T.Y. B.Sc. (IT) : Sem. V

Enterprise Java

Time : 2½ Hrs.] Prelim Question Paper Solution **[Marks : 75**

Q.1Attempt the following (any THREE) [15]

Q.1(a) What is an Enterprise Application? Explain architecture of an enterprise application. [5]

Ans.: An enterprise applications are complex, distributed, scalable, component – based and mission critical business application.

- It can be used on variety of platforms across internet, or extranets.

Architecture of an enterprise application :

- One of the key objectives of the Java EE platforms is to provide a standard environment to develop Enterprise Applications.
- It is a distributed, multi-tiered, application model which can be described as an enterprise application.
- Application logic is divided into components according to functions and the various application component put together make up the enterprise application.
- These components are physically installed on different physical location, depending on the tier in the multi–tired environment to which the application component belongs.
- All components work together in harmony across the enterprise.
- Enterprise architecture is divided into the following tiers:
  - Client tier services which run on the client machine.
  - Web tier services run on the server
  - Business tier services run on the server.
  - Enterprise information system [EIS] tier software run on the EIS server.

Q.1(b) What are different technologies provided by Java EE platform? [5]

Ans.: Following are different technologies provided by the Java EE platform :

(a) Web Application Technologies

(i) Java Servlet : A servlet is an application present inside a web server, it accept request from client, dynamically generate the response & then send the response to the client. It can be done using various servlet API (classes & interfaces).

(ii) Java Server Pages : A JSP is present inside web server, which is mainly used to embed. Java is HTML. It is more easier for those web designers who are more comfortable in scripting languages likes HTML.

(iii) Java Server Faces :JSP is an user interface framework for building web applications. It is server side, user interface component framework for building web applications.

(b) Enterprise Applications Technologies

(i) Enterprise Java Beans (EJB) : EJB are scalable, transactional, reusable, secure building blocks which are present pride web server & can be used by various web applications like servlet JSP etc.

(ii) Java persistence API (JPA) : JPA is a java standard based API for object relational mapping, and it is a solution for persistence. Persistence uses an ORM approach to bridge the gap between an object oriented model & relational database.

(c) Web Services Technologies

(i) SOAP and RESTFUL : The Java API for RESTFUL web services defines APIs for the development of such services built according to the representational state transfer [KEST] architectural style.

(ii) Java API for XML registries [JAXR] : JAXR allow accessing business and general purpose registries the web.

(d) Security Technologies

(i) Java authorization service provider contact for container [JACC] : JACC defines a contract between a Java EE application server and an authorization policy provider. All Java EE containers support the contract.

(ii) Java Authentication since provider interface for containers [JASPIC] : JASPIC specification defines a service provider interface [SPI] by which authentication providers that implement message authentication mechanism may be integrated indicator server message processing containers or relations.

Q.1(c) Explain life cycle of servlet with diagram. **[5]**

Ans.: **Life cycle of a servlet**

The entire life cycle of servlet consists of following phases :

1. Creation and Loading :

- In this a web server creates a servlet and load it inside the web container (servlet pool).

2. Instantiation :

- In this web server creates the instance i.e. object of a servlet, so that we can call its methods using that object.

3. Initialization :

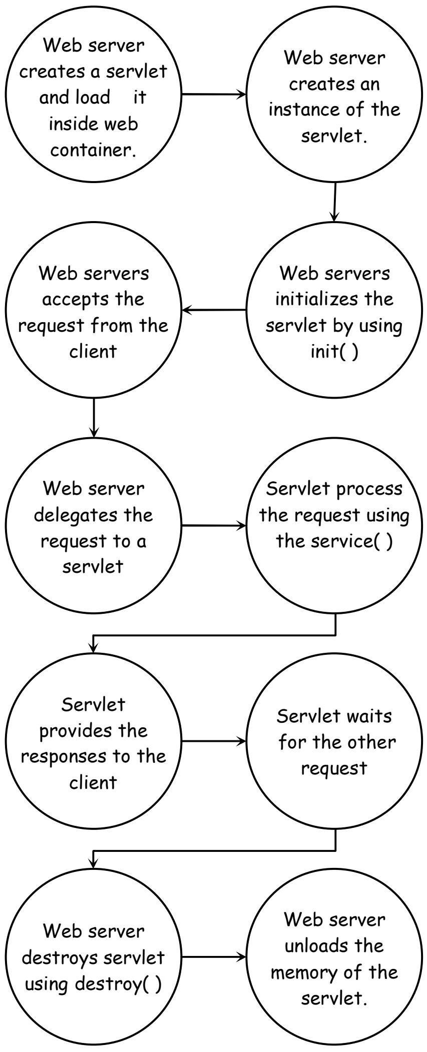
- In this when a servlet gets its first request, web server initializes that servlet using init( ).

