# UNIT-7 [Around 10+5=15 marks]

## Servlets and Java Server Pages

### ⊛Web Container:

Web container is runtime environment for a web application. The web applications run within web container. Web container may be built-in with web server or sometimes it can be seperate system. When a request is made from client to web server, it passes the request to web container, which finds the correct resource to handle the request, and then use ~~the response from~~ the resource to generate response and provide it to web server. Some of the important works done by web container are:

→ Communication support between web server and servlets and JSPs.
→ Lifecycle and Resource Management of servlet.
→ Multithreading support.
→ JSP Support
→ Miscellaneous Task.

servlet def^n [Imp]

### ⊛Introduction to Servlet:

Servlets are small programs that execute on the server side of a Web connection. Just as applets dynamically extend the functionality of a web ~~server~~ browser, servlets dynamically extend the functionality of a Web server. It is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-respone programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes. The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods.

Imp!

## ⊛ Life Cycle of a Servlet:

The Servlet Life Cycle has 5 stages in general. Among them three methods are central to the life cycle of a servlet. These are init(), service(), and destroy().

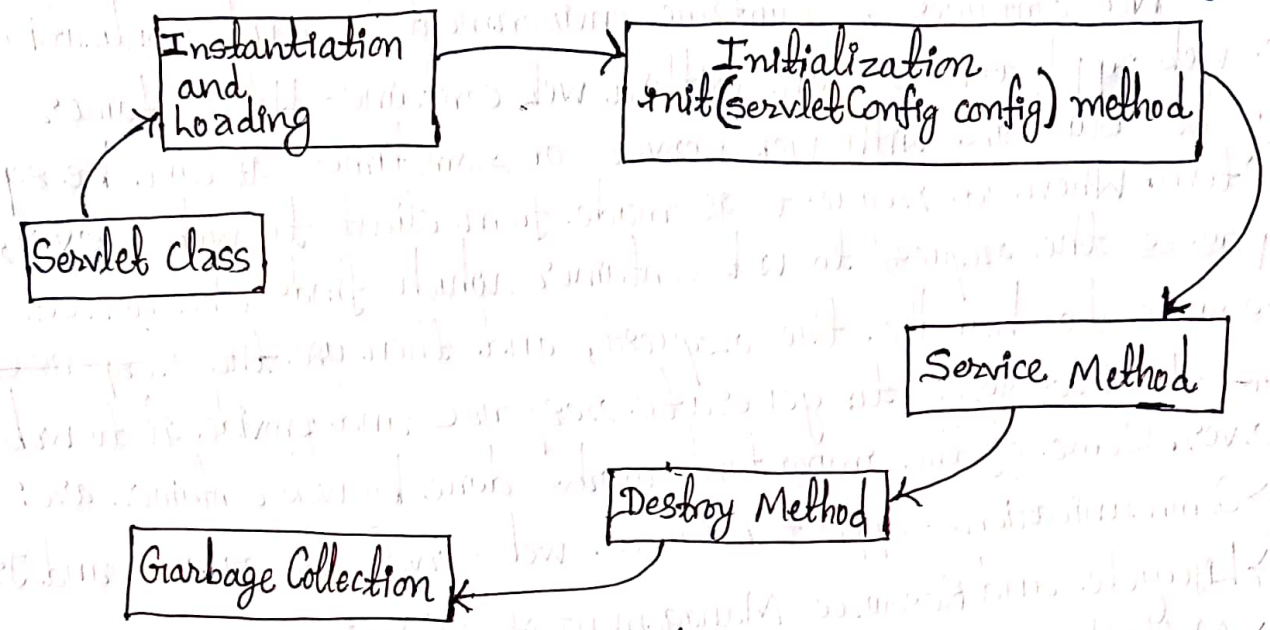


Fig: Servlet life cycle.

i) **Loaded and Instantiated:** The class loader is responsible to load the servlet class. Then it creates the instance of a servlet after loading the servlet class. ~~The servlet engine can instantiate more than one servlet instance.~~ The servlet instance is created only once in the servlet life cycle.

ii) **Initialized by calling the init() method:** The init method is used to initialize the servlet. Syntax of the init method is as follows:

public void init(ServletConfig config) throws ServletException

iii) **Process a clients request by invoking service() method:** The web container calls the service method each time when request for the servlet is received. The syntax is as follows:

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException.

iv) **Terminated by calling the destroy() method:** It gives the servlet an opportunity to clean up any resource for example memory, thread etc. Syntax: public void destroy()

v) **Garbage collected by the JVM:** Once the servlet is destroyed, garbage collector component of JVM is responsible collecting the garbage's.

