if(e.getSource()==b0)
```

```
                t.setText(t.getText().concat("0"));

        if(e.getSource()==bdec)
                t.setText(t.getText().concat("."));

        if(e.getSource()==badd)
        {
                a=Double.parseDouble(t.getText());
                operator=1;
                t.setText("");
        }

        if(e.getSource()==bsub)
        {
                a=Double.parseDouble(t.getText());
                operator=2;
                t.setText("");
        }

        if(e.getSource()==bmul)
        {
                a=Double.parseDouble(t.getText());
                operator=3;
                t.setText("");
        }

        if(e.getSource()==bdiv)
        {
                a=Double.parseDouble(t.getText());
                operator=4;
                t.setText("");
        }

        if(e.getSource()==beq)
        {
                b=Double.parseDouble(t.getText());

                switch(operator)
                {
                        case 1: result=a+b;
                                break;

                        case 2: result=a-b;
                                break;

                        case 3: result=a*b;
                                break;

                        case 4: result=a/b;
                                break;

                        default: result=0;
                }

                t.setText(""+result);
```

```
            }

            if(e.getSource()==bclr)
                  t.setText("");

            if(e.getSource()==bdel)
            {
                  String s=t.getText();
                  t.setText("");
                  for(int i=0;i<s.length()-1;i++)
                  t.setText(t.getText()+s.charAt(i));
            }
      }

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

9.Explain JDBC Architecture in detail.

Ans:

The JDBC API supports both two-tier and three-tier processing models for database access. The DBMS-proprietary protocol provides two-way communication between the client machine and the database server.

In the two-tier model, a Java application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

10. Explain Forward only ResultSet with example.

Ans:

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object. JDBC provides the following Connection class methods to create statements with desired result set:

1. createStatement(int *RSType*, int *RSConcurrency*);
2. prepareStatement(String *SQL*, int *RSType*, int *RSConcurrency*);
3. prepareCall(String *sql*, int *RSType*, int *RSConcurrency*);

The first argument indicates the type of a ResultSet object and the second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable.

Type of ResultSet: a ResultSet can be of one of the following types:

1. ResultSet.TYPE_FORWARD_ONLY - The cursor can only move in the forward direction.
2. ResultSet.TYPE_SCROLL_INSENSITIVE - The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
3. ResultSet.TYPE_SCROLL_SENSITIVE - The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

11. WAP JDBC program to insert records in Book table using prepared Statement.

Ans :

```
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:@localhost:1521:xe","system","oracle");

PreparedStatement stmt=con.prepareStatement("insert into Book values(?,?,?)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Adv java");
stmt.setInt(3,30) ;

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

con.close();

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

}
}
```

12. Advantages of Swing over AWT.

Ans:

- Swing provides both additional components and added functionality to AWT-replacement components.
- Swing components can change their appearance based on the current "look and feel" library that's being used. You can use the same look and feel as the platform you're on, or use a different look and feel.
- Swing components follow the Model-View-Controller paradigm (MVC), and thus can provide a much more flexible UI.
- Swing provides "extras" for components, such as:
  - Icons on many components
  - Decorative borders for components
  - Tooltips for components
- Swing components are lightweight (less resource intensive than AWT)
- Swing provides built-in double buffering
- Swing provides paint debugging support for when you build your own components.

13. Explain drivers in JDBC..

Ans:

→ The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

JDBC technology drivers fit into one of four categories.[2]

1. JDBC-ODBC bridge
2. Native-API driver
3. Network-Protocol driver (Middleware driver)
4. Database-Protocol driver (Pure Java driver) or thin driver.