4. Service :

- Once the servlet gets initialized it accepts the request and provides the response to client by using service( )
- The same servlet will wait for some other request and generate responses for them.

5. Destruction :

- In this if a servlet does not get request, a web server will destroy that servlet using destroy( ), to unload its memory so that the same memory can be used for some other purpose.


Web server creates a servlet and load it inside web container.
Web server creates an instance of the servlet.
Web servers initializes the servlet by using init( )
Web servers accepts the request from the client
Web server delegates the request to a servlet
Servlet process the request using the service( )
Servlet provides the responses to the client
Servlet waits for the other request
Web server destroys servlet using destroy( )
Web server unloads the memory of the servlet.

Q.1(d) Develop simple servlet question answer application. **[5]**

Ans.:
```
//index.jsp
<html>
    <head>
    <title>JSP Page</title>
    </head>
<body>
<form method="post" action="Question">
<h4><u><center>Check Your Knowledge ........</center></u></h4>
<h4>What JSP stand for </h4>
<input type="radio" name="q1" value="Java Server Pages"><b>Java Server
Pages</b><br>
<input type="radio" name="q1" value="Java Server Programming"><b>Java Server
Programming</b><br>
```

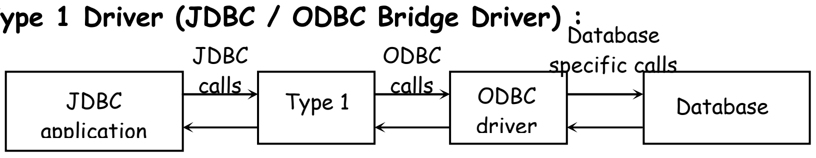
```
<input type="radio" name="q1" value="Java Service Pages"><b>Java Service
Pages</b><br>
<input type="radio" name="q1" value="Java Service Programming"><b>Java Service
Programming</b><br>
<h4>Which of the following languages can be used to write server side scripting in
ASP.NET?</h4>
<input type="radio" name="q2" value="C#"><b>C#</b><br>
<input type="radio" name="q2" value="VB"><b>VB</b><br>
<input type="radio" name="q2" value="C++"><b>C++</b><br>
<input type="radio" name="q2" value="Both a and b"><b>Both a and b</b><br>
<h4>What command is used to remove files in Linux?</h4>
<input type="radio" name="q3" value="dm"><b>dm</b><br>
<input type="radio" name="q3" value="rm"><b>rm</b><br>
<input type="radio" name="q3" value="delete"><b>delete</b><br>
<input type="radio" name="q3" value="None of the above"><b>None of the above</b><br>
<input type="Submit" name="btnsubmit">
</form></body></html>

//Question.java (Servlet)
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Question extends HttpServlet
{
public void doPost(HttpServletRequest request, HttpServletResponse response)throws
ServletException, IOException
{
response.setContentType("text/html;charset=UTF-8");
int correct=0;
int incorrect=0;
String a=request.getParameter("q1");
String b=request.getParameter("q2");
String c=request.getParameter("q3");
if(a.equals("Java Server Pages"))
{
correct++;
}
else
{
incorrect++;
}

if(b.equals("Both a and b"))
{
correct++;
}
else
{
incorrect++;
}

if(c.equals("rm"))
{
correct++;
}
else
{
incorrect++;
}
PrintWriter out = response.getWriter();
try
{
out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet Question</title>");
out.println("</head>");
out.println("<body>");
out.println("<h2>Result Of the Test</h1><br>");
out.println("<h3>Correct Answer :::"+correct+"</h3>");
out.println("<h3>Incorrect Answer :::"+incorrect+"</h3>");
out.println("</body>");
out.println("</html>");
}
catch(NumberFormatException e)
{
e.printStackTrace();
}
}
}
```

