Unit-1 (Foundation of Enterprise Programming)

Introduction to JDBC in Java

- Java Database Connectivity (JDBC) is an API that provides industry-standard and database-independent connectivity between Java applications and relational databases / non-relational databases.
- JDBC allows Java applications to perform database operations like querying, updating, and retrieving data from relational databases, non-relational databases, spreadsheets, and flat files.
- JDBC acts as an abstraction layer between Java application and the database, allowing programmers to write Java code once and run that code on different databases without modifying the code - only JDBC driver needs to be changed.
- It provides two packages java.sql and javax.sql.

Components of JDBC

There are generally four main components of JDBC through which it can interact with a database. They are as mentioned below:

1. JDBC API:

It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE and Java EE platforms to exhibit WORA(write once run anywhere) capabilities. The **java.sql** package contains interfaces and classes of JDBC API.

2. JDBC Driver manager:

It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.

3. JDBC Test suite:

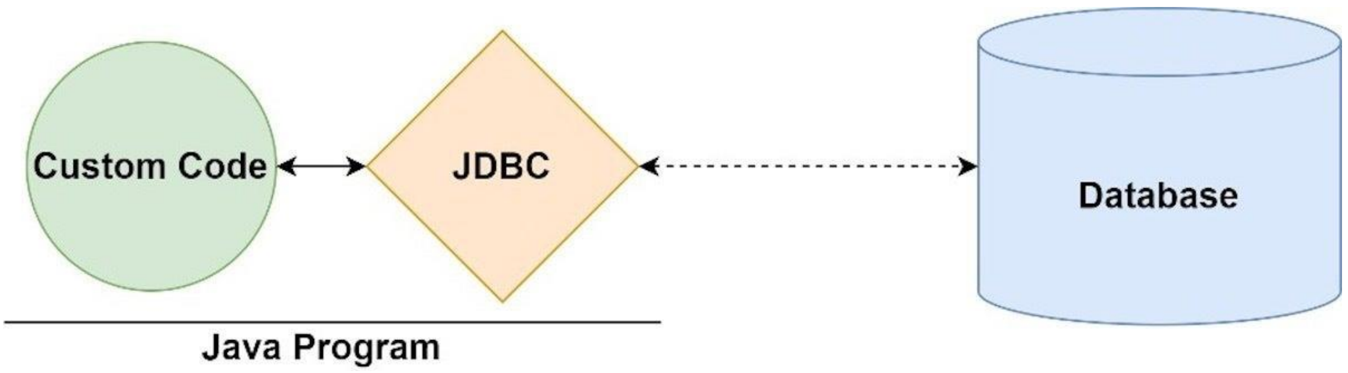
It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.

4. JDBC-ODBC Bridge Drivers:

It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the **sun.jdbc.odbc** package which includes a native library to access ODBC characteristics.

How JDBC works

- You can use JDBC to interact with a database from within a Java program.
- JDBC acts as a bridge from your code to the database.



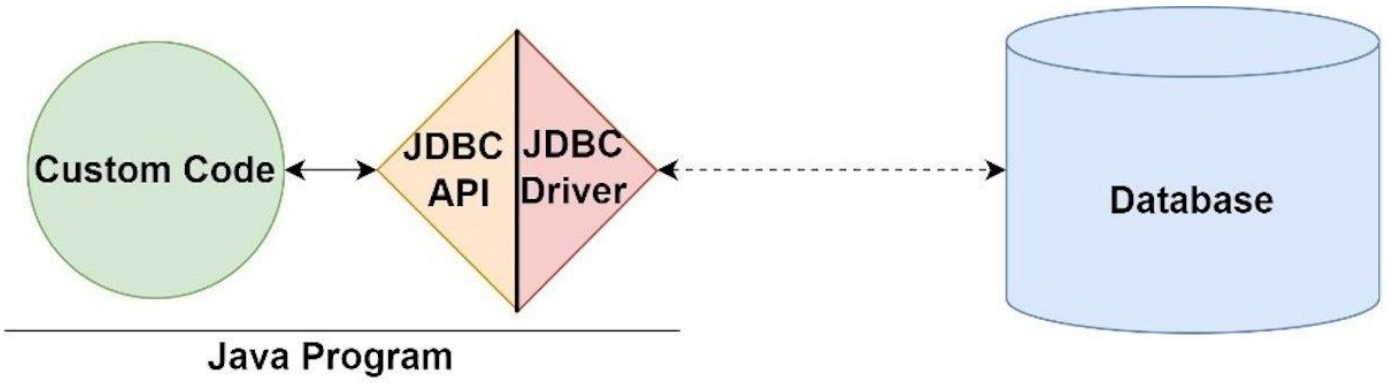
JDBC's Architecture

The JDBC interface consists of two layers:

- The JDBC API supports communication between the Java application and the JDBC manager.
- The JDBC driver supports communication between the JDBC manager and the database driver.

The JDBC API and JDBC driver have been refined extensively over the years, resulting in a feature-rich, performant, and reliable library.

JDBC is the common API that your application code interacts with. Beneath that is the JDBC-compliant driver for the database you are using.



JDBC Drivers

- are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the Database Management System can understand.
- JDBC drivers are the software components which implements interfaces in JDBC APIs to enable java application to interact with the database.
- There are four types of drivers.
    - Type-1 driver or JDBC-ODBC bridge driver
    - Type-2 driver or Native-API driver
    - Type-3 driver or Network Protocol driver
    - Type-4 driver or Thin driver

1. **Type-1 driver / JDBC-ODBC bridge driver**

   Uses ODBC driver to connect to the database. It converts JDBC method calls into the ODBC function calls. It is a universal driver because it can be used to connect to any of the databases. It is built-in with JDK so need to install separately in client machine.

2. **Type-2 driver / Native-API driver**

   It uses client-side libraries of the database. It converts JDBC method calls into native calls of the database API in order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.

3. **Type-3 driver / Network Protocol driver**

   It uses middleware that converts JDBC calls directly or indirectly into the vendor-specific database protocol. No client-side library is required to installed. Database migration is very easy using this driver.

4. **Type-4 driver / Thin driver**
   It is also called native protocol driver. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

JDBC API (Classes and Interfaces)

DriverManager class

In Java, the DriverManager class it an interface between the User and the Driver. This class is used to have a watch on driver which is been used for establishing the connection between a database and a driver. The DriverManager class have a list of Driver class which are registered and are called as DriverManager.registerDriver().

S.No.	Method	Description
1	public static void registerDriver(Driver driver)	It is used for Registering the Driver with the Driver Manager.
2	public static void deregisterDriver(Driver driver)	It is used for Deregistering the Driver with the Driver Manager.
3	public static Connection getConnection(String Url)	It is used for establishing a connection with the given URL.
4	public static Connection getConnection(String Url, String username, String password)	It is used for establishing the connection with the given URL, username and password.

Connection interface

In Java, The Connection interface is used for creating the session between the application and the database. This interface contains Statement, PreparedStatement and DatabaseMetaData. The connection objects are used in Statement and the **DatabaseMetaData. commit(), rollback()** etc.. are some of the methods of Connection Interface.

S.No	Method	Description
1	public Statement createStatement()	It is used for creating an object of statement for executing the SQL queries.
2	public Statement createStatement(intresultSetType,intresultSetConcurrency)	It is used for creating objects for the ResultSet from the given type and concurrency.
3	public void setAutoCommit(boolean status)	It is used for setting the commit status. By default, it is always true.

4	public void commit()	It is used to save the changes which have been commit or rollback permanent
5	public void rollback()	It is used to delete the changes which have been commit or rollback permanent
6	public void close()	It is used to delete the changes which have been commit or rollback permanent

Statement interface

In Java, The Statement interface is used for executing queries using the database. This interface is a factory of ResultSet. It is used to get the Object of ResultSet. Methods of this interface is given below.

S.No.	Method	Description
1	public ResultSetexecuteQuery(String sql)	It is used for executing the SELECT query
2	public intexecuteUpdate(String sql)	It is used for executing any specified query
3	public boolean execute(String sql)	It is used when multiple results are required.
4	public int[] executeBatch()	It is used for executing the batch of commands.

ResultSet interface

In Java, the ResultSet Interface is used for maintaining the pointer to a row of a table. In starting the pointer is before the first row. The object can be moved forward as well as backward direction using TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int). Methods of this interface is given below.

S.No.	Method	Description
1	public boolean next()	It is used for moving the cursor to the next position from the current position.
2	public boolean previous()	It is used for moving the cursor to the previous position from the current position.

3	public boolean first()	It is used for moving the cursor to the first position from the current position.
4	public booleanlast()	It is used for moving the cursor to the Last position from the current position.
5	public booleanabsolute(int row)	It is used for moving the cursor to the specified position from the current position.
6	public booleanrelative(int row)	It is used for moving the cursor to the relative row number from the current position.
7	public intgetInt(intcolumnIndex)	It is used to get the data from the specified position.
8	public intgetInt(String columnName)	It is used to get the data from the specified column name of the current row.
9	public StringgetString(intcolumnIndex)	It is used to get the data from the specified column name of the current row in form of an integer.
10	public StringgetString(StringcolumnIndex)	It is used to get the data from the specified column name of the current row in form of string.

ResultSet Type Values

You can create a Statement that returns result sets in one of the following types:

- TYPE_FORWARD_ONLY: the result set is not scrollable (default).
- TYPE_SCROLL_INSENSITIVE: the result set is scrollable but not sensitive to database changes.
- TYPE_SCROLL_SENSITIVE: the result set is scrollable and sensitive to database changes.

ResultSet Concurrency Values

A Statement can return result sets which are read-only or updatable, specified by one of the following constants defined in the ResultSet interface:

- CONCUR_READ_ONLY: the result set cannot be used to update the database (default).
- CONCUR_UPDATABLE: the result set can be used to update the database.