## ⊛ Servlet APIs: ↔ Empl. for short notes

Two packages contain the classes and interfaces that are required to build servlets. These are javax.servlet and javax.servlet.http. They constitute the Servlet API. These packages are not part of the Java core packages. Instead, they are standard extensions. Therefore, they are not included in the Java Software Development Kit. We must download Tomcat or Glass Fish server to obtain their functionality.

**The javax.servlet Package:** This package contains a number of interfaces and classes that establish the framework in which servlets operate. Following are some of the interfaces and classes:

| Interface | Description |
|---|---|
| Servlet | Declares life cycle methods for a servlet. |
| ServletConfig | Allows servlets to get initialization parameters. |
| ServletRequest | Used to read data from a client request. |
| ServletResponse | Used to write data to a client response. |

| Class | Description |
|---|---|
| GenericServlet | Implements the Servlet and ServletConfig interfaces. |
| ServletInputStream | Provides an input stream for reading requests. |
| ServletOutputStream | Provides an output stream for writing responses. |
| ServletException | Indicates a servlet error occurred. |

**The javax.servlet.http Package:** This package contains a number of interfaces and classes that are commonly used by servlet developers. Following are some of the interfaces and classes:

| Interface | Description |
|---|---|
| HttpServletRequest | Enables servlets to read data from HTTP request. |
| HttpServletResponse | Enables servlets to write data to an HTTP response. |
| HttpSession | Allows session data to be read and written. |

| Class | Description |
|---|---|
| Cookie | Allows state information to be stored on client machine. |
| HttpServlet | Provides methods to handle HTTP requests and responses. |
| HttpSessionEvent | Encapsulates a session-changed event. |

[Imp]

## ⊛ Writing Servlet Programs:

The servlet program can be created by three ways:

i) By implementing servlet interface
ii) By inheriting GenericServlet class
iii) By inheriting HttpServlet class.

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

There are six steps to write Servlet Program (Here we are using apache tomcat server):

i) Create a directory structure
ii) Create a Servlet
iii) Compile the Servlet
iv) Create a deployment descriptor
v) Start the server and deploy the project
vi) Access the servlet

Example DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException{
        res.setContentType("text/html"); //setting the content type.
      PrintWriter pw=res.getWriter(); //get the stream to write data.
      //writing html in the stream
            pw.println("<html><body>");
            pw.println("Welcome to servlet");
            pw.println("</body></html>");

      pw.close(); //closing the stream
}}
```

## ⊛Processing Forms using Servlet: [Imp]

HTML Forms collect data from the user via input elements and then:

→ Request will be sent from HTML Form
→ Request will be either GET/POST method in servlet side
→ Response will be provided from servlet as an HTML form.

Usually sensitive data such as password should be sent using POST method, because GET method passes information from browser to web server producing a long string in browser's URL which can be accessible.

Servlet reads parameters in a form using the following methods depending on the situation:

- getParameter(): We call request.getParameter() method to get the value of a form parameter.
- getParameterValues(): We call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- getParameterNames(): We call this method if we want a complete list of all parameters in the current request.

**Example:** Simple Servlet program that reads and displays data from HTML Form. (using POST method).

Get Method मागे doPost को सट्टा doGet लेख्ने उस्तै same.

Question मा: How do you handle HTTP request (POST/GET) using Servlet भनी यदि example तयार्दा हुन्छ।

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

public class FormServlet extends HttpServlet{
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    //Get the request paramaters
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    //Set the response content type
    response.setContentType("text/html");
    //Get the output stream for writing the response.
    PrintWriter out = response.getWriter();
```

reading using getParameter() यस

```
//Print the response
    out.println("<html>");
    out.println("<head>");
    out.println("<title> Form Servlet </title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1> Form Servlet </h1>");
    out.println("<p> Username: " + username + "</p>");
    out.println("<p> Password: " + password + "</p>");
    out.println("</body>");
    out.println("</html>");
  }
}
```

Almost many things are exactly similar to database connection in unit-4

## ⊛ Database Access with Servlets:

To access a database from a servlet, we need to do following:

1) Load the database driver class.

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

2) Establish a connection to the database:

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/mydatabase", "username", "password");
```

3) Create a statement object:

```
Statement stmt = con.createStatement();
```

4) Execute a SQL Statement.

