ASSIGNMENT

1. Differentiate ServletContext and ServeletConfig in Servlet with proper example.

Soln :

☆ Servlet Config :-

→ Servlet Config available in javax.servlet.*; package.

→ Servlet Config object is one per servlet class.

→ Object of Servlet Config will be created during initialisation process of the servlet.

→ The config object is public to a particular servlet only.

→ Scope: As long as servlet is executing, ServletConfig objects will be available it will be destroyed once the servlet execution is completed.

→ We should give request explicitly, in order to create Servelet Config object for the first time.

→ In web.xml - <init-param> tag will be appear under <servlet-class> tag.

ex:

```
<servlet>
  <servlet-name> ServletConfigTest
                </servlet-name>
  <servlet-class> package
                </servlet-class>

  <init param>
    <param name> Topic </param-name>

    <param-value> Difference
        between Servletconfig and
            ServletContext
    </param-value>
  </init-param>
</servlet>
```

☆ ServletContext :

→ ServletContext available javax.servlet.* package;

→ ServletContext object is global to entire web application

→ Object of ServletContext will be created at the time of web-application deployment.

→ Scope : As long as web application is executing, Servlet context object will be available and will be destroyed once the application is removed from the server.

→ Servlet context object will be available even before giving first request in. web.xml - <context-param> tag will appear under <web-app> tag.

ex :

```
<context-param>
    <param-name> Global variable
        </param-name>

    <param-value> package
        </param-value>

</context-param>
```

Servlet Config	Servlet Context
→ Servlet Config object represent single Servlet	→ It represent whole web application running on particular JVM and common for all.
→ Its like local parameter associated with particular servlet.	→ Its like global parameter associated with whole app.
→ Its a name value pair defined inside the servlet section of web.xml file so it has servlet wide scope.	→ Servlet Context has application wide scope so define outside of servlet tag in web.xml file.
→ getServletConfig() method is used to get the config object.	→ getServletContext() method is used to get the context object.
→ Example : Shopping cart of user is a specific to particular user so here Servlet config is used.	→ Ex: To get MIME type of a file or application Section related info is stored using servlet context object.

Q2: What do you mean by Session? what are different tracking mechanisms.

Soln :- Session :

→ A session is a conversation between the server and a client. A conversation consists of a series of continuous requests and responses.

→ When there is a series of continuous requests and responses from the same client to a server; the server cannot identify from which client it is getting requests. Because HTTP is a stateless protocol.

→ There are 5 different tracking mechanism to achieve this which are:

1. User Authorization
2. Hidden fields.
3. URL rewriting
4. Cookies
5. Session tracking API.

→ The first four methods are traditionally used for session tracking in all the server side technologies. The session tracking API method is provided by underlying technology. Session tracking API is built on top of first four methods.

1. User Authorization:

→ User can be authorized to use the web application in different ways. Basic concept is that the user will provide username and password to login to the application. Based on that user can be identified and the session can be maintained.

2. Hidden Fields:

<INPUT TYPE = "hidden" Name = "technology" value = "servlet">

Hidden fields like above can be inserted in the webpages and information can be sent to server for session tracking. These fields are not visible directly to the user, but can be viewed using view source option from the browsers. This type doesn't need any special configuration from the browser of server and by default available to use for session tracking. This cannot be used for session tracking when the conversation included static resources like HTML Pages.

3. URL Rewriting :

Original URL : http://server:port/servlet/ServletName

Rewritten URL : http://server:port/servlet/ServletName?sessionid=1456

→ When a request is made, additional parameter is appended with URL. In general added additional parameter will be sessionid or sometimes the userid. It will suffice to track the session. This type of session tracking doesn't need any special support from the browser.

→ We need to keep track of the parameter as a chain link until the conversation completes and also should make sure that the parameter don't clash with other application parameters.

4. Cookies :

→ Cookies are the mostly used technology for session tracking. Cookie is a key value pair of information sent by the server to browser. This should be saved by browser in its space in client computer.

→ Whenever the browser sends a request to that server it sends cookie along with it. Then server can identify the client using the cookie.

→ Shipplt :

```
Cookie cookie = new Cookie("userID",
                           "7456");
res.addCookie(cookie);
```

→ Session tracking is easy to implement and maintain using cookies.

5 Session Tracking API :-

→ Session tracking API is built on top of first four methods. This is in order to help the developer to minimize the overhead session tracking. This type of session tracking is provided by the underlying technology.