Example

```
try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        conn =DriverManager.getConnection("jdbc:mysql://localhost:3306/myData",
                "root", "pwd");
        Statement stmt = conn.createStatement();

        ResultSet result = stmt.executeQuery(“select * from employees”);
        while(result.next()) {
                System.out.println(result.getInt(1) + "\t" + result.getString(2) +
                " " + result.getString(3));
        }
        conn.close();
}
catch (Exception e) {
        System.out.println("my Error :" + e.getMessage());
}
```

PreparedStatement interface

In Java, The PreparedStatement interface is a subinterface of Statement. It is mainly used for the parameterized queries. A question mark (?) is passed for the values. The values to this question marks will be set by the PreparedStatement. Methods of this interface is given below.

S.No.	Method	Description
1	public void setInt(intparamIndex, int value)	It is used for setting the integer value for the given parameter index.
2	public void setString(intparamIndex, String value)	It is used for setting the String value for the given parameter index.
3	public void setFloat(intparamIndex, float value)	It is used for setting the Float value for the given parameter index.
4	public void setDouble(intparamIndex, double value)	It is used for setting the Double value for the given parameter index.
5	public intexecuteUpdate()	It is used for executing a query.
6	public ResultSetexecuteQuery()	It is used for executing the select query.

Difference between Statement and Prepared Statement

Statement	Prepared Statement
This JDBC API interface is used for static SQL statements at run time.	The PreparedStatement interface is used for dynamic SQL statements at run time.
It is base interface.	It extends statement interface.
Each execution is parsed and compiled by the database, which can be inefficient for repeated executions.	The SQL statement is precompiled, and the database can reuse the precompiled statement, improving performance for repeated executions.
Vulnerable to SQL injection attacks as it does not support parameterized queries.	Protects against SQL injection attacks by using parameterized queries, where the input values are treated as parameters rather than executable code.

Suitable for executing simple SQL queries without parameters.	Suitable for executing dynamic SQL queries with parameters, offering better performance and security.
Statement is slower as compared to PreparedStatement in java JDBC.	PreparedStatement is faster because it is used for executing precompiled SQL statement in java JDBC.
There could be a possibility of writing concatenated SQL statements while using the Statement interface.	There’s no need of writing concatenated SQL statements when using the PreparedStatement interface.
We can not use statement for reading binary data.	We can use Preparedstatement for reading binary data.
No binary protocol is used for communication.	Binary protocol is used for communication.

Example

```
public static void main(String[] args) {
        int mEmpno,mDeptno;
        String mFname, mLname, mMobile, mEmail, mJob;

        Scanner scn = new Scanner(System.in);

        System.out.print("Enter the valid employee number :");
        mEmpno = scn.nextInt();
        System.out.print("Enter the valid employee fname  :");
        mFname = scn.next();
        System.out.print("Enter the valid employee lanme  :");
        mLname = scn.next();
        System.out.print("Enter the valid employee mobile :");
        mMobile = scn.next();
        System.out.print("Enter the valid employee email  :");
        mEmail = scn.next();
        System.out.print("Enter the valid employee job    :");
        mJob = scn.next();
        System.out.print("Enter the valid employee deptno :");
        mDeptno = scn.nextInt();

        Connection conn = null;
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn =DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/classicmodels?useSSL=false",
                    "root", "password");

                String SQL = "insert into empmast values(?,?,?,?,?,?,?)";
                PreparedStatement pstmt = conn.prepareStatement(SQL);

                //set the value to each parameter
                pstmt.setInt(1, mEmpno);
                pstmt.setString(2, mFname);
                pstmt.setString(3, mLname);
                pstmt.setString(4, mMobile);
                pstmt.setString(5, mEmail);
                pstmt.setString(6, mJob);
                pstmt.setInt(7, mDeptno);

                int rowaffected = pstmt.executeUpdate();
                if(rowaffected > 0) {
                    System.out.println("Record inserted successfully !");
                }
                else
                    System.out.println("Error in insert !");
                conn.close();
        }
        catch (Exception e) {
            System.out.println("Error :" + e.getMessage());
        }
    }
```

ResultSetMetaData Interface

The ResultSetMetaData interface is used to get metadata from the ResultSet object. Metadata are the data about data. Methods of this interface is given below.

S.No.	Method	Description
1	public int getColumnCount()throws SQLException	It is used to get the total number of columns.
2	public String getColumnName(int index)throws SQLException	It is used to get the name of the column of a specified column index.
3	public String getColumnTypeName(int index)throws SQLException	It is used to get the name of the column of a specified index.
4	public String getTableName(int index)throws SQLException	It is used to get the name of a table from the specified column index

DatabaseMetaData Interface

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc. Methods of this interface is given below.

S.No.	Method	Description
1	public String getDriverName() throws SQLException	It is used to get driver name of the current connection.
2	public String getDriverVersion() throws SQLException	It is used to get driver version number of the current connection.
3	public String getUserName() throws SQLException	It is used to get user name of the current connection.
4	public String getDatabaseProductName() throws SQLException	It is used to get the database vendor product name of the current connection.

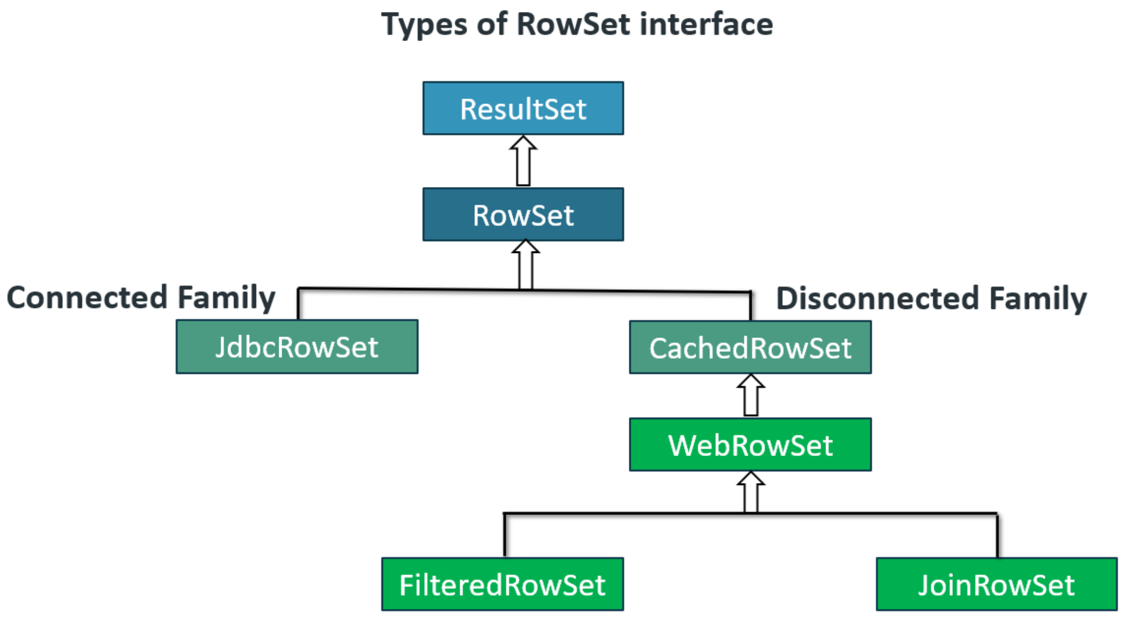
5	public String getDatabaseProductVersion() throws SQLException	It is used to get the database vendor product version number of the current connection.
5	public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types) throws SQLException	It is used to list database objects from connection database.

CallableStatement Interface

The CallableStatement interface extends the PreparedStatement interface and provides methods for executing stored procedures in a database. Stored procedures are precompiled SQL statements stored in the database, which can be executed with the help of CallableStatement.

RowSet Interface

- A JDBC RowSet facilitates a mechanism to keep the data in tabular form.
- It is the wrapper of ResultSet.
- A JDBC RowSet object holds tabular data in a style that makes it more adaptable and simpler to use than a result set.



JdbcRowSet

- It basically acts as a wrapper around the ResultSet object with some additional functionality.
- It is only connected RowSet in the family.

- The primary advantage of using JdbcRowSet is that it enables the ResultSet object to be used as a JavaBeans component.
- It is scrollable and updatable capabilities to the ResultSet object.

CachedRowSet

- CachedRowSet object is unique because it can operate without being connected to its data source. We call this a “disconnected RowSet object”.
- it caches its data in memory so that it can operate on its own data instead of the data stored in a database.

WebRowSet

- It extends CachedRowSet capabilities but is very special in the sense that in addition to providing all the features of CachedRowSet, it can read and write XML document.
- it can write itself to an XML document and can also read that XML document to convert itself back to a WebRowSet.
- It is mainly used in enterprise application scenario or in web service communication.

FilteredRowSet

- It is an extension of WebRowSet.
- filtering data based on criteria to fetch selected rows from the data source so that we can work with the relevant data.
- It is something like using the WHERE clause without writing an SQL.

Introduction to Maven

- Maven is an automation and management tool developed by Apache Software Foundation.
- In simple words, Maven is a tool that can be used for building and managing any Java-based project. maven makes the day-to-day work of Java developers easier and generally helps with the comprehension of any Java-based project.
- It allows developers to create projects, dependency, and documentation using Project Object Model and plugins.
- It has a similar development process as ANT, but it is more advanced than ANT.

The key features of Maven are:

- Convention over configuration: avoid as much configuration as possible, by choosing real world default values and supplying project templates (archtypes).
- Dependency management: support the definition and usage of dependencies to other libraries and projects. During the build, the Maven build system resolves the dependencies and it also builds the dependent projects if needed.
- Extensible via plug-ins: The Maven build system is extensible via plug-ins, which allows to keep the Maven core small. The Maven core does for example not know how to compile Java source code, this is handled by the compiler plug-in.

How to use Maven

- To configure the Maven in Java, you need to use Project Object Model, which is stored in a pom.xml-file.
- POM includes all the configuration setting related to Maven. Plugins can be configured and edit in the <plugins> tag of a pom.xml file. And developer can use any plugin without much detail of each plugin.
- When user start working on Maven Project, it provides default setting of configuration, so the user does not need to add every configuration in pom.xml

What is Build LifeCycle ?

A build lifecycle is a well-defined sequence of Phases, which define the order in which the goals are to be executed. Here phase represents a stage life cycle.