→ **TYPE 1 : (** JDBC-ODBC bridge**)**

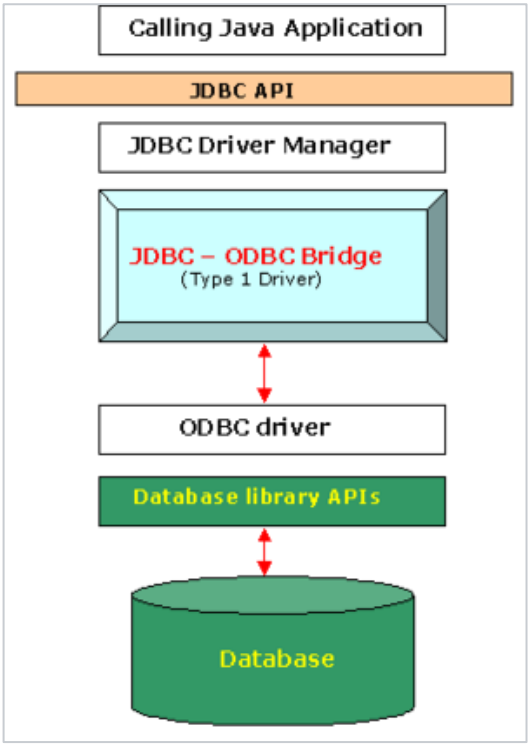
- The JDBC type 1 driver, also known as the **JDBC-ODBC bridge**, is a database driver implementation that employs the ODBC driver to connect to the database.
- The driver converts JDBC method calls into ODBC function calls.
- The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the underlying operating system the JVM is running upon.
- Also, use of this driver leads to other installation dependencies; for example, ODBC must be installed on the computer having the driver and the database must support an ODBC driver.
- If a driver has been written so that loading it causes an instance to be created and also calls `DriverManager.registerDriver` with that instance as the parameter (as it should do), then it is in the Driver Manager's list of drivers and available for creating a connection.
- It tests the drivers by calling the method Driver.connect on each one in turn, passing them the URL that the user originally passed to the method `DriverManager.getConnection`. The first driver that recognizes the URL makes the connection.

Advantages -

- Almost any database for which an ODBC driver is installed can be accessed, and data can be retrieved.

Disadvantages -

- Performance overhead since the calls have to go through the JDBC(java database connectivity) bridge to the ODBC(open database connectivity)driver, then to the native database connectivity interface (thus may be slower than other types of drivers).
- The ODBC driver needs to be installed on the client machine.
- Not suitable for applets, because the ODBC driver needs to be installed on the client.
- Specific ODBC drivers are not always available on all platforms; hence, portability of this driver is limited.
- No support from JDK 1.8 (Java 8) .



→ **TYPE 2 :(** Native-API driver**) -**

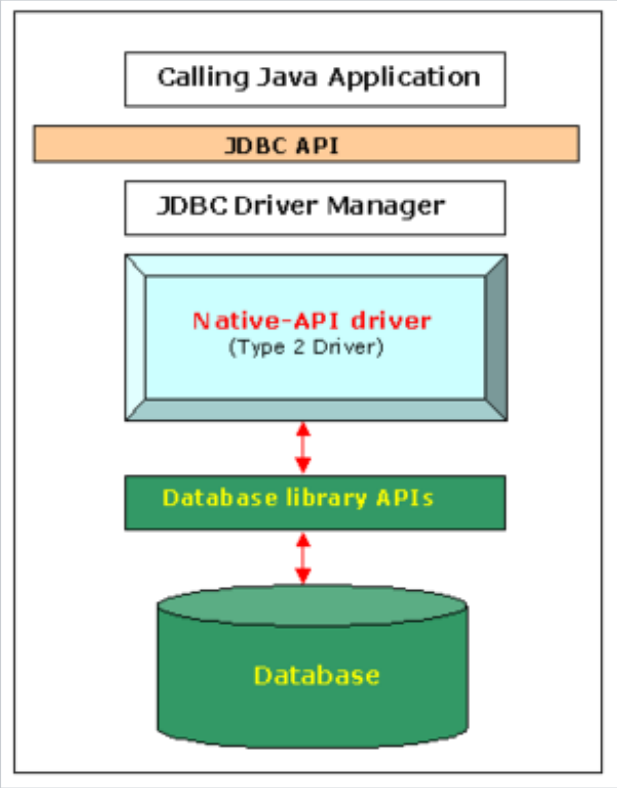
- The JDBC type 2 driver, also known as the **Native-API driver**, is a database driver implementation that uses the client-side libraries of the database.
- The driver converts JDBC method calls into native calls of the database API. For example: Oracle OCI driver is a type 2 driver.

Advantages

- As there is no implementation of JDBC-ODBC bridge, it may be considerably faster than a Type 1 driver.