```
ResultSet rs = stmt.executeQuery("SELECT * FROM users");
```

5) Process the result set using various methods of 'ResultSet' object such as 'next', 'getString', 'getInt' etc.

6) Finally close the connection.

## ⊛Handling Cookies in Servlet:

Short Concepts of cookies and session [Imp]

Cookies and sessions are two common mechanisms used in web application development to store information about a user and track their activity across multiple requests.

Cookies are small pieces of data that are stored by the browser on the clients computer. They can be used to store information such as user's preferences or login status. In servlet we can use the 'javax.servlet.http.Cookie' class to create, read and delete cookies.

⇒To create a cookie and add it to the response, we can use the 'addCookie' method of the 'HttpServletResponse' object. For Example:

```
Cookie cookie = new Cookie("username", "john");
response.addCookie(cookie);
```

⇒To read a cookie from the request, we can use the 'getCookies' method of the 'HttpServletRequest' object. This returns an array of Cookie objects, which we can loop through to find the one we are looking for. For Example:

```
Cookie[] cookies = request.getCookies();
if (cookies != null){
  for(Cookie cookie : cookies){
    if(Cookie.getName().equals("username")){
      String username = cookie.getValue();
        //do something with username like print
    }
  }
}
```

⇒To delete a cookie, we can set its maximum age to 0 and add it to the response. For example:

```
Cookie cookie = new Cookie("username", "");
cookie.setMaxAge(0);
response.addCookie(cookie);
```

## ⊛Handling Sessions in Servlet:

Sessions are used to store information about a user across multiple requests. When a user visits a web application, the server creates a unique session ID and sends it back to the browser in the form of a cookie.

In a servlet, we can use the 'getSession' method of the 'HttpServletRequest' object to access the current session. This method returns a 'javax.servlet.http.HttpSession' object, which we can use to store and retrive information about the user. To store an attribute in the session, we can use the 'setAttribute' method of the 'HttpSession' object. For Example:

```
HttpSession session = request.getSession();
session.setAttribute("username", "roshan");
```

⇒To retrive an attribute from the session, we can use the 'getAttribute' method of the 'HttpSession' object. For example:

```
HttpSession session= request.getSession();
String username= (String) session.getAttribute("username");
```

⇒To invalidate the current session and remove all attributes, we can use the 'invalidate' method of the 'HttpSession' object. For example:

```
HttpSession session= request.getSession();
session.invalidate();
```

## ⊛Servlet vs. JSP: [Imp.]

| Servlets | JSP |
|---|---|
| i) Servlet is a pure java code | i) JSP is a tag based approach. |
| ii) We write HTML in servlet code. | ii) We write JSP code in HTML. |
| iii) Servlet is faster than JSP | iii) JSP is slower than Servlet. |
| iv) Writing code for servlet is harder. | iv) Writing code for JSP is easier. |
| v) Servlet can accept all protocol requests. | v) JSP only accept http requests. |
| vi) Modification in Servlet is a time consuming task because it includes reloading. | vi) JSP modification is fast, we just need to click refresh button. |
| vii) Servlet do not have implict objects | vii) JSP have implicit objects. |
| viii) In MVC pattern, servlet act as a controller. | viii) In MVC pattern, JSP act as a view. |

## ⊛ JSP Access Model:

In JSP, the access model refers to the way in which the built-in objects and other resources are made available to the JSP page. There are three main access models in JSP, which are used ~~use~~ according to the need of application:

i) Page-Based Access: In this model, the built-in objects are made available to the JSP page through local variables.
Syntax: 'request.getParameter("paraName");'

ii) Action-Based Access: In this model, the built-in objects are made available to the JSP page through the use of custom tags.

iii) Bean-Based Access: In this model, the built-in objects are made available to the JSP page through JavaBeans.
Syntax: 'beanName.propertyName'.

## Q. What is JSP? Discuss with suitable example. [Imp, V. Imp]

Java Server Pages (JSP) is a server side technology to create dynamic java web application. It allows java programming code to be embedded in the HTML pages. JSP provides following scripting elements (i.e, JSP Syntaxes):

JSP Comments: Syntax: <%--comments--%>

JSP Scriplet: This tag is used to execute java code in JSP.
Syntax: <% java code %>

JSP Expression: It is used to insert a single java expression directly into the response message, by placing inside a out.print() method.
Syntax: <%= Java Expression %>

JSP Declaration: This tag is used to declare variables and methods.
Syntax: <%! Variable or Method declaration %>

Example: Let we take a simple JSP program to display "Tribhuwan University" 10 times, as our example.

Solution:

```
<!DOCTYPE html>
<html>
    <head> <title> Sample JSP Program </title>
    </head>
    <body> <h1> Displaying "Tribhuwan University" 10 times. </h1>
```