Phase	Handles	Description
Prepare-resources	resource copying	Resource copying can be customized in this phase.
validate	Validating the Information	Validates the project configuration.
compile	Compilation	Compiles the source code into bytecode.
test	Test the project	Runs the tests for the project.
package	Packaging the project	Package the compiled code and resources into artifact (eg. JAR or WAR)
install	Install the artifact	Install the artifact in the local repository.
deploy	Deploy the project	Copies the artifact to a remote repository.

What is Maven Repository ?

- Maven repositories are directories of packaged JAR files with some metadata.
- The metadata are POM files related to the projects each packaged JAR file belongs to, including what external dependencies each packaged JAR has.
- This metadata enables Maven to download dependencies of your dependencies recursively until all dependencies are download and put into your local machine.
- Maven has three types of repository :
    - Maven searches for dependencies in this repositories. First maven searches in Local repository
    - then Central repository
    - then Remote repository if Remote repository specified in the POM.
    - **Local repository :** A local repository is a directory on the machine of developer. This repository contains all the dependencies Maven downloads. Maven only needs to download the dependencies once, even if multiple projects depends on them (e.g. ODBC). By default, maven local repository is user_home/m2 directory. example – **C:\Users\asingh\.m2**

- **Central repository :** The central Maven repository is created Maven community. Maven looks in this central repository for any dependencies needed but not found in your local repository. Maven then downloads these dependencies into your local repository.
- **Remote repository :** Remote repository is a repository on a web server from which Maven can download dependencies.it often used for hosting projects internal to the organization. Maven then downloads these dependencies into your local repository.

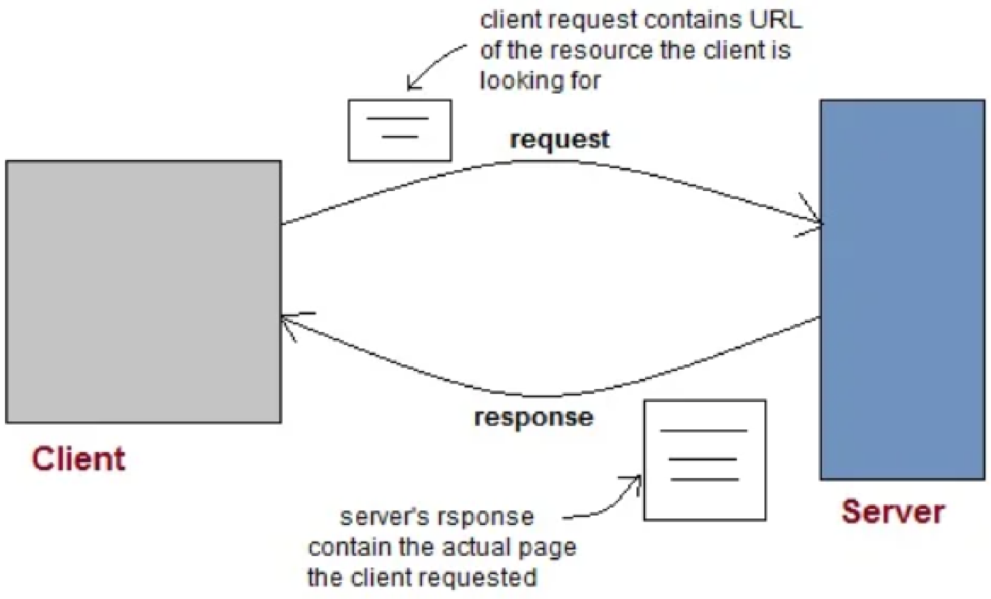
What is Maven Used For?

- Maven can be used for the following:
- We can easily build a project using maven.
- We can add jars and other dependencies of the project easily using the help of maven.
- Maven provides project information (log document, dependency list, unit test reports, etc.)
- Maven is very helpful for a project while updating the central repository of JARs and other dependencies.
- With the help of Maven, we can build any number of projects into output types like the JAR, WAR, etc without doing any scripting.
- Using maven we can easily integrate our project with a source control systems (such as Subversion or Git).
- Maven also helps in managing the project's build lifecycle, including tasks like compiling, testing, packaging, and deploying the code.
- Maven provides a standard project structure, making it easy for developers to understand the layout of the project and locate specific files.
- Maven supports multi-module projects, allowing developers to work on multiple related projects simultaneously and manage their dependencies efficiently.
- Maven plugins can be used to add additional functionality to the build process, such as code coverage analysis, static code analysis, and more.
- Maven is highly customizable, allowing developers to configure the build process to meet their specific needs and requirements.
- Maven simplifies the process of managing project dependencies, ensuring that the correct versions of libraries and frameworks are used throughout the project.

Unit-2 (Servlet Programming)

Introduction to Web

- Web consists of billions of clients and server connected through wires and wireless networks.
- The web clients make requests to web server. The web server receives the request, finds the resources and return the response to the client.
- When a server answers a request, it usually sends some type of content to the client.
- The client uses web browser to send request to the server.
- The server often sends response to the browser with a set of instructions written in HTML (Hypertext Markup Language).
-

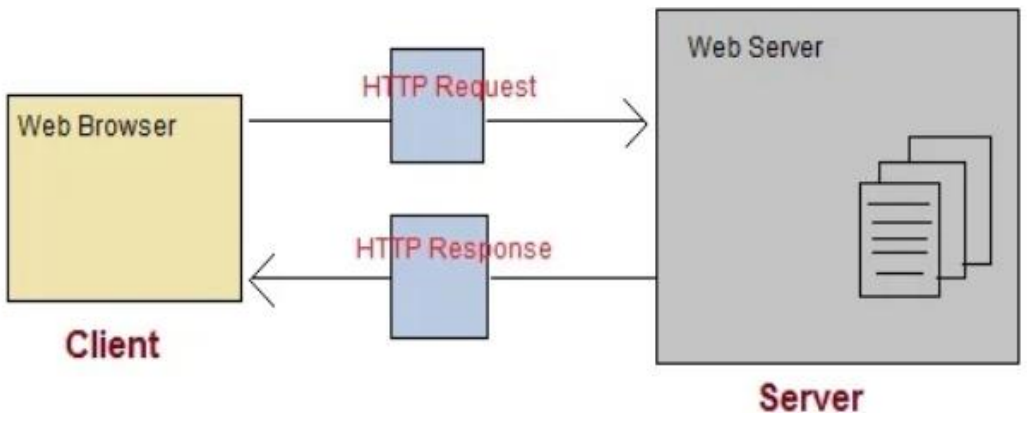

client request contains URL of the resource the client is looking for
request
response
Client
Server
server's rsponse contain the actual page the client requested

Web Application

- A website is a collection of static files(pages) such as HTML pages, images, graphics etc.
- A Web application is a web site with dynamic functionality on the server. Google, Facebook, Twitter are examples of web applications.

HTTP (Hypertext Transfer Protocol)

- HTTP is a protocol that clients and servers use on the web to communicate.
- It is similar to other internet protocols such as SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol) but there is one fundamental difference.
- HTTP is a stateless protocol i.e HTTP supports only one request per connection. This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
- The client sends an HTTP request and the server answers with an HTML page to the client, using HTTP.



HTTP Methods

- HTTP request can be made using a variety of methods, but the ones you will use most often are Get and Post.
- The method name tells the server the kind of request that is being made, and how the rest of the message will be formatted.

HTTP Methods and Description

Method Name	Description
GET	Request to retrieve information from server using a given URI.
POST	Request for server to accept the entity enclosed in the body of HTTP method.
PUT	• This is same as POST, but POST is used to create, PUT can be used to create as well as update. • It replaces all current representations of the target resource with the uploaded content
DELETE	Request for the Server to delete the resource.
OPTIONS	Request for communication options that are available on the request/response chain.
HEAD	Identical to GET except that it does not return a message-body, only the headers and status line.

Difference between GET and POST method

GET Method	POST Method
Data is sent in header to the server	Data is sent in the request body
Get request can send only limited amount of data	Large amount of data can be sent.
Get request is not secured because data is exposed in URL	Post request is secured because data is not exposed in URL.
Get request can be bookmarked and is more efficient.	Post request cannot be bookmarked.

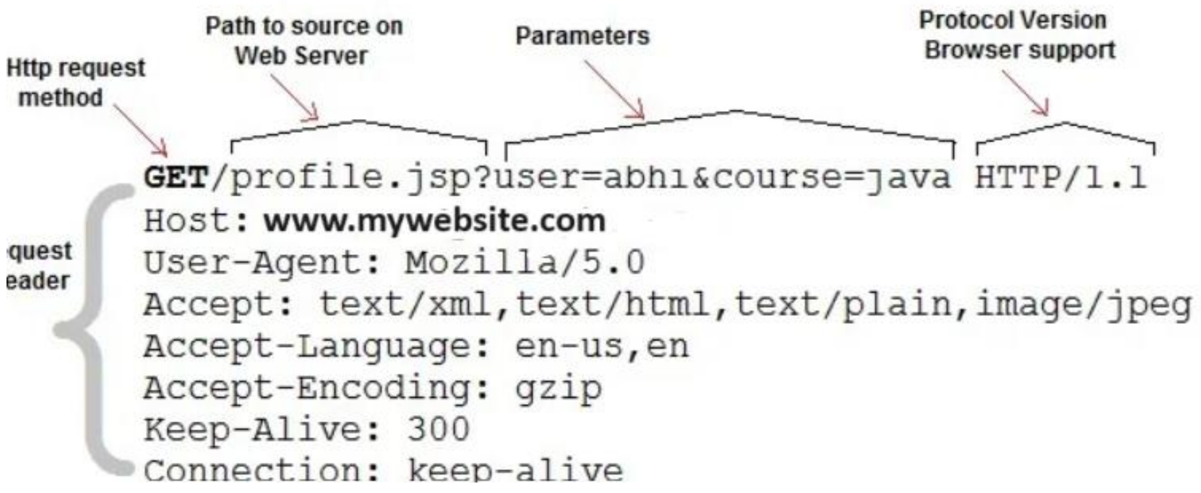
General Difference between PUT and POST methods

Following are some basic differences between the PUT and the POST methods :

- POST to a URL creates a child resource at a server defined URL while PUT to a URL creates/replaces the resource in its entirety at the client defined URL.
- POST creates a child resource, so POST to /books will create a resources that will live under the /books resource. Eg. /books/1. Sending the same post request twice will create two resources.
- PUT is for creating or replacing a resource at a URL known by the client.
- PUT must be used for CREATE when the client already knows the url before the resource is created.
- PUT replaces the resource at the known url if it already exists, so sending the same request twice has no effect. In other words, calls to PUT are idempotent.