→ This is best of all methods because all the management and errors related to session tracking will be taken care by container itself. Every client of server will be mapped with a javax.servlet.http.HttpSession object. Java Servlets can use the session object to store and retrive java objects across sessions.

Teacher's Signature

3. Compare UDP based and TCP based socket programming.

Soln Socket:

A socket is one endpoint of a two-way communication link between two programs running on different computers on a network. A socket is bound to a port number so that transport layer can identify the application that data is destined to be sent to.

⇒ UDP Based Socket programming:

UDPServer.java

```
package com.package.java.socket;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class UDPServer {

    private DatagramSocket udpSocket;
    private int port;

    public UDPServer(int port) throws
            SocketException, IOException {
```

```
        this.port = iport;
        this.udpsocket = new DatagramSocket
                                (this.port);
    }

    private void listen() throws Exception{
        System.out.println("Running server at"
    + InetAddress.getLocalHost()+"--");
        String msg;
        while(true){
            byte[] buf = new byte[256];
            DatagramPacket packet = new
                    DatagramPacket(buf,
                buf.length);
            udpSocket.recieve(packet);
            msg = new String(packet.getData().
                                trim();
            System.out.println(
                "Message from " +
            packet.getAddress().getHostAddress()
                + ": " + msg);
        }
```

```
public static void main (String [] args)
                throws Exception {

    UDPserver client = new UDPserver
                (Integer.parseInt(args[0]));
    client.listen();

  }

}
```

→ UDP Client Socket

```
package com.package.java.socket;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;
import java.net.DatagramSocket;
import java.net.*;

public class UDPClient {

    private DatagramSocket udpSocket;
    private InetAddress serverAddress;
    private int Port;
    private Scanner scanner;

    private UDPClient (String destinationAdds,
                int port)
```

```
throws IOException{
    this.serverAddress = InetAddress.
                getByName (destinationAddr);
    this.port = port;
    udpSocket = new DatagramSocket
                (this.port);
    scanner = new Scanner (System.in);
}

public static void main (String [] args)
        throws NumberFormatException,
                IOException {
    UDPClient sender = new
                UDPClient (args[0],
        Integer.parseInt (args[1]));
    System.out.println
        ("Running UDP Client at " +
        InetAddress.getLocalHost() + ");
    sender.start();
}

private int start() throws
        IOException {
    String in;
```

```
while (true) {
    in = scanner.nextLine();
    DatagramPacket p = new DatagramPacket(
        in.getBytes(), in.getBytes().length,
        serverAddress port);
    this.udpSocket.send(p);
  }
 }
}
```

→ TCP based Server programming :-

```
public class TCPServer {
    private ServerSocket serverSocket;
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;
```

```
public void start (int port){

    serversocket = new ServerSocket (port);
    clientSocket = serverSocket.accept();
    out = new PrintWriter
          (ClientSocket.getOutputStream(),
                    true);
    in = new BufferedReader (new
    InputStreamReader
          (ClientSocked.getInputStream()));

    String greeting = in.readLine();
    if ("Hello server".equals(greeting))
      {
          out.println ("hello client");
      }
    else{
        out.println ("Unrecognised
                    message");
        }
    }

public void stop(){
    in.close();
    out.close();
    clientSocket.close();
    serverSocket.close();
}
```

```
public static void main (String[] args){
    GServer server = new GServer();
    server.start(6666);
  }
}
```

⇒ TCP Client Socket

```
public class GClient{

    private Socket clientSocket;
    private PrintWriter out;
    private BufferReader in;

    public void startConnection(String ip,
                                int port){
        clientSocket = new Socket(ip, port);

        out = new PrintWriter
            (clientSocket.getOutputStream(),
                                true);
        in = new BufferReader(new
            InputStreamReader(
                clientSocket.getInputStream()));

    }
    public String sendMessage(String msg)
    {
        out.println(msg);
        String resp = in.readLine();
        return resp;
}
```

Teacher's Signature.....

```
public void stopConnection () {

    in.close();
    out.close();
    clientSocket.close();

  }

}
```

→ Clearly here TCP listen for connections on it socket while UDP just don't do that.

→ TCP accepts conneitions address from client before exchanging information while UDP just accepts information and stores
" data, client_addr = sockfd.recvfrom"
the IP address from which it recieved information.

→ The receiving function.
TCP uses recv(1024) to recv data "1024" while, UDP uses recv from (1024) basically because the address from which data was recieved also needs to be store.

4. Explain following tags in JSF.

(1) h:inputText :-

→ The JSF <h:inputText> tag is used to render an input field on the webpage. It is used within <h:form> tag to declare input field that allows user to input data.