```
<table>
    <%
        for (int i=1; i<=10; i++){
    %>
        <tr><td>Tribhuwan University</td></tr>
    <% } %>
</table>
</body>
</html>
```

Here we used table to display to <tr> <td> used. We can also print directly as Tribhuwan University <br> without any tags

## ⊛ JSP Implicit Objects:

In JSP, implicit objects are objects that are automatically available to the JSP page and do not need to be explicitly created. Some of the implicit objects in JSP are as follows:

i) **request:** It Represents the HTTP request received by the web server and provides access to request-specific information such as parameters, headers, and session attributes.

ii) **response:** It Represents the HTTP response that will be sent back to the client and allows the JSP to control the content and headers of the response.

iii) **pageContext:** Provides access to various components of the JSP environment, such as the ServletContext, HttpSession, and JspWriter.

iv) **session:** Represents the HTTP session for a user and allows the JSP to access and modify session-level data.

v) **page:** Represents the current JSP page and provides access to the page's attributes and methods.

These implicit objects can be accessed using standard Java syntax within a JSP page. For example, to access the request object, we would use 'request', and to access a parameter passed in the request, we would use 'request.getParameter("paramName")';

## ⊛ JSP Syntax for Directions:

JSP Syntax मा direction हुन्छ so यता लेखेको

Syntax: <%@ directive %>

It is used to specify various attributes for the page or to include other resources in the page.

Examples:

```
<%@ attribute="value" %>
<%@ include file="filename" %>
```

## ⊛ Object Scope: [Imp]

In JSP, object scope refers to the availability of an object within the page, application, or session. The scope of an object determines where it can be accessed and for how long it is available. There are four object scopes in JSP:

i) Page scope: An object with page scope is available only within the current JSP page and is not accessible from other pages.

ii) Request scope: An object with request scope is available within the current request and response cycle and is ~~is~~ not accessible from other requests.

iii) Session scope: An object with session scope is available to all pages within a user's session and is not accessible from other sessions.

iv) Application scope: An object with application scope is available to all pages in the web application and is not accessible from other applications.

## ⊛ Processing Forms in JSP: [Imp]

To process a form in a JSP page, we can use a combination of HTML and JSP to create the form and handle the form submission. Here is an example of simple form that allows user to enter their name and submit the form:

```
<form action="processForm.jsp" method="post">
    Name: <input type="text" name="name"> <br>
    <input type="submit" value="Submit">
</form>
```

To process the form submission in the JSP page, we can use the 'request' implicit object to access the form data. For example:

```
<%
    String name = request.getParameter("name");
    //process the form data
%>
```

This code uses the 'getParameter' method of the 'request' object to retrive the value of the "name" parameter from the form submission. We can then use this value to process the form data as needed.

## ⊛ Database Access with JSP:

To access a database from a JSP page, we will need to follow following steps:

→ Configure a JDBC driver and connection pool.
→ Establish a database connection.
→ Execute a SQL query
→ Process the result set
→ Close the connection.

Example:

```
<%@ page import="java.sql.*" %>
<%
    //Load the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

  //Configure the connection pool
    String url="jdbc:mysql://localhost:3306/mydatabase";
    String username ="myuser";
    String password="myuser123";
    Connection pool=DriverManager.getconnection(url,username,password);
  //Establish a connection
    Connection conn=pool.getConnection();
  //Execute a SQL query
    Statement stmt=conn.createStatement();
    ResultSet rs=stmt.executeQuery("SELECT * FROM mytable");

  //Process the result set
    while(rs.next()){
          String name=rs.getString("name");
          out.println("Name: "+name+"<br>");
    }
  //Close the connection
    conn.close();
%>
```

This example loads the MySQL JDBC driver, establishes a connection to a database named "mydatabase", executes a SQL query to select all rows from a table named "mytable", and prints the "name" column of each row to web page.

## ⊛Introduction to Java Web Frameworks:

A web framework is a collection of libraries and tools that make it easier to develop web applications. Web Frameworks provide a standard way to build and deploy web applications, and they can help to save time and effort. There are many Java web frameworks available., and each has its own strengths and limitations. Some popular Java web frameworks include:

i) **Spring:** Spring is widely used framework for building Java applications. It provides a range of features including dependency injection, data access, and web support.

ii) **Hibernate:** Hibernate is an object-relational mapping (ORM) framework that simplifies the process of storing and retriving data fom a database. It is often used with Spring.

iii) **Struts:** Struts is an older web framework that is still used by some developers. It uses a MVC architecture and provides support for actions, forms, and validation.

iv) **JSF:** It is a framework for building user interfaces for Java Web applications. It provides support for actions, forms, validation. JSF stands for JavaServer Faces.

v) **Play:** Play is a modern web framework that is designed to be easy to use and scalable. It uses an event-driven architecture and supports real-time web applications.

If my notes really helped you,then you can support me on esewa for my hardwork.

Esewa ID: 9806470952