Anatomy of an HTTP GET request

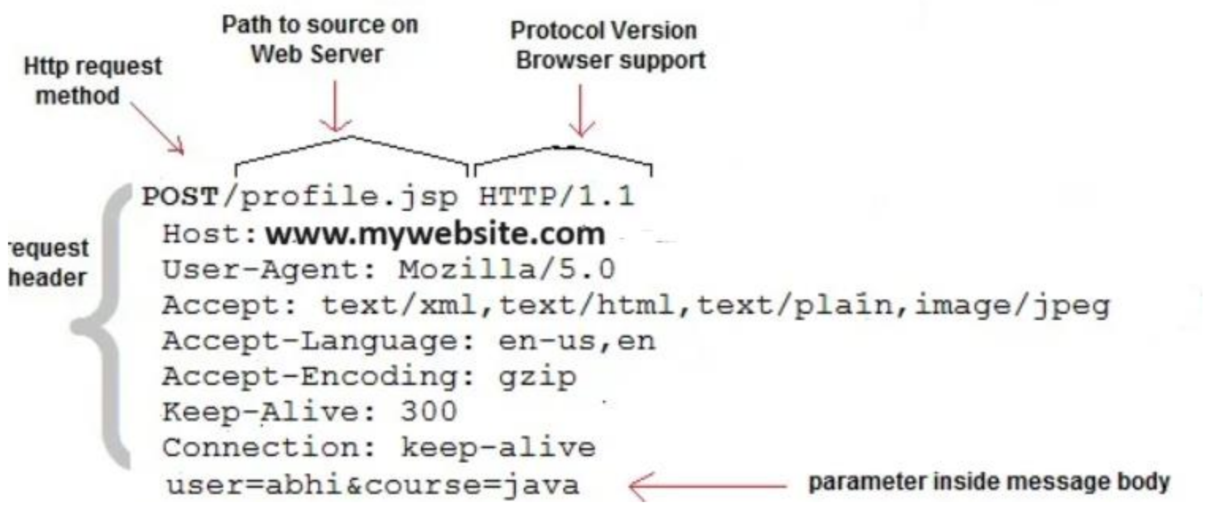
Get request contains path to server and the parameters added to it.


Http request method
Path to source on Web Server
Parameters
Protocol Version Browser support
quest
eader

```
GET/profile.jsp?user=abhi&course=java HTTP/1.1
Host: www.mywebsite.com
User-Agent: Mozilla/5.0
Accept: text/xml,text/html,text/plain,image/jpeg
Accept-Language: en-us,en
Accept-Encoding: gzip
Keep-Alive: 300
Connection: keep-alive
```

Anatomy of an HTTP POST request

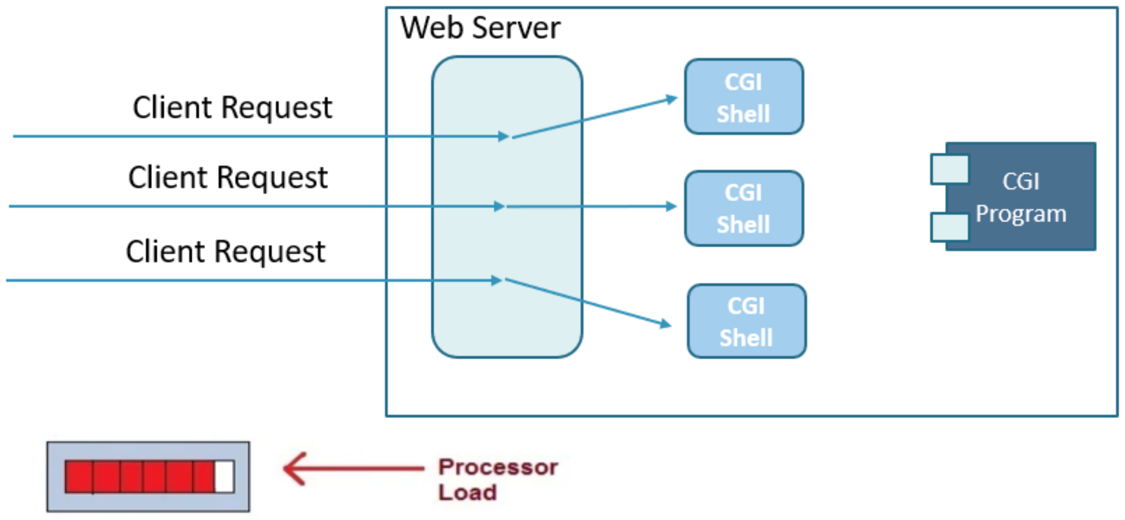
- Post requests are used to make more complex requests on the server. For instance, if a user has filled a form with multiple fields and the application wants to save all the form data to the database.
- Then the form data will be sent to the server in POST request body, which is also known as Message body.


Http request method
Path to source on Web Server
Protocol Version Browser support
request header
parameter inside message body

```
POST/profile.jsp HTTP/1.1
 Host: www.mywebsite.com
 User-Agent: Mozilla/5.0
 Accept: text/xml,text/html,text/plain,image/jpeg
 Accept-Language: en-us,en
 Accept-Encoding: gzip
 Keep-Alive: 300
 Connection: keep-alive
 user=abhi&course=java
```

CGI (Common Gateway Interface)

- Before Servlets, CGI(Common Gateway Interface) programming was used to create web applications.
- User clicks a link that has URL to a dynamic page instead of a static page.
- The URL decides which CGI program to execute.
- Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
- The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.



Advantages

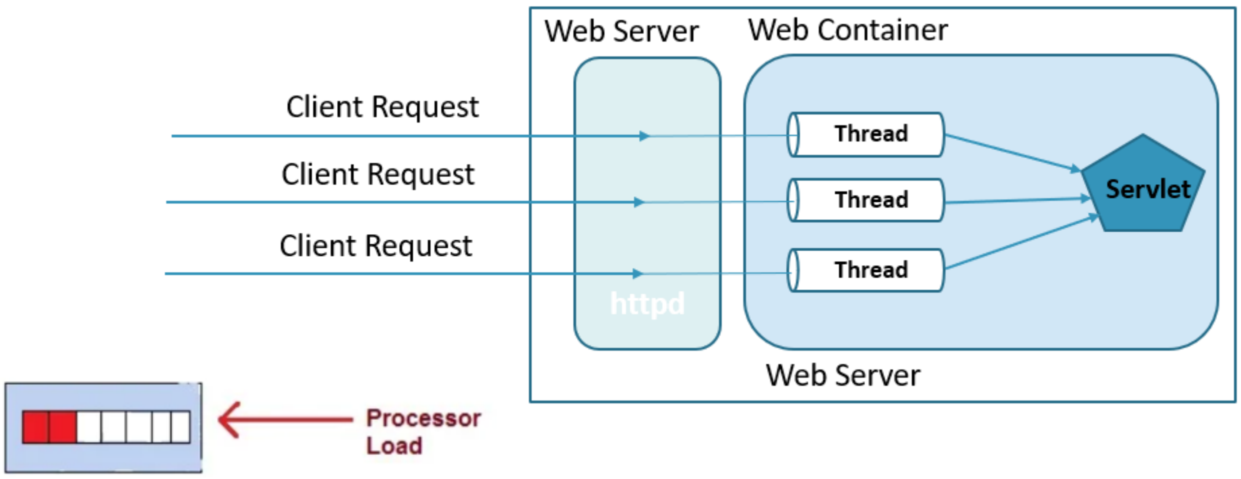
- Can be written in a variety of languages
- A CGI program with bugs does not crash the Web server
- Concurrency issues are isolated at the database

Disadvantages

- Response time of the CGI programs is high
- It is not scale well
- It is not always secure or object oriented.
- CGI code is combined with HTML, which is not a good separation of presentation and logic.

What is Servlet

- Servlets is a java-based web technology from Sun Microsystems. Servlet technology is J2EE technology.
- A servlet is a user-defined public java Class that implements javax.servlet.Servlet interface and it is managed by the servlet container (is also called Servlet Engine).
- Properties of servlet are:
  - A servlet is a Dynamic Web Resource that enhances the functionality of the webserver.
  - A servlet is a Web Server-Side Piece of Code (server-side program).
  - A servlet is a Web Component.
- Servlets use the classes within the java packages javax.servlet and javax.servlet.http.
- For each client request servlet container will generate a separate thread on the respective servlet object. If we increase the number of requests even containers will create a number of threads instead of processes so it increase the performance of the server-side application.



Advantages of Servlet.

Portability: Servlets are highly portable across operating systems and server implementations because servers are written in java and follow well known standardized APIs so they are.

Powerful: It is possible to do several things with the servlets which were difficult or even impossible to do with CGI.

Efficiency: As compared to CGI the invocation of the servlet is highly efficient. When the servlet gets loaded in the server, it remains in the server's memory as a single object instance.

Safety: As servlets are written in java, servlets inherit the strong type safety of java language. Servlets are generally safe from memory management problems because of Java's automatic garbage collection and a lack of pointers.

Integration: Servlets are tightly integrated with the server. Servlet basically uses the server to translate the file paths, perform logging, check authorization, and MIME type mapping, etc.

Extensibility: The servlet API is designed in such a way that it can be easily extensible. As it stands today, the servlet API supports HTTP Servlets, but later it can be extended for another type of servlets.

What is Servlet Container / Servlet Engine?

- A servlet container is a compiled, executable program.
- The main function of the container is to load, initialize, and execute servlets.
- The servlet container is the official Reference Implementation for the Java Servlet and Java Server Pages technologies.
- The Java Servlet and Java Server Pages specifications are developed by Sun under the Java Community Process.
- A container handles a sizable number of requests because it can hold many active servlets, listeners, etc.
- It is interesting to notice here that the container and therefore the objects during a container are multithreaded. So, each object must be thread-safe during a container because the multiple requests are being handled by the container thanks to the doorway of quite one thread to an object at a time.