Disadvantages

- The vendor client library needs to be installed on the client machine.
- Not all databases have a client-side library.
- This driver is platform dependent.
- This driver supports all Java applications except applets.



→ **TYPE 3** : (Network Protocol Driver (middleware driver)) -

- The JDBC type 3 driver, also known as the Pure Java driver for database middleware,[7] is a database driver implementation which makes use of a middle tier between the calling program and the database. The middle-tier (application server) converts JDBC calls directly or indirectly into a vendor-specific database protocol.

- This differs from the type 4 driver in that the protocol conversion logic resides not at the client, but in the middle-tier.

Functions

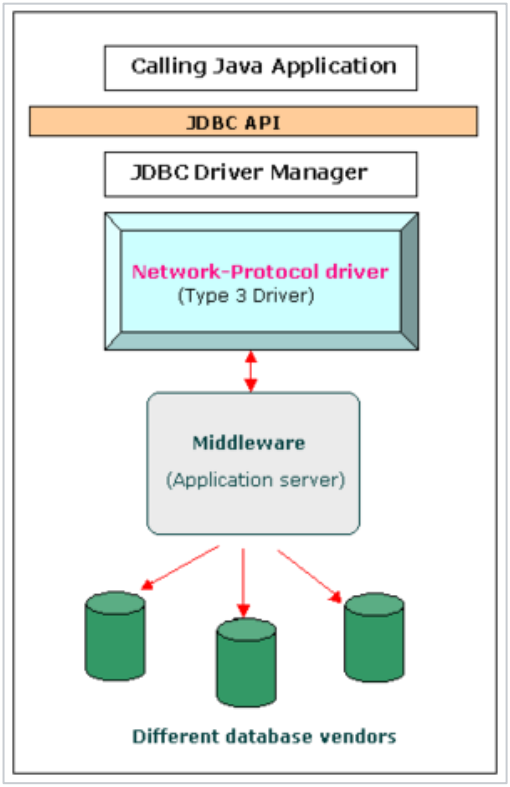
- Sends JDBC API calls to a middle-tier net server that translates the calls into the DBMS-specific network protocol.The translated calls are then sent to a particular DBMS.
- Follows a three-tier communication approach.
- Can interface to multiple databases – Not vendor specific.
- The JDBC Client driver written in java, communicates with a middleware-net-server using a database independent protocol, and then this net server translates this request into database commands for that database.
- Thus the client driver to middleware communication is database independent.

Advantages -

- Since the communication between client and the middleware server is database independent, there is no need for the database vendor library on the client. The client need not be changed for a new database.
- The middleware server can provide typical middleware services like caching (of connections, query results, etc.), load balancing, logging, and auditing.
- A single driver can handle any database, provided the middleware supports it.
- E.g.: IDA Server

Disadvantages -

- Requires database-specific coding to be done in the middle tier.
- The middleware layer added may result in additional latency, but is typically overcome by using better middleware services.



→ TYPE 4 :(Database-Protocol driver/Thin Driver(Pure Java driver)) -

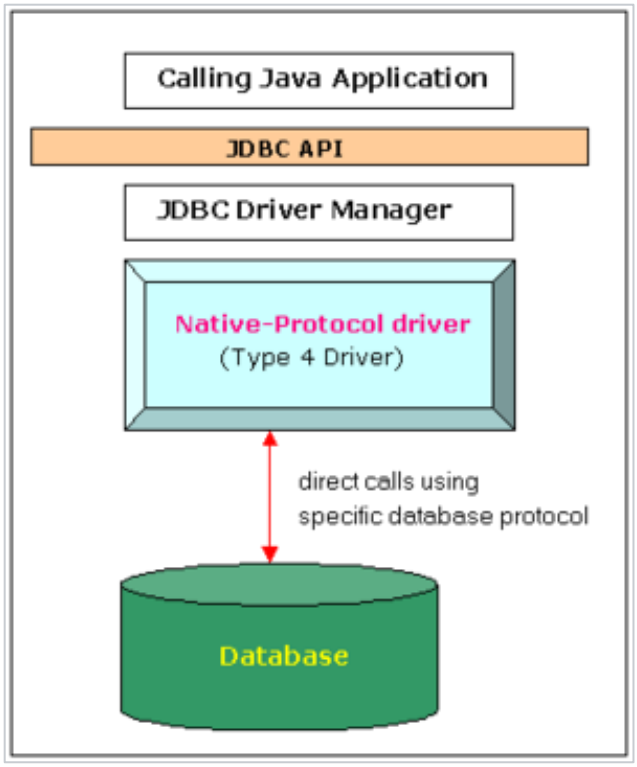
- The JDBC type 4 driver, also known as the Direct to Database **Pure Java Driver**, is a database driver implementation that converts JDBC calls directly into a vendor-specific database protocol.
- Written completely in Java, type 4 drivers are thus platform independent. They install inside the Java Virtual Machine of the client. This provides better performance than the type 1 and type 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls.
- As the database protocol is vendor specific, the JDBC client requires separate drivers, usually vendor supplied, to connect to different types of databases.