→ The value attribute refers to name property of a managed bean named User. This property holds the data for name component.

→ <h:inputText id = "username" value = "#{user.name}" label = "username" maxlength = "10"

size = "15" alt = "username" autocomplete = "off" readonly = "false" required = "true"

requiredMessage = "Username is required". style = "color:red" accesskey = "q" >

</h:inputText>

(2.) h:outputText:

→ It is used to render a plain text. If the "styleClass", "style", "dir" or "lang" attributes are present, render a "span" element. If the "styleClass" attribute is present, render its value as value of "class" attribute.

→ <h:outputText value="hello">
</h:outputText>

(3.) h:form:

→ The <h:form> tag represents an input form. It includes child components that can contain data which is either presented to the user or submitted with form. It can also include HTML markup to layout the components on page.

→ <h:form id="user-form">

<h:outputLabel for="username">UserName
</h:outputLabel>
<h:inputText id="username" value
="#{user.name}" required
="true"
requiredMessage="Username is
required" />
.

```
<h:commandButton id = "submit-button"
    value = "Submit" action = "response.xhtml"
</h:form>
```

(4) h:commandButton.:

→ It creates a submit button and is used submit an application form.

Syntax: <h:commandButton> </h:commandButton>

→ eg:
```
<h:form id = "user-form">
<h:outputLabel for = "username">
    UserName </h:outputLabel>

<h:inputText id = "username"
    value = "#{user.name}"
    required = "true"
    requiredMessage = "username
        is required" /><br/>

<h:commandButton id = "submit-button"
    value = "Submit"
    action = "response.xhtml" />

</h:form>
```

(5.) h:inputTextarea

→ JSF renders this as an HTML "textarea" element. It allows a user to enter a multiline string.

→ <h:inputTextarea id = "text-area-id".
value = "#{user.address}"
required = "true"
requiredMessage = "Address required"
cols = "50" rows = "10">

</h:inputTextarea>

(6.) h:inputSecret

→ It is a standard password field which accepts one line of texts with no spaces and displays it as a set of asterisks as it is entered. In other words, it is used to create a HTML password field which allows a user to input a string without the actual string appearing in field.

→ <h:inputSecret value = "#{user.password}"
maxlength = "10" size = "15"
required = "true".
requiredMessage = "Password req."/>
</h:inputSecret>.

(7) h:graphicImage

→ JSF renders an HTML element "img" tag. This tag is used to render an image on web page.

→ <h:graphicImage id = "image-id"
name = "user-image"
url = "#{user.fileLocation()}"
height = "50px" width = "50px"
alt = "Image not found">

</h:graphicImage>.

5. What is HQL? How does it different from SQL? List its advantages.

Sol^n → HQL is an abbreviation for hibernate query language. Hibernate is platform to connect the traditional databases to object-oriented language. The query language in hibernate is similar to SQL in traditional RDBMS except for the fact we use an entity in HQL instead of tables. It is written embedded in Java and various functions from Java.Library.

→ It can be called as an object oriented language embossed with SQL query statements. It is a flexible and user-friendly language having its own syntax and grammer to retrive, store, update information. from database.

→ HQL is an XML file format linked java from the frontend to the database in the backend. The SQL queries which we fire in the database directly using sql queries can be written in hql as well.

→ The HQL has its syntax where we can write the query and then that query is converted into SQL statements which can be understood by database.

→ ex: String hqlquery = "FROM Test";
Query q = session.createQuery
(hqlquery);
List display = q.list();
AS clause: From eg Test AS T or
From Test AS T.

Parameters	SQL	HQL.
→ Full Form	Structured Query Language	Hibernate Query Language.
→ Type of Programming language.	Traditional Query Language.	JAVA based OOP query language.
→ Concerns	It pertains to the relation bet^n two tables or columns	It pertains two the relation bet^n two objects.
→ User-friendly	Offers complex interface to new users	Provides user-friendly interface.
→ Features	Uses tables and columns	Uses JAVA classes and variables.
→ Interaction with database.	Directly interact with the database.	Uses the 'Hibernate' interface to interact with database.
→ Speed	Native SQL is usually faster	Non-native HQL is usually slower since its runtime based on mapping, but speed can be increased

Teacher's Signature.....

→ Advantages of HQL:

→ The coder does not have any obligation to learn the SQL language.

→ HQL is object oriented and its performance is good when we link our front-end

→ HQL has caching memory and thereby improves speed.

→ HQL supports popular features of OOPs concept like polymorphism, inheritance and association.