Why learn Servlet?

- The primary purpose of Servlet specification is to define a strong mechanism for sending content to a client as defined by the Client/Server model.
- Servlets are most popularly used for generating dynamic content on the online and have native support for HTTP.
- A servlet is a Java programming language class that wants to extend the capabilities of servers that host applications accessed by means of a request-response programming model.
- Servlets can answer any sort of request, they're commonly wont to extend the appliance hosted by web servers.
- Java Servlet technology defines HTTP specific servlet classes for such applications.

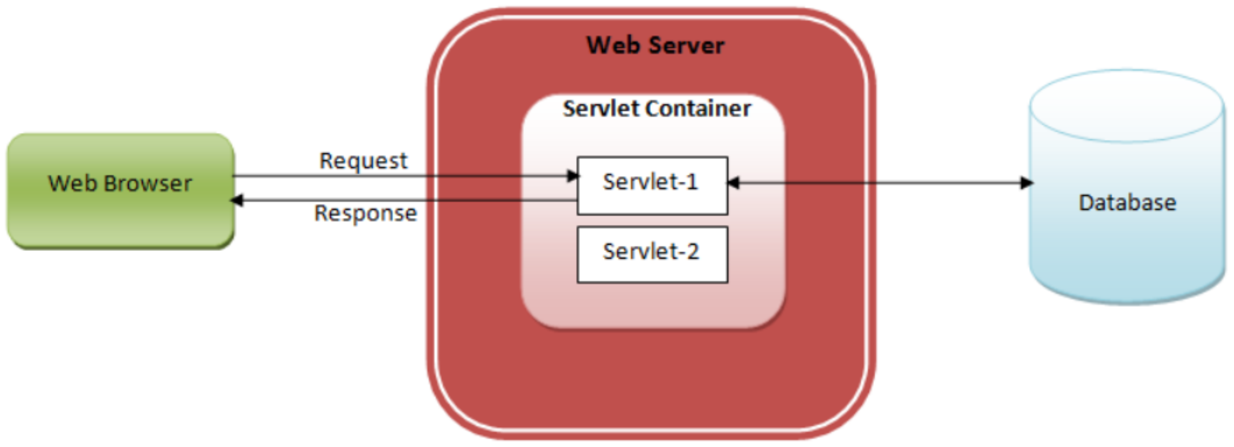
Applications of Servlet

- It helps reads the explicit data sent by the clients. It includes an HTML form on a Web page.
- It reads the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types, and compression schemes that the browser understands.
- It processes the data and generates the results. This process includes talking to a database, invoking a Web service, or computing the response directly.
- It sends explicit data to the clients. This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- It sends the implicit HTTP response to the clients. This includes telling the browsers or other client's what type of document is being returned, and other such tasks.

Servlet Architecture

- A Servlet is a class, which implements the javax.servlet.Servlet interface.
- However instead of directly implementing the javax.servlet.Servlet interface we extend a class that has implemented the interface like
    - javax.servlet.GenericServlet or javax.servlet.http.HttpServlet.

This is how a servlet execution takes place when client (browser) makes a request to the webserver.



Step 1 : Client web browser sends the request to the web server.

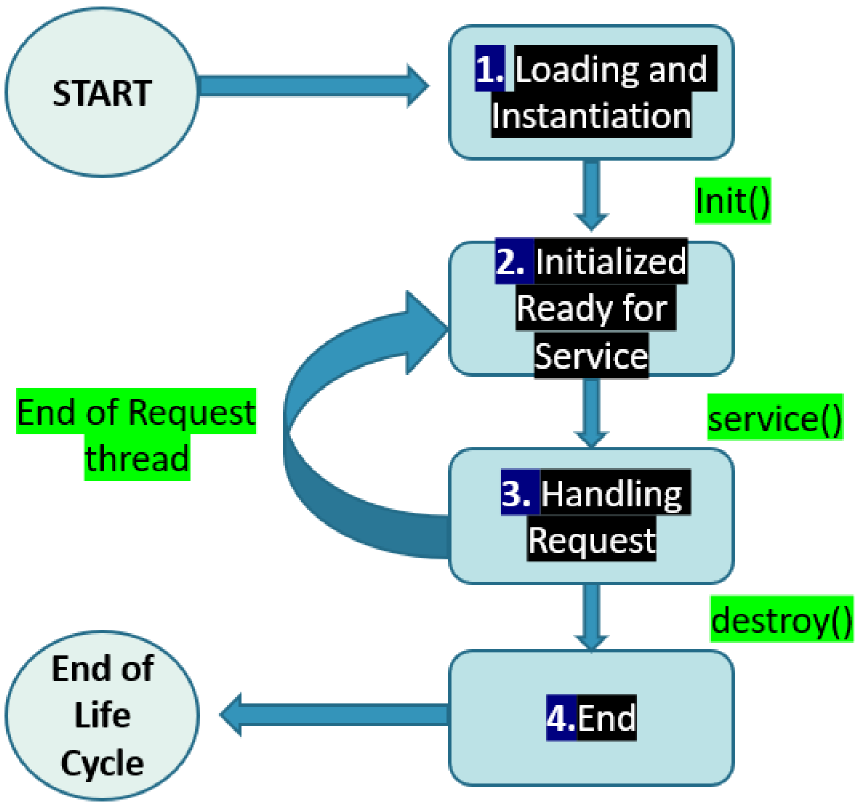
Step 2 : Web server receives the request and sends it to the servlet container. Servlet container is also called web container or servlet engine. It is responsible for handling the life of a servlet.

Step 3 : Servlet container understands the request's URL and calls the particular servlet. Actually, it creates a thread for execution of that servlet. If there are multiple requests for the same servlet, then for each request, one thread will be created.

Step 4 : Servlet processes the request object and prepares response object after interacting with the database or performing any other operations and sends the response object back to the web server.

Step 5 : Then web server sends the response back to the client.

Java Servlet Life Cycle



The Java Servlet Life cycle includes three stages right from its start to the end until the Garbage Collector clears it. These three stages are described below.

1. init()
2. service()
3. destroy()

1. init()

The init() is the germinating stage of any Java Servlet. When a URL specific to a particular servlet is triggered, the init() method is invoked.

Another scenario when the init() method gets invoked is when the servers are fired up. With every server starting up, the corresponding servlets also get started, and so does the init() method.

One important specialty of the init() method is the init() method only gets invoked once in the entire life cycle of the Servlet, and the init() method will not respond to any of the user's commands.

The init() method Syntax:

```
public void init() throws ServletException {
        //init() method initializing
}
```

2. service()

The service() method is the heart of the life cycle of a Java Servlet. Right after the Servlet's initialization, it encounters the service requests from the client end.

The client may request various services like:

- GET
- PUT
- UPDATE
- DELETE

The service() method takes responsibility to check the type of request received from the client and respond accordingly by generating a new thread or a set of threads per the requirement and implementing the operation through the following methods.

- doGet() for GET
- doPut() for PUT
- doUpdate() for UPDATE
- doDelete() for DELETE

The service() method Syntax:

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

}
```

3. destroy()

Like the init() method, the destroy() method is also called only once in the Java Servlet's entire life cycle. When the destroy() method is called, the Servlet performs the cleanup activities like, Halting the current or background threads, Making a recovery list of any related data like cookies to Disk.

After that, the Servlet is badged, ready for the Garbage collector to have it cleared.

The destroy() method Syntax:

```
public void destroy() {
   //destroy() method finalizing
}
```

What is Deployment Descriptor File?

- A web application's deployment descriptor maps the http request with the servlets.
- When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.
- The deployment descriptor should be named as web.xml. It resides in the application's WAR file under the WEB-INF/ directory.
- Root element of web.xml should be <web-app>. <servlet> element map a URL to a servlet using <servlet-mapping> element. To map a URL to a servlet, you declare the servlet with the <servlet> element, then define a mapping from a URL path to a servlet declaration with the <servlet-mapping> element.
- The <servlet> element declares the servlet class and a logical name used to refer to the servlet by other elements in the file.
- You can declare multiple servlets using the same class but name for each servlet must be unique across the deployment descriptor. The <servlet-mapping> element specifies a URL pattern and the name of a declared servlet to use for requests whose URL matches the pattern. The URL pattern can use an asterisk (*) at the beginning or end of the pattern to indicate zero or more of any character.
- The standard does not support wildcards in the middle of a string, and does not allow multiple wildcards in one pattern. The pattern matches the full path of the URL, starting with and including the forward slash (/) following the domain name. The URL path cannot start with a period (.).

Structure of deployment descriptor file – web.xml