Advantages -

- Completely implemented in Java to achieve platform independence.
- These drivers don't translate the requests into an intermediary format (such as ODBC).
- The client application connects directly to the database server. No translation or middleware layers are used, improving performance.
- The JVM can manage all aspects of the application-to-database connection; this can facilitate debugging.

Disadvantages -

- Drivers are database specific, as different database vendors use widely different (and usually proprietary) network protocols.



UNIT 2

1. WAP to demonstrate Life Cycle of a servlet.

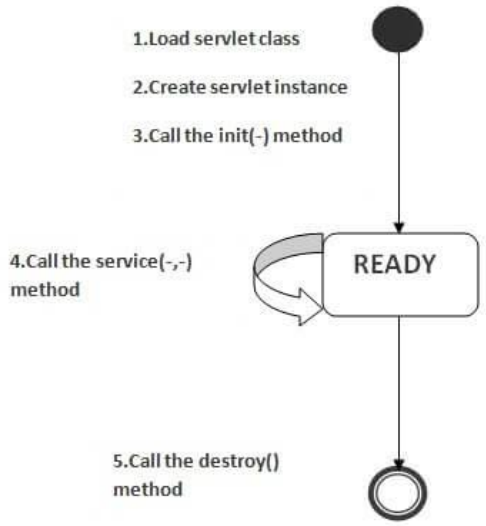
Ans :

```
// Java program to demonstrate lifecycle of a servlet
// Importing required Java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class AdvanceJavaConcepts extends HttpServlet
{
private String output;

// Initializing servlet
public void init() throws ServletException
{
        output = "Advance Java Concepts";
}

// Requesting and printing the output
public void doGet(HttpServletRequest req,
                                        HttpServletResponse resp)
        throws ServletException, IOException
        {
                resp.setContentType("text/html");
                PrintWriter out = resp.getWriter();
                out.println(output);
        }

        // Finalization code (End of the servlet life cycle)
        public void destroy()
        {
                System.out.println("Over");
        }
}
```

/."

→ 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.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



→ As displayed in the above diagram, there are three states of a servlet: new, ready and end. → The servlet is in new state if servlet instance is created.

→ After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks.

→ When the web container invokes the destroy() method, it shifts to the end state.

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:

```
1. 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:

```
1. public void service(ServletRequest request, ServletResponse response)
2.   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:

```
1. public void destroy()
```

2. Write a program to demonstrate servletconfig interface.

Ans : DemoServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class DemoServlet extends HttpServlet {
6. public void doGet(HttpServletRequest request, HttpServletResponse response)
7.   throws ServletException, IOException {
```

```
8.
9.      response.setContentType("text/html");
10.     PrintWriter out = response.getWriter();
11.
12.     ServletConfig config=getServletConfig();
13.     String driver=config.getInitParameter("driver");
14.     out.print("Driver is: "+driver);
15.
16.     out.close();
17.     }
18.
19.}
```

web.xml

