**INTERNITY FOUNDATION**

**TASK-9**

**Submitted By:**

**Amisha Singhal**

**Java Batch**

**Servlets  
Web Application basics**

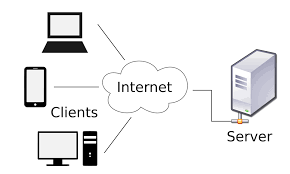
**Ans-** A Java web application is a collection of dynamic resources (such as Servlets, JavaServer Pages, Java classes and jars) and static resources (HTML pages and pictures). A web application is computer software that can be accessed using any web browser. Usually, the frontend of a web application is created using the scripting languages such as HTML, CSS, and JavaScript, supported by almost all web browsers. A Java web application can be deployed as a WAR (Web Archive) file. A WAR file is a zip file which contains the complete content of the corresponding web application.

**Client-Server Architecture**

**Ans-** The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and delivers the data packets requested back to the client. Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.

In a client/server architecture, the server acts as the producer and the client acts as a consumer. The server houses and provides high-end, computing-intensive services to the client on demand. These services can include application access, storage, file sharing, printer access and/or direct access to the server’s raw computing power.

Client/server architecture works when the client computer sends a resource or process request to the server over the network connection, which is then processed and delivered to the client. A server computer can manage several clients simultaneously, whereas one client can be connected to several servers at a time, each providing a different set of services.

****

**Servlet**

**Ans-** Servlet is a technology which is used to create a web application. Servlet is an API that provides many interfaces and classes including documentation. Servlet is an interface that must be implemented for creating any Servlet. Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests. Servlet is a web component that is deployed on the server to create a dynamic web page.



**HTTP Servlet**

**Ans-** The **HttpServlet** class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

In Http Servlet there is no need to override the service() method because this method dispatches the Http Requests to the correct method handler, for example if it receives HTTP GET Request it dispatches the request to the doGet() method.

**Eg-**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**// Extend HttpServlet class**

**public class HelloWorld extends HttpServlet {**

**private String message;**

**public void init() throws ServletException {**

**// Do required initialization**

**message = "Hello World";**

**}**

**public void doGet(HttpServletRequest request, HttpServletResponse response)**

**throws ServletException, IOException {**

**// Set response content type**

**response.setContentType("text/html");**

**// Actual logic goes here.**

**PrintWriter out = response.getWriter();**

**out.println("<h1>" + message + "</h1>");**

**}**

**public void destroy() {**

**// do nothing.**

**}**

**OUTPUT- Hello World!**

**Servlet Lifecycle**

**Ans**- The entire life cycle of a Servlet is managed by the **Servlet container** which uses the **javax.servlet.Servlet** interface to understand the Servlet object and manage it.

Stages of the Servlet Life Cycle:

* Loading a Servlet.
* Initializing the Servlet.
* Request handling.
* Destroying 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: |

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.

**Syntax:**

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.

**Syntax:**

public void destroy()

**  
Servlet Engine**

**Ans-** Servlet Engines are a Java platform technology for extending servlets. All Servlet Engines have a common API. The Servlet Connector component of ArcIMS requires a Servlet Engine if your Web server does not support servlets natively.  
  
Java Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems.Servlets provide a component-based, platform-independent method for building Web-based applications. **Redirect & Request Dispatcher**

**Ans-** The **RequestDispatcher** interface allows you to do a server side forward/include whereas **sendRedirect()** does a client side redirect.  
  
SendRedirect() will search the content between the servers. it is slow because it has to intimate the browser by sending the URL of the content. then browser will create a new request for the content within the same server or in another one.  
  
RquestDispatcher is for searching the content within the server i think. its the server side process and it is faster compare to the SendRedirect() method. but the thing is that it will not intimate the browser in which server it is searching the required date or content, neither it will not ask the browser to change the URL in URL tab. so it causes little inconvenience to the user.

**Session Tracking**

**Ans- Session Tracking** tracks a user’s requests and maintains their state. It is a mechanism used to store information on specific users and in order to recognize these user’s requests when they connect to the web server.

Sessions are shared among the servlets accessed by a client. This is convenient for applications made up of multiple servlets. For example, **Duke's Bookstore** uses session tracking to keep track of the books being ordered by a user. All the servlets in the example have access to the user's session.

To use session tracking,

* Obtain a session (an [HttpSession](http://jserv.javasoft.com/products/java-server/documentation/webserver1.1/apidoc/javax.servlet.http.HttpSession.html" \t "_top) object) for a user.
* Store or get data from the HttpSession object.
* Invalidate the session (optional).

**URL Writing**

**Ans-** Url rewriting is a process of appending or modifying any url structure while loading a page. The request made by client is always a new request and the server cannot identify whether the current request is send by a new client or the previous same client. Due to This property of HTTP protocol and Web Servers are called stateless. But many times we should know who is client in the processing request.

It will always work whether cookie is disabled or not (browser independent).Extra form submission is not required on each page.

**Syntax:**

**url?name1=value1&name2=value2&??  
HTTP Authentication**

**Ans-** HTTP Basic authentication implementation is the simplest technique for enforcing access controls to web resources because it doesn't require cookies, session identifier and login pages. Rather, HTTP Basic authentication uses static, standard HTTP headers which means that no handshakes have to be done in anticipation. Basic authentication requires an instance of UsernamePasswordCredentials (which NTCredentials extends) to be available, either for the specific realm specified by the server or as the default credentials.

**Form Based Authentication**

**Ans-** Form-based authentication allows you to control the look and feel of the login page. Form-based authentication works like basic authentication, except that you specify a login page that is displayed instead of a dialog and an error page that's displayed if login fails.

Like basic authentication, form-based authentication is not secure because passwords are transmitted as clear text. Unlike basic and digest authentication, form-based authentication is defined in the servlet specification, not the HTTP specification.

Form-based login allows customization of the login page, but not the authentication process itself.

**Connectivity with Database using JDBC**

**Ans-** 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.

 The steps to connect with database using JDBC are:

* Register the Driver class
* Create connection
* Create statement
* Execute queries
* 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:

**public** **static** **void** forName(String className)**throws** ClassNotFoundException

**2) Create the connection object**

|  |
| --- |
| The **getConnection()** method of DriverManager class is used to establish connection with the database. |

**Syntax:**

**public** **static** Connection getConnection(String url)**throws** SQLException

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

**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:**

**public** Statement createStatement()**throws** SQLException

**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:**  
**public** ResultSet executeQuery(String sql)**throws** SQLException

**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:**

**public** **void** close()**throws** SQLException