```
<servlet>
<servlet-name>Practice1</servlet-name>
        <servlet-class>
                com.parul.controller.servlet
        </servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>Practice1</servlet-name>
        <url-pattern>/ </url-pattern>
</servlet-mapping>
```

Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

- javax.servlet
- javax.servlet.http

Some Important Classes and Interfaces of **javax.servlet**

Interfaces	**Classes**
Servlet	ServletInputStream
ServletContext	ServletOutputStream
ServletConfig	ServletRequestWrapper
ServletRequest	ServletResponseWrapper
ServletResponse	ServletRequestEvent
ServletRequestListener	ServletContextEvent
RequestDispatcher	ServletRequestAttributeEvent
SingleThreadModel	ServletContextAttributeEvent
Filter	ServletException
FilterConfig	UnavailableException
FilterChain	GenericServlet
ServletRequestListener	

Some Important Classes and Interface of **javax.servlet.http**

Interfaces	**Classes**
HttpServlet	HttpServletRequest
HttpServletResponse	HttpSessionAttributeListener
HttpSession	HttpSessionListener
Cookie	HttpSessionEvent

Methods of Servlet Interface

Interfaces	Classes
public void init(ServletConfigconfig)	It is used for initializing the servlet. It is invoked only once by the web container in a servlet life cycle.
public void service(ServletRequestreq, ServletResponse res)	It is used for providing a response to all the incoming request. It is invoked every time by the web container for each request.
public void destroy()	It is used for destroying the servlet. It is invoked only once in a life cycle of a servlet.
public ServletConfiggetServletConfig()	It is used to get the object of ServletConfig.
Public String getServletInfo()	It is used to get information about writer, copyright etc of a servlet.

HTTP Servlet

- HttpServlet is also an abstract class.
- This class gives implementation of various service() methods of Servlet interface.
- To create a servlet, we should create a class that extends HttpServlet abstract class. The Servlet class that we will create, must not override service() method.
- Our servlet class will override only the doGet() and/or doPost() methods.
- The service() method of HttpServlet class listens to the Http methods (GET, POST etc) from request stream and invokes doGet() or doPost() methods based on Http Method type.

Method	Description
public void service(ServletRequest req,ServletResponse res)	It is used for securing the service method by creating objects of request and response.
protected void service(HttpServletRequest req, HttpServletResponse res)	It is used for receiving a service method.
protected void doGet(HttpServletRequest req, HttpServletResponse res)	It is invoked by the web container and it is used for handling the GET request.
protected void doPost(HttpServletRequest req, HttpServletResponse res)	It is invoked by the web container and it handles the POST request.
protected void doHead(HttpServletRequest req, HttpServletResponse res)	It is invoked by the web container and it handles the HEAD request.
protected void doOptions(HttpServletRequest req, HttpServletResponse res)	It is invoked by the web container and it handles the OPTIONS request.
protected void doPut(HttpServletRequest req, HttpServletResponse res)	It is invoked by the web container and it handles the OPTIONS request.

protected void doTrace(HttpServletRequest req, HttpServletResponse res)	It is invoked by the web container and it handles the TRACE request
protected void doDelete(HttpServletRequest req, HttpServletResponse res)	It is invoked by the web container and it handles the DELETE request.
protected long getLastModified(HttpServletRequest req)	It is used for getting the time of last modified HttpServletRequest.

GenericServlet class

- GenericServlet is an abstract class.
- This class implements the servlet, ServletConfig and Serializable interface.
- This class provides the implementation of most of the basic servlet methods.
- The protocol of this class is independent as it can handle any type of request.

Method	**Description**
public void init(ServletConfig config)	It is used for initialization of a servlet.
public abstract void service(ServletRequest request, ServletResponse response)	It is used for providing all the services for the incoming request. When a user request then only it invokes.
public void destroy()	It is used for destroying the servlet. It is invoked only once in a life cycle of a servlet.
public ServletConfig getServletConfig()	It is used to get the object of ServletConfig
public String getServletInfo()	It is used to get information about writer, copyright etc of a servlet.
public void init()	It is a very easy and convenient method for programmers.
public ServletContext getServletContext()	It is used for getting object of a servlet
public String getInitParameter(String name)	It is used for getting all the parameter values from the given parameter names.
public Enumeration getInitParameterNames()	It is used for getting parameters which are defined in web.xml files
public String getServletName()	It is used for getting the name of a servlet object.
public void log(String msg)	It is used for writing a message in a servlet log file.
public void log(String msg, Throwable t)	It is used for writing a message in a servlet log file and stack trace.

Servlet Config

- ServletConfig is an object, it will store all the configuration details of a specific servlet, where the configuration details include the logical name of the servlet, initialization parameters, reference of ServletContext object, and so on.
- ServletConfig is an object, it will provide the entire view of a specific servlet. In the web application, the container will prepare ServletConfig objects individually to every and each servlet.
- In web application execution, the container will prepare ServletCofig object immediately after servlet instantiation and just before calling init() method in servlet initialization.
- The container will destroy the ServletConfig object just before servlet de-instantiation.
- If we declare any data in the ServletConfig object then that data is going to be shared up to the respective servlet. Due to the above reasons, the scope of the ServletConfig object is up to a particular servlet.

ServletConfig methods

- getInitParameter(java.lang.String name):
  It is used to return a String containing the value of the named initialization parameter, or null if the parameter does not exist.
- getInitParameterNames():
  It returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
- getServletContext():
  It returns a reference to the ServletContext in which the caller is executing.
- getServletName():
  It returns the name of this servlet instance.

Example:
Web.xml

```
<web-app>
  <servlet>
    <servlet-name>firstservlet</servlet-name>
    <servlet-class>com.teknobizz.controller</servlet-class>

    <init-param>
      <param-name> username </param-name>
      <param-value> jones@aol.com </param-value>
    </init-param>

    <init-param>
      <param-name> fullname </param-name>
      <param-value> Mark Jones  </param-value>
    </init-param>
  </servlet>
```

```
 <servlet-mapping>
   <servlet-name>firstservlet</servlet-name>
   <url-pattern>/</url-pattern>
 </servlet-mapping>

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

</web-app>
```

Firstservlet.java

```
protected void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException {
        PrintWriter pw=res.getWriter();
        res.setContentType("text/html");

        ServletConfig conf = getServletConfig();

        String username = conf.getInitParameter("username");
        String fullname = conf.getInitParameter("fullanme");

        pw.println("Username :” + username + “<br/>Fullname :” + fullname);
        pw.close();
}
```

Index.html

```
<form action=" Firstservlet" method="post">
     Example on ServletConfig<br>
     <input type="submit" value="Show Data">
</form>
```

Output

```
Username : jones@aol.com
Fullname : Mark Jones
```

Servlet Context

- It will manage all the context details of a particular web application, where the context details include the logical name of the web application and context parameters, and so on.
- It will provide a complete view of a particular web application. ServletContext object will be prepared by the container the moment when we start the server i.e. the time when we deploy the web application.
- It can be created and removed only by the server, not by the programmer.
- ServletContext object is created at the time of deploying the project.
- ServletContext object will be removed by the server when we un-deploy the project.
- ServletContext object will be destroyed by the container when we shut down the server i.e. the time when we un-deploy the web application. Due to the above reasons, the life of the ServletContext object is almost all the life of the respective web application.
- If we declare any data in the ServletContext object then that data will be shared with all the number of resources that are available in the present web application. Due to the above reason, ServletContext will provide more shareability.
- In web applications, the container will prepare ServletContext object irrespective of getting requests from the client.
- In the web application, ServletContext will allow both parameters and attributes data. Due to the above reason, ServletContext will allow both Static Inclusion and Dynamic Inclusion of data.

Usage of ServletContext Object

- It provides an interface between the container and the servlet.
- It is used to get configuration information from the web.xml file.
- It is used to set, get, or remove attributes from the web.xml file.
- It is used to provide inter-application communication.

ServletContext Methods

- public String getInitParameter(String name):
  It is used to get the particular context parameter value from ServletContext.
- public Enumeration getInitParameterNames():
  It is used to get all the context parameter names from ServletContext object.
- public void setAttribute(String name, Object value):
  In the web application, the ServletContext object is able to allow attributes. To set an attribute on the ServletContext object we will use this method.
- public Object getAttribute(String name):
  It is used to get an attribute from the ServletContext object.
- public void removeAttribute(String name):
  It is used to remove an attribute from the ServletContext object.
- public Enumeration getAttributeNames():
  It is used to get all the names of attributes from the ServletContext object.

Example:

Web.xml

```
<web-app>
   <context-param>
      <param-name> num1 </param-name>
      <param-value> 100 </param-value>
   </context-param>

   <context-param>
      <param-name> num2  </param-name>
      <param-value> 200 </param-value>
   </context-param>
  <servlet>
    <servlet-name>ctxservlet</servlet-name>
    <servlet-class>com.teknobizz.controller</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name> ctxservlet </servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

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

</web-app>
```

ctxservlet.java

```
protected void doPost(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
   {
      PrintWriter pw=res.getWriter();
      res.setContentType("text/html");

      ServletContext context = getServletContext();

      String num1 = context.getInitParameter("num1");
      String num2 = context.getInitParameter("num2");

      pw.println("num1 value is :" + num1+ " and num2 is :"  + num2);

     pw.close();
   }
```

```
<form action = "ctxservlet" method="post">
    Example on ServletContext<br>
    <input type="submit" value="Click Here">
</form>
```

Output

num1 value is : 100 and num2 is : 200

Differences between ServletConfig and ServletContext

ServletConfig	ServletContext
Defined in javax.servlet package.	Defined in the same javax.servlet package.
Values are placed in web.xml file.	Values are placed in web.xml file.
getInitParameter() defined ServletConfig is used to read values.	Same method getInitParameter() is also present in ServletContext interface also to read the values
When the values of <param-value> are changed, the Servlet need not be compiled again and thereby maintenance of code is easier.	The same case also with <context-param>.
No limit on the existence of <init-param> tags in <servlet>.	the web.xml fiel can have any number of <context-param> tags.
Servlet is initialized with the initialization data provided in <init-param>. This data is specific for one Servlet only.	The data provided in <context-param> tag is meant for the usage of all Servlets under execution.
Data in included within <servlet> tag.	Data is included within <context-param> tag.
It is local data for a particular Servlet.	It is global data sharable by all servlets.
Each Servlet comes with a separate ServletConfig object.	There will be only one ServletContext object available accessed by all Servlets.

ServletRequest

- When a client sends a request to the web server, the servlet container creates ServletRequest & ServletResponse objects and passes them as an argument to the servlet's service() method.
- The request object provides the access to the request information such as header and body information of request data.

Method	Description
Object getAttribute(String name)	return attribute set on request object by name
Enumeration getAttributeName()	return an Enumeration containing the names of the attributes available inthis request
int getContentLength()	return size of request body
int getContentType()	return media type of request content
String getParameter(String name)	returns value of parameter by name
Enumeration getParameterNames()	returns an enumeration of all parameter names
String[] getParameterValues(String name)	returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist
ServletContext getServletContext()	return the servlet context of current request.
String getServerName()	returns the host name of the server to which the request was sent
void removeAttribute(String name)	removes an attribute from this request
void setAttribute(String name, Object o)	stores an attribute in this request.

HttpServletRequest interface

HttpServletRequest interface adds the methods that relates to the HTTP protocol.

Methods	Description
String getContextPath()	returns the portion of the request URI that indicates the context of the request
Cookies getCookies()	returns an array containing all of the Cookie objects the client sent with this request
String getQueryString()	returns the query string that is contained in the request URL after the path
HttpSession getSession()	returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session
String getMethod()	Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
Part getPart(String name)	gets the Part with the given name
String getPathInfo()	returns any extra path information associated with the URL the client sent when it made this request.
String getServletPath()	returns the part of this request's URL that calls the servlet

Example : Cookies getCookies()