```
1.  <web-app>
2.
3.  <servlet>
4.  <servlet-name>DemoServlet</servlet-name>
5.  <servlet-class>DemoServlet</servlet-class>
6.
7.  <init-param>
8.  <param-name>driver</param-name>
9.  <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
10.</init-param>
11.
12.</servlet>
13.
14.<servlet-mapping>
15.<servlet-name>DemoServlet</servlet-name>
16.<url-pattern>/servlet1</url-pattern>
17.</servlet-mapping>
18.
```

19. </web-app>

3. **Differentiate between servlet and JSP.**

Answer:

JSP	Servlet
JSP is protocol dependent it can handle only HTTP and HTTPS protocol.	Servlet is Protocol independent it can handle any type of protocol i.e FTP, HTTP
Time taken to generate response for first request is more	Time taken to generate response for first request is less
Business logic kept separate from presentation logic.	Business logic tightly coupled with presentation logic.
JSP is a scripting language which can generate dynamic response.	Servlet is a java Program which can generate dynamic response.
It's easier to code in JSP than in Servlets.	Its little much code to write in Servlet than JSP
In MVC, JSP act as a view.	In MVC Servlet act as a Controller.
Implicit object is available in JSP. i.e. request , response , session	There is no implicit object available we have to create it

4. **Write a servlet program to manage session using Httpsession.**

Ans:

5. **Explain any 5 implicit JSP objects.**

Ans :

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

PrintWriter out=response.getWriter();

Example of out implicit object

In this example we are simply displaying date and time.

index.jsp

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

2) JSP request object : The JSP request is an implicit object of **type HttpServletRequest i.e. created for each jsp request** by the web container. **It can be used to get request information such as parameter, header information, remote address, server name**, server port, content type, character encoding etc.

→ It can also be used to set, get and remove attributes from the jsp request scope.

Example of JSP request implicit object

index.html

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

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

3) JSP response implicit object : In JSP, **response is an implicit object of type HttpServletResponse**. The instance of HttpServletResponse is created by the web container for each jsp request.

→ It can be **used to add or manipulate response such as redirect response to another resource**, send error etc.

Example of response implicit object

index.html

```
1. <form action="welcome.jsp">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>
```

welcome.jsp

```
1. <%
2. response.sendRedirect("http://www.google.com");
3. %>
```

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

→ Generally, it is used to get initialization parameter from the web.xml file.

Example of config implicit object:

index.html

```
1. <form action="welcome">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>
```

web.xml file

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <jsp-file>/welcome.jsp</jsp-file>
6.
7. <init-param>
```

```
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));

String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

5) JSP application implicit object : In JSP, **application is an implicit object of type ServletContext**.

→ The instance of ServletContext is **created only once by the web container when application or project is deployed on the server**.

→ This object can be **used to get initialization parameter from configuration file** (web.xml). It can also be **used to get, set or remove attribute** from the application scope.

→ This initialization **parameter can be used by all jsp pages**.

6) session implicit object : In JSP, session is an **implicit object of type HttpSession**.The Java developer can **use this object to set,get or remove attribute or to get session information**.

7) pageContext implicit object : In JSP, pageContext is an implicit **object of type PageContext class**.The pageContext object can be used **to set,get or remove attribute from one of the following** scopes:
page
request

session
application

→ In JSP, **the default scope is page**.

8) page implicit object : In JSP, page is an implicit **object of type Object class**.This **object is assigned to the reference of auto generated servlet class**. It is written as:
Object page=this;
For using this object it must be cast to Servlet type.
For example: **<% (HttpServlet)page.log("message"); %>**
Since, it is of type Object it is less used because **you can use this object directly in jsp**.
For example: **<% this.log("message"); %>**

9) exception implicit object : In JSP, exception is an implicit **object of type java.lang.Throwable** class. This object can be **used to print the exception**. But **it can only be used in error pages**.It is better to learn it after page directive.

6. What is the structure of web.xml.

Ans:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
   "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <servlet>
    <servlet-name>controlServlet</servlet-name>
    <servlet-class>com.jenkov.butterfly.ControlServlet</servlet-class>
        <init-param>
            <param-name>myParam</param-name>
            <param-value>paramValue</param-value>
        </init-param>
  </servlet>

  <servlet-mapping>
```

```
    <servlet-name>controlServlet</servlet-name>
    <url-pattern>*.html</url-pattern>
  </servlet-mapping>
</web-app>
```

- <servlet> element is used to configure the servlet, inside this tag we specify the servlet name and class.
- Then we map the servlet to a URL or a URL pattern. We do this inside the <servlet-mapping> element.
- Using the <init-param> element within the servlet element, we can pass parameters to a servlet from the web.xml file. The parameter can be accessed in the servlet using this code: this.myParam = servletConfig.getInitParameter("myParam");

