Q.1 GET requests a representation of the specified resource. Note that GET should not be used for operations that cause side-effects, such as using it for taking actions in web applications. One reason for this is that GET may be used arbitrarily by robots or crawlers, which should not need to consider the side effects that a request should cause.

and

POST submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both.

Q.2

A cookie is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object

response.addCookie(ck);//adding cookie in the response

Q.3

Hidden from Fields: This is a browser independent approach of maintaining state between requests in a web applicaion.In this approach information to be persistent is stored in invisible text fields which are added to the response page.When a request is submitted from the response page value of invisible text fields is submitted as request parameters.

Disadvantage

This approach can be used if input forms are used for submitting request.

Only textual informaton can be persistent.

Url Rewriting: This is another way to support state tracking. With URL rewriting, the parameter that we want to pass back and forth between the Web browser and client is appended to the URL. URL rewriting is the lowest common denominator of session tracking, and is used when a client does not accept cookies. We modified the CookieServletto implement the same state tracking mechanism technique, but by using URL rewriting.URL Rewriting can be used in place where we don't want to use cookies.The URL Rewriting is used for maintaining the session.

Q.4 JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

Making a connection to a database.

Creating SQL or MySQL statements.

Executing SQL or MySQL queries in the database.

Viewing & Modifying the resulting records.

Q.5

An application programming interface (API), in the context of Java, is a collection of prewritten packages, classes, and interfaces with their respective methods, fields and constructors. Similar to a user interface, which facilitates interaction between humans and computers, an API serves as a software program interface facilitating interaction.

In Java, most basic programming tasks are performed by the API’s classes and packages, which are helpful in minimizing the number of lines written within pieces of code.

Q.2

A) public static void MyGETRequest() throws IOException {

URL urlForGetRequest = new URL("https://jsonplaceholder.typicode.com/posts/1");

String readLine = null;

HttpURLConnection conection = (HttpURLConnection) urlForGetRequest.openConnection();

conection.setRequestMethod("GET");

conection.setRequestProperty("userId", "a1bcdef"); // set userId its a sample here

int responseCode = conection.getResponseCode();

if (responseCode == HttpURLConnection.HTTP\_OK) {

BufferedReader in = new BufferedReader(

new InputStreamReader(conection.getInputStream()));

StringBuffer response = new StringBuffer();

while ((readLine = in .readLine()) != null) {

response.append(readLine);

} in .close();

// print result

System.out.println("JSON String Result " + response.toString());

//GetAndPost.POSTRequest(response.toString());

} else {

System.out.println("GET NOT WORKED");

}

}

Q.2

b) 1) JDBC-ODBC bridge driver

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

2) Native-API driver

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

3) Network Protocol driver

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

4) Thin driver

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

Q.3

A)

1.  ODBC is procedural and JDBC is object oriented.

2. ODBC is used to provide connection between front-end application(other than java) and back-end (database like ms-access) and JDBC is used to provide connection between JAVA and database(oracle,sybase,DB2,ms-access).

3. ODBC is for Microsoft and JDBC is for java applications.

4. ODBC can't be directly used with Java because it uses a C interface.

5. ODBC makes use of pointers which have been removed totally from java.

Q.3

b) RequestDispatcher in Servlet

[RequestDispatcher Interface](https://www.javatpoint.com/requestdispatcher-in-servlet)

[Methods of RequestDispatcher interface](https://www.javatpoint.com/requestdispatcher-in-servlet#rdmethod)

[forward method](https://www.javatpoint.com/requestdispatcher-in-servlet#rdforward)

[include method](https://www.javatpoint.com/requestdispatcher-in-servlet#rdinclude)

[How to get the object of RequestDispatcher](https://www.javatpoint.com/requestdispatcher-in-servlet#rdhow)

[Example of RequestDispatcher interface](https://www.javatpoint.com/requestdispatcher-in-servlet#rdex)

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

How to get the object of RequestDispatcher

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:

Syntax of getRequestDispatcher method

public RequestDispatcher getRequestDispatcher(String resource);

Example of using getRequestDispatcher method

RequestDispatcher rd=request.getRequestDispatcher("servlet2");

//servlet2 is the url-pattern of the second servlet

rd.forward(request, response);//method may be include or forward

Q.4

A)

index.html

<form action="servlet1" method="post">

Name:<input type="text" name="username"/><br/><br/>

Password:<input type="password" name="userpass"/><br/><br/>

<input type="submit" value="login"/>

</form>

FirstServlet.java

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class FirstServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

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

    String p=request.getParameter("userpass");

    if(LoginDao.validate(n, p)){

        RequestDispatcher rd=request.getRequestDispatcher("servlet2");

        rd.forward(request,response);

    }

    else{

        out.print("Sorry username or password error");

        RequestDispatcher rd=request.getRequestDispatcher("index.html");

        rd.include(request,response);

    }

    out.close();

    }

}

 LoginDao.java

import java.sql.\*;

public class LoginDao {

public static boolean validate(String name,String pass){

boolean status=false;

try{

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

Connection con=DriverManager.getConnection(

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

PreparedStatement ps=con.prepareStatement(

"select \* from userreg where name=? and pass=?");

ps.setString(1,name);

ps.setString(2,pass);

ResultSet rs=ps.executeQuery();

status=rs.next();

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

return status;

}

}

WelcomeServlet.java

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class WelcomeServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)

    throws ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

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

    out.print("Welcome "+n);

    out.close();

    }

}

Q.4

b) @WebServlet("/login")

public class LoginServlet extends HttpServlet {

@EJB

private UserService userService;

@Override

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

request.getRequestDispatcher("/WEB-INF/login.jsp").forward(request, response);

}

@Override

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

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

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

User user = userService.find(username, password);

if (user != null) {

request.getSession().setAttribute("user", user);

response.sendRedirect(request.getContextPath() + "/home");

} else {

request.setAttribute("error", "Unknown login, try again");

doGet(request, response);

}

}

}

Then you can check for that in a login [filter](https://stackoverflow.com/tags/servlet-filters/info) like below:

@WebFilter("/\*")

public class LoginFilter implements Filter {

@Override

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws ServletException, IOException {

HttpServletRequest request = (HttpServletRequest) req;

HttpServletResponse response = (HttpServletResponse) res;

HttpSession session = request.getSession(false);

String loginURI = request.getContextPath() + "/login";

boolean loggedIn = session != null && session.getAttribute("user") != null;

boolean loginRequest = request.getRequestURI().equals(loginURI);

if (loggedIn || loginRequest) {

chain.doFilter(request, response);

} else {

response.sendRedirect(loginURI);

}

}

// ...

}

public class RedirectFilter implements Filter {

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

HttpServletRequest req=(HttpServletRequest)request;

//check if "role" attribute is null

if(req.getSession().getAttribute("role")==null) {

//forward request to login.jsp

req.getRequestDispatcher("/login.jsp").forward(request, response);

} else {

chain.doFilter(request, response);

}

}

}