```
Cookie usercookie = new Cookie(“username”, “jones@aol.com”);
Cookie fullnamecookie = new Cookie(“fullname”, “Martin Jones”);

response.addCookie(usercookie);
response.addCookie(fullnamecookie);

Cookie[] cookies = request.getCookies();

PrintWriter writer = response.getWriter();

for (Cookie aCookie : cookies) {
   String name = aCookie.getName();
   String value = aCookie.getValue();
    writer.println(name + " = " + value);
}
```

What is Servlet Collaboration?

- The exchange of information among servlets of a particular Java web application is known as Servlet Collaboration.
- This enables passing/sharing information from one servlet to the other through method invocations.

What are the principle ways provided by Java to achieve Servlet Collaboration?

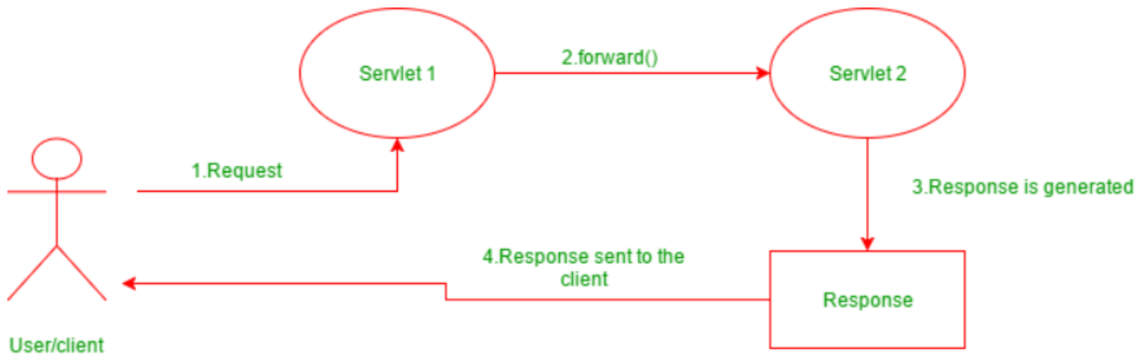
The servlet api provides two interfaces namely:

1. javax.servlet.RequestDispatcher
2. javax.servlet.http.HttpServletResponse

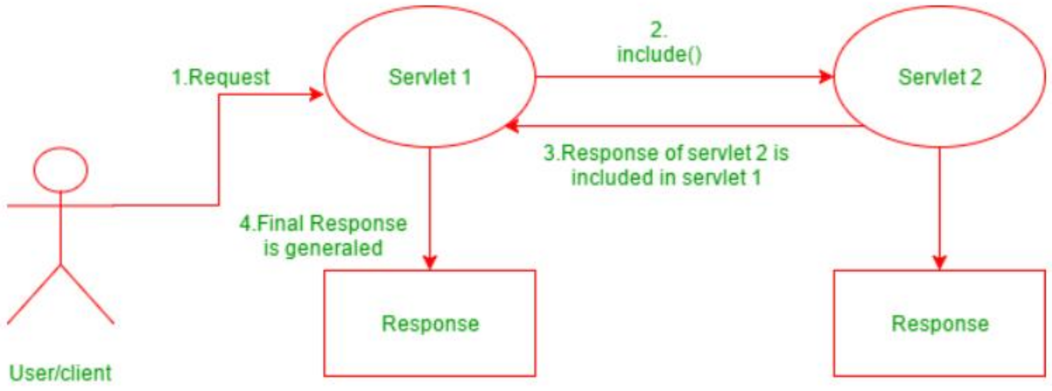
These two interfaces include the methods responsible for achieving the objective of sharing information between servlets.

Using RequestDispatcher Interface

- The RequestDispatcher interface provides the option of dispatching the client's request to another web resource, which could be an HTML page, another servlet, JSP etc.
- It provides the following two methods:
    - **public void forward(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException:**
        - The forward() method is used to transfer the client request to another resource (HTML file, servlet, jsp etc).
        - When this method is called, the control is transferred to the next called resource.
        - The following diagram explains the way it works:



- **public void include(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException:**
    - The include() method is used to include the contents of the calling resource into the called one.
    - When this method is called, the control still remains with the calling resource.
    - It simply includes the processed output of the calling resource into the called one.
    - The following diagram explains how it works:


1.Request
Servlet 1
2. include()
Servlet 2
3.Response of servlet 2 is included in servlet 1
4.Final Response is generaled
Response
Response
User/client

Which One to Use?

There is no universally correct approach — it depends on the situation.

Use forward when:

- We want to pass control to a resource in the same web app
- We want to preserve the data attributes in the original request

Use sendRedirect when:

- If we want to transfer control to another domain, then we would use sendRedirect.
- Our request is changing server state (e.g. insert/update/delete to database)

Using HttpServletResponse Interface

- The HttpServletResponse interface is entrusted with managing Http responses. To achieve servlet collaboration, it uses the following method:
- **public void sendRedirect(String URL)throws IOException;**
- This method is used redirect response to another resource, which may be a servlet, jsp or an html file. The argument accepted by it, is a URL which can be both, absolute and relative.
- It works on the client side and uses the browser's URL bar to make a request.

Session Tracking in Servlet

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver

HTTP is a "stateless" protocol, which means that each time a client requests a Web page, the client establishes a new connection with the Web server, and the server does not retain track of prior requests.

- The conversation of a user over a period of time is referred to as a session. In general, it refers to a certain period of time.
- The recording of the object in session is known as tracking.
- Session tracking is the process of remembering and documenting customer conversations over time. Session management is another name for it.
- The term "stateful web application" refers to a web application that is capable of remembering and recording client conversations over time.

Session Management is a mechanism used by the Web container to store session information for a particular user.

There are four different techniques used by Servlet application for session management are as follows:

1. Cookies
2. Hidden form field
3. URL Rewriting
4. HttpSession

Need of Session Tracking

Session tracking is a crucial aspect of web development, especially in the context of servlets and other server-side technologies, for several important reasons:

Maintaining User State: One of the primary purposes of session tracking is to maintain user state across multiple HTTP requests. Without session tracking, each HTTP request made by a user to a web application would be stateless, meaning the server wouldn't have any way to identify or remember individual users between requests.

Security: Session tracking helps enhance security. For example, after a user logs out, the session can be invalidated to prevent unauthorized access to protected resources. Additionally, session tokens can be generated securely to mitigate the risk of session hijacking. Some Best practices for securing session data and mitigating session hijacking are: Use HTTPS, Generate Strong Session IDs, Regenerate Session IDs.

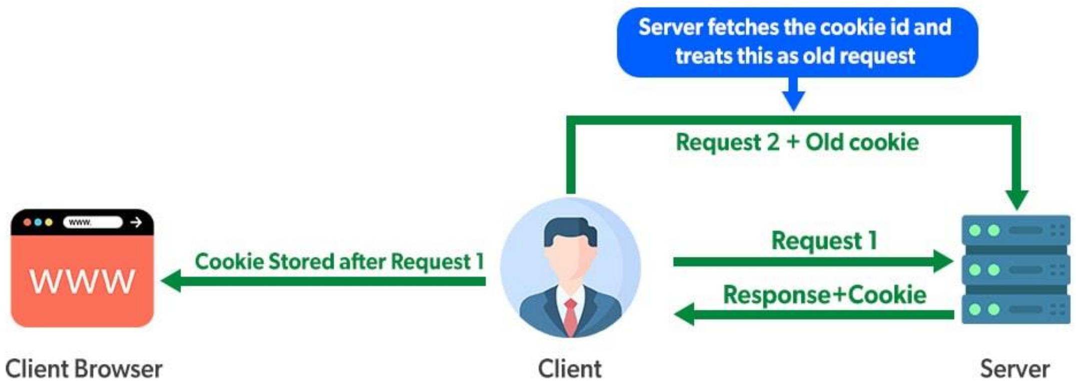
Performance Optimization: In some cases, session data can be cached or stored more efficiently, reducing the server load and improving application performance.

Tracking User Activities: Session tracking can be used to log user activities and interactions within the application. This information can be valuable for analytics, auditing, and troubleshooting purposes.

Different Techniques of Session Tracking

Cookies

- Cookies are small pieces of data sent from the server and stored on the client's browser.
- They are widely used method for session tracking. When a user visits a web application, a unique session ID is often stored in a cookie on their browser.
- Subsequent requests from the same client include this session ID in the request headers, allowing the server to associate the request with a specific session.
- Cookies are easy to implement and are supported by most web browsers.
- They are well-suited for tracking user sessions and preferences.
- There are two types of cookies.
- **Non-persistent cookie**
    - It is valid for single session only. It is removed each time when user closes the browser.
- **Persistent cookie**
    - It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.


Server fetches the cookie id and treats this as old request
Request 2 + Old cookie
Cookie Stored after Request 1
Request 1
Response+Cookie
www
WWW
Client Browser
Client
Server

Methods in Cookies

clone(): Overrides the standard java.lang.Object.clone method to return a copy of this Cookie.
getComment(): Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
getDomain(): Gets the domain name of this Cookie.
getMaxAge(): Gets the maximum age in seconds of this Cookie.
getName(): Returns the name of the cookie.
getPath(): Returns the path on the server to which the browser returns this cookie.
getSecure(): Returns true if the browser is sending cookies only over a secure protocol, or false if the browser can send cookies using any protocol.
getValue(): Gets the current value of this Cookie.
getVersion(): Returns the version of the protocol this cookie complies with.
setValue(String newValue): Assigns a new value to this Cookie.

Example

```
Cookie c = new Cookie("user_name", name);
response.addCookie(c);