7. Write a program in JSP using jsp: forward action tag with parameter to print date.

Ans :

index.jsp

```
1. <html>
2. <body>
3. <h2>this is index page</h2>
4.
5. <jsp:forward page="printdate.jsp" >
6. <jsp:param name="name1" value="value1" />
7. </jsp:forward>
8.
9. </body>
10. </html>
```

printdate.jsp

```
1. <html>
2. <body>
3.
4. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
5. <%= request.getParameter("name1") %>
6.
```

7. </body>
8. </html>

8. **Develop servlet application of basic calculator.**

Ans :

html code :-

```
      <html>
    <head>
        <title>Calculator</title>


    </head>
    <body>
        <form action="calculator" method="get" name="frm">

               Enter num1:
               <input name="txt1" type="text" />


                Enter num2:
               <input name="txt2" type="text" />


               Operator

                <select name="op">

                    <option value="Addition">Addition</option>
                    <option value="Subtraction">Subtraction</option>
                    <option value="multiplication">multiplication</option>
                    <option value="division">division</option>
                </select>


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

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

Calculator.java :-

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

```
public class calculator extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

                PrintWriter out = response.getWriter();
                String n1 = request.getParameter("txt1");
                String n2 = request.getParameter("txt2");
                String op = request.getParameter("op");


                if(op.equals("Addition")){
                     out.println((Integer.parseInt(n1) + Integer.parseInt(n2)));
                }
                else if(op.equals("Subtraction")){
                     out.println(Integer.parseInt(n1) - Integer.parseInt(n2));
                }
                else if(op.equals("multiplication")){
                     out.println(Integer.parseInt(n1) * Integer.parseInt(n2));
                }
                else{
                    out.println(Integer.parseInt(n1) / Integer.parseInt(n2));
                }
    }
}
```

9. Explain JSP Actions? How they are different from directives?

Ans :

JSP Actions are basically predefined functions. These actions use constructs in the XML syntax to control the behaviour of the servlet engine. There is only one syntax for the Action elements:
<jsp:action_name attribute="value" />
Available JSP Actions:

1. *<jsp:include>* : This action **lets you insert files into the page** being generated. e.g.: <jsp:include page= "URL" flush= "true" />
2. *<jsp:useBean>* : The useBean action is very versatile. It first **searches for an existing object utilizing the id and the scope variable**. If an object is **not found, it then tries to create the specified object**. e.g.: <jsp:useBean id= "name" class= "package.class" />
3. *<jsp:setProperty>* : This action is **used to set the property of a Bean**. e.g.: <jsp:setProperty name= "myName" property= 'myProperty" />
4. *<jsp:getProperty>* : This action is **used to retrieve the value of a given property**, **convert it to a String**, and **insert it into the output**. e.g.: <jsp:getProperty name= "myName" property= "myProperty" />
5. *<jsp:forward>* : The forward action **terminates the action of the current page and forwards the request to another resource** such as a static page, another JSP page, or a Java Servlet. e.g.: <jsp:forward page= "URL" />

6. *<jsp:plugIn>* : This action is **used to insert JAVA components inside a JSP page**. It determines the **type of browser** and **inserts the <object> or <embed> tags as required**. If the needed plugins are not available, it downloads them and then executes the JAVA component. The JAVA component can be an Applet or a JavaBean.
7. *<jsp:param>* : The <param> action **can be used to send parameters to the Applet or the Bean**, within the <plugin> element. It can also be **used along with the forward and include actions to include additional request parameters to the included or forwarded resources**.

10. Explain Directives in JSP.

Ans :

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.

Directives in JSP provide directions and instructions to its container, telling it how to handle certain aspects of the JSP processing. A JSP directive affects the overall structure of the servlet class.

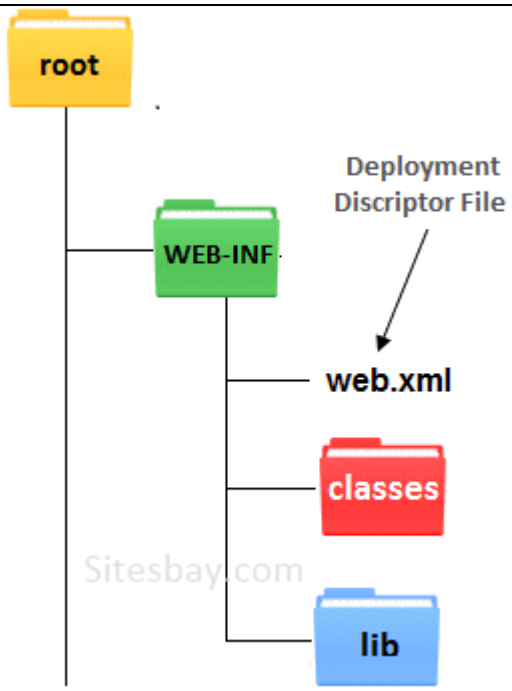
There are three types of directive tag −

Directive Type	Syntax & Description
Page Directive	<%@ page ... %> Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
Include Directive	<%@ include ... %> Includes a file during the translation phase.
Taglib Directive	<%@ taglib ... %> Declares a tag library, containing custom actions, used in the page

11. What is Deployment descriptor in servlet? Explain its structure.

Ans :

It is an xml file which acts as a mediator between a web application developer and a web container. It is also called Deployment Descriptor Files.



In web.xml file, we configure the following

- Servlet
- JSP
- Filter
- Listeness
- Welcome files
- Security
- etc....

To configure a servlet file we need the following two xml tag.

- <Servlet>
- <Servlet-mapping>

<servlet>: tag are used for configure the servlet, Within this tag we write Servlet name and class name.

<Servlet-mapping>: tag are used for map the servlet to a URL .

The structure of web.xml files is already defined by sun MicroSystem. So as a developer we should follows the structure to develop the configuration of web resources.

While configuring a servlet in web.xml, three names are added for it.

- Alias name or register name
- Fully qualified class name
- url-pattern

Syntax