6. Write a web application using JSP to get an integer N (N > 0) and display all prime number upto N.

Soln:

```
<!DOCTYPE html>
<html>
  <head>
    <meta content = "text/html" >
    <title> JSP Prime Number </title>
  </head>
  <body>

    <%
```

```
int n = Integer.parseInt(request.
        getParameter("id"));
int i, number = 1, count;

out.println("Prime Number : ");
while (number <= n)
{
    count = 0;
    i = 2;
    while (i <= number/2)
    {
        if (number % i == 0)
        {
            count++;
            break;
        }
        i++;
    }
    if (count == 0 && number != 1)
    {
        out.print(number + " ");
    }
    number++;
}
%>
</body>
</html>
```

7. Write a programme to demonstrate Hibernate Table Per Hierarchy using annotation.

Soln: Person.java

```
import javax.persistence.*;

@Entity
@Table(name = "person")
@Inheritance(strategy = InheritanceType.
                        SINGLE_TABLE)
@DiscriminatorColumn
        (name = "type", discriminatorType
        = DiscriminatorType.STRING)
@DiscriminatorValue(value = "person")


public class Person {
@Id
@GeneratedValue(strategy = GenerationType
                        .AUTO)
@Column(name = "id")
private int id;

@Column(name = "name")
private String name;

public void setId(int id) {
        this.id = id;
}
```

```
public void setName (String name) {
    this.name = name;
}

public int getId () {
    return this.id;

}
    public String getName () {
        return this.name;
    }

}
```

→ Professor.java

```
import javax.persistance.*;

@Entity
@DiscriminatorValue (value = "professor")
public class Professor extends Person {

    @Column (name = "prof_id")
    private string prof_id;

    @Column (name = "subject")
    private string subject;
```

```
    public void setProfId(String prof_id){
        this.prof_id = prof_id;
    }

    public void setSubject(String subject){
        this.subject = subject;
    }

    public String getProfId(){
        return this.prof_id;
    }

    public String getSubject(){
        return this.subject;
    }

}
```

→ Student.java

```
importance javax.persistence.*;

@Entity
@DiscriminatorValue(value="Student")

public class Student extends Person{

    @Column(name="clg_id")
    private String clg_id;
```

```
@Column (name = "branch")
private String branch;

public void setClgId (String clg_id){
    this.clg_id = clg_id;
}

public void setBranch (String branch){
    this.branch = branch;
}

public String getClgId (){
    return this.clg_id;
}

public String getBranch (){
    return this.branch;
}

}
```

→ StorePata.java

```
import java.util.Scanner;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.
                    StandardServiceRegistry;
import org.hibernate.boot.registry.
                    StandardServiceRegistryBuilder;

public class StorePata{

    public static void main (String arg[]){

        StandardServiceRegistry ssr =
            new StandardServiceRegistryBuilder().
                configure("hibernate.cfg.xml")
                .build();

        Metadata meta = new MetadataSources
            (ssr).getMetadataBuilder().build();

        SessionFactory factory = meta.getSession
                                FactoryBuilder().build();
        Session session = factory.openSession();
        Transaction t = session.beginTransaction();
```

```
Person p = new Person();
p.setName("Bill Gates");

Student s = new Student();
s.setName("Elon Musk");
s.setClgId("1BIT441");
s.setBranch("IT");

Professor prf = new Professor();
prf.setName("Richard Feynman");
prf.setProfId("HA07");
prf.setSubject("Physics");

session.persist(p);
session.persist(s);
session.persist(prf);

t.commit();
System.out.println("All record stored
                    Successful");

session.close();
factory.close();

}
}
```

⇒ hibernate.cfg.xml

```
<!DOCTYPE hibernate mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 5.3//EN
    "http.org//www.hibernate.org/dtd/
            hibernate-configuration">
<hibernate-configuration>
<session-factory>
    <property name = "hbm2ddl.auto">
            update </property>
    <property name = "dialect">
            Org.hibernate.dialect.MySQL5Dialect
            </property>
    <property name = "hibernate.connection.url">
            jdbc:mysql://localhost:3306/college
            </property>
    <property name = "hibernate.connection.
            driver_class">
            com.mysql.jdbc.Driver</property>
    <property name = "hibernate.connection.
            username">root</property>
```

Date : / /

```
<property name = "hibernate.connection.
    password "></property>

<mapping class = "Person" />
<mapping class = "Student" />
<mapping class = "Professor" />

</session-factory>
</hibernate-configuration>
```

Teacher's Signature.....