Cookie[] cookies = request.getCookies();
boolean f = false;
String name = "";
if (cookies == null) {
        out.println(
            "<h1>You are new user, go to home page and submit your institute's name");
          return;
        }
else {
        for (Cookie c : cookies) {
            String tname = c.getName();
            if (tname.equals("user_name")) {
              f = true;
              name = c.getValue();
            }
          }
        }
        if (f) {
          out.println("<h1> Hello, welcome back "
                + name + " </h1>");
          out.println("<h2>Thank you!!</h2>");
        }
        else {
          out.println(
            "<h1>You are new user, go to home page and submit your institute's name");
        }
}
```

Hidden Form Fields

- Hidden form fields are HTML form elements that are not visible to the user but are included in the HTML form.
- Hidden fields can store session-related information, such as a session ID. When a user submits a form, the hidden field data is sent back to the server along with the other form data, allowing the server to maintain the session state.
- This technique is often used when cookies are disabled or when developers want to ensure session data is transmitted with each form submission.

URL Rewriting

- Url rewriting is a process of appending or modifying any url structure while loading a page.

- If your browser does not support cookies, URL rewriting provides you with another session tracking alternative. URL rewriting is a method in which the requested URL is modified to include a session ID. There are several ways to perform URL rewriting.
- In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair seperated by an equal(=) sign.

Example

```
<form method="post" action=" MyServlet">
   Name:<input type="text" name="user" /><br/>
   Password:<input type="text" name="pass" ><br/>
   <input type="submit" value="submit">
</form>

public class MyServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      String name = request.getParameter("user");
      String pass = request.getParameter("pass");

      if(pass.equals("1234"))
         response.sendRedirect("newServlet?user_name="+ name);
   }
}

public class newServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      String user = request.getParameter("user_name");
      out.println("Welcome "+user);
   }
}
```

Session

- The HttpSession object is used for session management.
- A session contains information specific to a particular user across the whole application.
- When a user enters into a website (or an online application) for the first time HttpSession is obtained via request.getSession(), the user is given a unique ID to identify his session.
- This unique ID can be stored into a cookie or in a request parameter.
- The HttpSession stays alive until it has not been used for more than the timeout value specified in tag in deployment descriptor file(web.xml).
- The default timeout value is 30 minutes, this is used if you don't specify the value in tag. This means that when the user doesn't visit web application time specified, the session is destroyed by servlet container.
- The subsequent request will not be served from this session anymore, the servlet container will create a new session.

Methods of HttpSession

public void setAttribute(String name, Object value):

Binds the object with a name and stores the name/value pair as an attribute of the HttpSession object. If an attribute already exists, then this method replaces the existing attributes.

public Object getAttribute(String name):

Returns the String object specified in the parameter, from the session object. If no object is found for the specified attribute, then the getAttribute() method returns null.

public Enumeration getAttributeNames():
Returns an Enumeration that contains the name of all the objects that are bound as attributes to the session object.
public void removeAttribute(String name):
Removes the given attribute from session.

setMaxInactiveInterval(int interval):
Sets the session inactivity time in seconds. This is the time in seconds that specifies how long a session remains active since last request received from client.

Example (index.html)

```
<form action="login">
        User Name:<input type="text" name="userName"/><br/>
        Password:<input type="password" name="userPassword"/><br/>
        <input type="submit" value="submit"/>
</form>
```

Login Servlet

```
public class login extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response){
     try{
           response.setContentType("text/html");
           PrintWriter pwriter = response.getWriter();

           String name = request.getParameter("userName");
           String password = request.getParameter("userPassword");

           pwriter.print("Hello "+name);
           pwriter.print("Your Password is: "+password);

           HttpSession session=request.getSession();
           session.setAttribute("uname",name);
           session.setAttribute("upass",password);

           pwriter.print("<a href='welcome'>view details</a>");
           pwriter.close();
    }catch(Exception exp){
       System.out.println(exp);
     }
  }
}
```

Welcome Servlet

```
public class welcome extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response){
  try{
      response.setContentType("text/html");
      PrintWriter pwriter = response.getWriter();

      HttpSession session=request.getSession(false);

      String myName=(String)session.getAttribute("uname");
      String myPass=(String)session.getAttribute("upass");

      pwriter.print("Name: "+myName+" Pass: "+myPass);
      pwriter.close();
  } catch(Exception exp){
      System.out.println(exp);
   }
  }
}
```

Unit-3 (JSP Programming)

Introduction to JSP:

Java-Server Pages (JSP) is a technology used to create web pages that can change based on user interactions or data. Think of a restaurant menu that changes daily. JSP allows us to create similar dynamic web pages on the internet. It lets us mix Java code with HTML, so the pages can show different content depending on the situation.

Advantages of JSP:

JSP has several benefits:

- **Easy to Use**: You can mix Java code directly into HTML, making it simple to create web pages that change.
- **Platform Independent**: JSP works on any server that supports Java, like a universal charger that works in any country.
- **Reusable Components**: You can create small parts of code (like building blocks) and use them in different places, which saves time and effort.
- **Separation of Concerns**: JSP separates the look of the page from the logic behind it, making it easier to manage.

Difference Between JSP and Servlet:

JSP and Servlets are both used to create dynamic web pages, but they have some differences:

- **Syntax**: JSP uses HTML with embedded Java code, like mixing chocolate chips in a cookie dough. Servlets use pure Java code, like a plain cookie.
- **Complexity**: JSP is simpler and better for creating web pages, while Servlets are more complex and used for backend processes.
- **Maintenance**: JSP separates the look from the logic, making it easier to update.
- **Compilation**: JSP pages are converted into Servlets when needed, while Servlets are compiled ahead of time.

Lifecycle of a JSP Page:

A JSP page goes through several stages, like preparing a dish in a restaurant:

1. **Translation**: The JSP file is translated into a Servlet, like reading a recipe.
2. **Compilation**: The Servlet is compiled into bytecode, like preparing ingredients.
3. **Loading**: The compiled Servlet is loaded into memory, like heating the pan.
4. **Instantiation**: An instance of the Servlet is created, like cooking the dish.
5. **Initialization**: The jspInit() method initializes the Servlet, like setting the table.
6. **Request Handling**: The jspService() method handles client requests, like serving the dish.
7. **Destruction**: The jspDestroy() method cleans up resources, like washing the dishes.

JSP Scripting Elements:

JSP scripting elements allow adding Java code directly into HTML pages, like adding spices to a recipe. There are four types:

Scriptlet Tag:

The scriptlet tag lets you add Java code within an HTML page. The code inside a scriptlet tag runs every time the page is requested.

<%

// Java code

String message = "Welcome to our website!";

out.println(message);

%>

Declaration Tag:

The declaration tag allows declaring variables and methods in a JSP page. This code is outside the main content but can be used anywhere on the page.

<%!

// Declaration

int counter = 0;

public void incrementCounter() {

counter++;

}

%>

Directives Tag:

Directives give global information about the JSP page. There are three types: page, include, and taglib. Page **4** of **8**

<%@ page language="java" contentType="text/html; charset=UTF-8" %>

<%@ include file="header.jsp" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

Expressions Tag:

The expressions tag lets you add Java expressions directly into the HTML page. The result is converted to a string and inserted into the HTML.

<%= new java.util.Date() %>

Note: JSP scripting elements provide flexibility in embedding Java code within HTML.

JSP Comments:

JSP comments are used to add notes and documentation to the JSP code, like writing notes in a recipe book. Comments are not visible to users.

<%-- This is a JSP comment --%>

JSP Implicit Objects:

JSP provides several implicit objects that are automatically available in JSP pages. These objects simplify development by providing direct access to commonly used objects, like kitchen tools in a chef's hands.

JSP Request:

The request object represents the client's request, like an order slip in a restaurant. It provides methods to access request parameters, attributes, and headers.

String username = request.getParameter("username");

JSP Response:

The response object represents the server's response, like the chef's response to the order slip. It provides methods to set response headers, status codes, and to send binary data.

response.setContentType("text/html"); response.setStatus(HttpServletResponse.SC_OK);

JSP Config:

The config object provides access to the servlet configuration information, including initialization parameters, like special instructions in a recipe.

ServletConfig config = getServletConfig();

String param = config.getInitParameter("paramName");

JSP Application:

The application object represents the servlet context, like the kitchen environment in a restaurant. It provides methods to interact with the web application's environment.

ServletContext context = getServletContext();

String realPath = context.getRealPath("/index.jsp");

JSP Session:

The session object represents the client's session, like a regular customer's preferences. It provides methods to store and retrieve session attributes.

HttpSession session = request.getSession();

session.setAttribute("username", "John");

JSP Out:

The out object is used to send content to the client, like a waiter serving a dish to a customer. It provides methods to write text, HTML, and binary data.

out.println("Welcome to our restaurant!");

JSP Page Context:

The pageContext object provides access to all the implicit objects and serves as a container for page attributes, like a manager overseeing the kitchen.

pageContext.setAttribute("attributeName", "value");

JSP Exception:

The exception object is used to handle exceptions in JSP pages. It is available only in error pages, like a manager dealing with a complaint.

<%@ page isErrorPage="true" %>

<%= exception.getMessage() %>

JSP Action:

JSP actions are XML tags that perform specific tasks in JSP pages, like specific instructions in a recipe. Actions can include content, forward requests, and interact with JavaBeans.

<jsp:include page="header.jsp" />

<jsp:forward page="login.jsp" />

<jsp:useBean id="user" class="com.example.User" /> Page **7** of **8**

Expression Language:

Expression Language (EL) simplifies access to data stored in JavaBeans components and other objects. EL expressions are enclosed in ${} and can be used directly in JSP pages.

${user.username}

${sessionScope.username}

Note: EL simplifies data access in JSP pages.

Introduction to JSTL:

Pages Standard Tag Library (JSTL) is a collection of useful JSP tags that encapsulate core functionality common to many JSP applications. JSTL tags can be used for iteration, conditionals, XML manipulation, and internationalization.

Note: JSTL provides a standard set of tags to simplify JSP development.

JSTL Core Tags:

JSTL core tags provide basic functionality such as iteration, conditionals, and URL management. The core tag library is identified by the prefix c.

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:forEach var="item" items="${items}">

${item.name}

</c:forEach>

<c:if test="${user.loggedIn}">

Welcome, ${user.username}!

</c:if>