```
<web-app>
<servlet>
<servlet-name>alias name</servlet-name>
<servlet-class>fully qualified class name</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>alias name</servlet-name>
<url-pattern>/url pattern</url-pattern>
</servlet-mapping>

  </web-app>
```

12. WAP a JSP program to manage session.

Ans :

13. Write a servlet code snippet to show the use of RequestDispatcher.

Ans :

14. Write a servlet program to print your result in tabular format.

Ans :

15. Explain JSP Life cycle.

Ans :

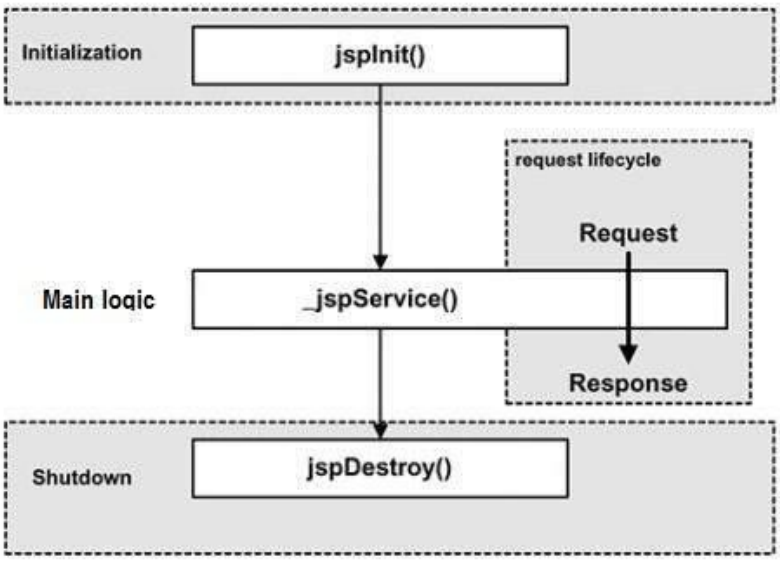
A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

Paths Followed By JSP

The following are the paths followed by a JSP −

Compilation
Initialization
Execution
Cleanup

The four phases have been described below −



1)JSP Compilation :
When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps −

Parsing the JSP.
Turning the JSP into a servlet.
Compiling the servlet.

2)**JSP Initialization :**

When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method −

```
public void jspInit(){
  // Initialization code...
}
```

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

3)JSP Execution :

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP.

The _jspService() method takes an HttpServletRequest and an HttpServletResponse as its parameters as follows −

```
void _jspService(HttpServletRequest request, HttpServletResponse response) {
  // Service handling code...
}
```

The _jspService() method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, GET, POST, DELETE, etc.

4)JSP Cleanup :

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The jspDestroy() method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

16. Explain working of servlet with directory structure.

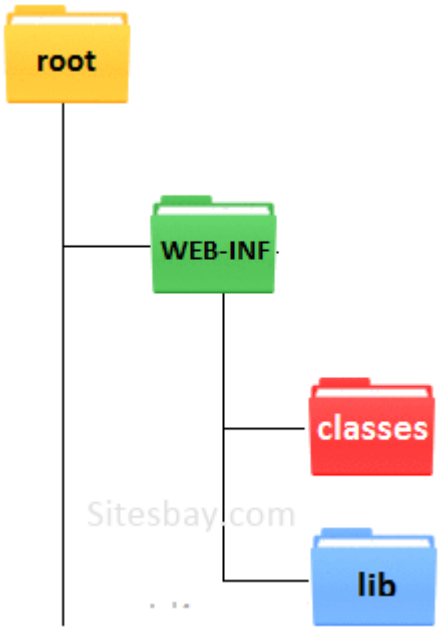
Answer :

For creating web application we should follow standard directory structure provided by sun MicroSystem. Sun MicroSystem has given directory structure to make a web application server independent.

According to directory structure

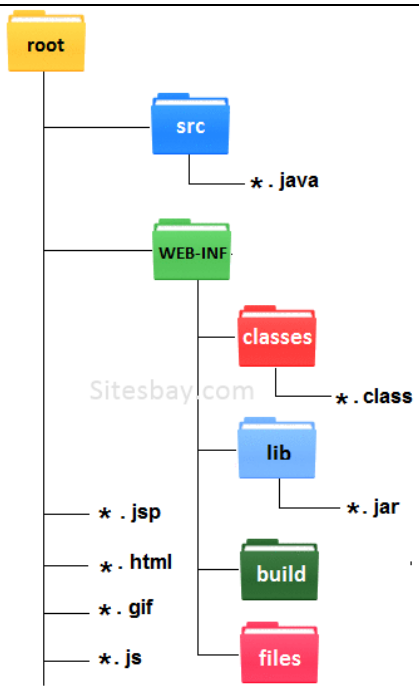
- An application contain root folder with any name.
- Under root folder a sub folder is required with a name WEB-INF.

- Under WEB-INF two sub folder are required classes and lib.



- All jar files placed inside lib folder.
- Under root folder src folder are place for .java files

- Under root folder or under WEB-INF any other folders can exits.
- All image, html, .js, jsp, etc files are placed inside root folder
- All .class files placed inside classes folder.


root
src
*.java
WEB-INF
classes
*.class
lib
*.jar
*.jsp
*.html
*.gif
*.js
build
files

UNIT 3

1.

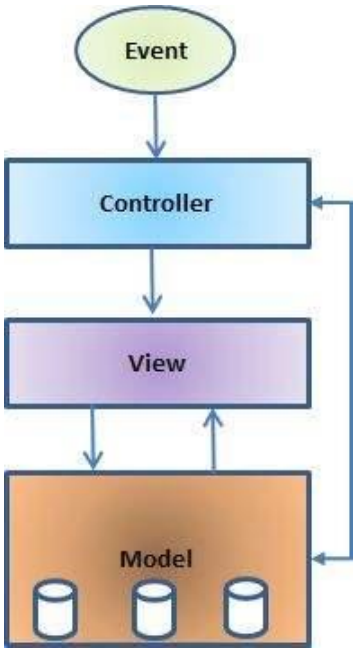
2. Explain MVC architecture in detail?

Ans :

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts −

- Model − The lowest level of the pattern which is responsible for maintaining data.
- View − This is responsible for displaying all or a portion of the data to the user.
- Controller − Software Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.



The Model :-

The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

The View :- It means presentation of data in a particular format, triggered by a controller's decision to present the data. They are script-based templating systems like JSP, ASP, PHP and very easy to integrate with AJAX technology.

The Controller

- ***Note: - For Program you can write only the methods with comments.***
- ***Write the comments wherever necessary in the program.***