Q.1(e) Explain different types of JDBC drives. **[5]**

Ans.: **JDBC Drivers :**

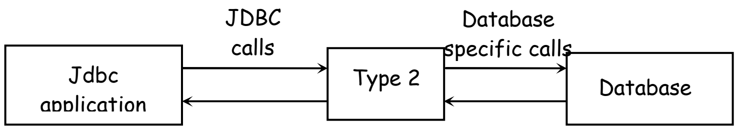
- JDBC Drivers are mainly used to convert Jdbc calls into database specific calls and vice versa.
- Following are 4 types of JDBC drivers :

1. Type 1 Driver (JDBC / ODBC Bridge Driver) :



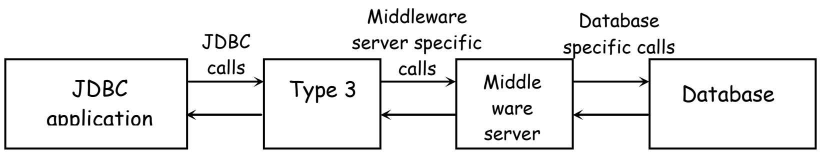
- In this type 1 driver will convert JDBC calls in ODBC calls and vice versa.
- **Advantage :** It is free and easy to implement.
- **Disadvantage :** It is slower as compared to other types of drivers.

2. Type 2 Driver (Native APT Driver) :



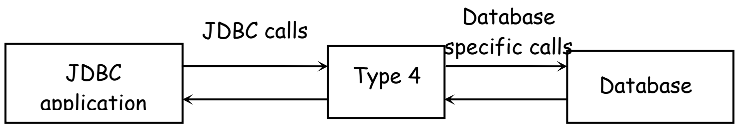
- Here Type 2 driver will convert JDBC calls into database specific call, but to send those calls it uses native API.
- **Advantage :** It is faster as compared to type 1 and type 3 driver.
- **Disadvantage :** Just to send the calls it requires Native API.

3. Type 3 Driver (Network Protocol Driver) :



- Here type 3 driver will convert JDBC calls into middleware server specific calls and vice versa.
- Middleware server is normally used to implement different security and traffic control mechanism.
- **Advantage :** It is comparatively secure among all other type of drivers.
- **Disadvantage :** It is slower as compared to type 2 and type 4 driver.

4. Type 4 Driver (Database Protocol Driver) :



- Type 4 driver will directly convert JDBC calls into database specific calls and vice versa, as well as it has the capability to send those calls.
- **Advantage :** It is the fastest of all drivers.
- **Disadvantage :** It is costliest of all drivers.

Q.1(f) Write a program to accept details of a person & using servlet store those details in database. **[5]**

Ans.:

```
//index.jsp
<html>
<body>
    <form method="post" action="test">
    Name
    <input type="text" name="a" value="" size="10">
    <br>
    Password
    <input type="password" name="b" value="" size="10">
    <br>
    Address
    <textarea name="c" rows="4" cols="10"></textarea>
    <br>
    <input type="submit" value="Register">
    </form>
    </body>
    </html>

    //test.java
    import java.io.*;
    import javax.servlet.*;
    import javax.servlet.http.*;
    public class test extends HttpServlet
    {
        public void doPost(HttpServletRequest req, HttpServletResponse res)throws
    ServletException, IOException
    {
    res.setContentType("text/html");
    PrintWriter pw = res.getWriter();
        String s1=req.getParameter("a");
         String s2=req.getParameter("b");
        String s3=req.getParameter("c");
        Class.forName("com.mysql.jdbc.Driver");
    Connection c=DriverManager.getConnection("jdbc:mysqp://localhost/sss",
    "root","root");
    PreparedStatement ps=c.prepareStatement("insert into employee values
    (?,?,?)");
    ps.setString(1,s1);
    ps.setString(2,s2);
    ps.setString(3,s3);
    ps.execute();
    ps.close();
    c.close();
    pw.println("Record Inserted");
    pw.close();
    }
}
```

Q.2Attempt the following (any THREE) **[15]**

Q.2(a) Index.html file with two text boxes to enter username and password and a login button. Login.java servlet class to process the response. If the password is "servlet" it should forward the request to Welcomeservlet.java which displays "Welcome <username>". If the password is not "servlet", login.java should display "sorry username or password error!!!". **[5]**

Ans.:

```
//index.jsp
<html>
<body>
    <form method="post" action="login">
     <b>Username</b>
    <input type="text" name="u" value="" size="10">
     <br>
      <b>Password</b>
    <input type="password" name="p" value="" size="10">
    <br>
    <input type="submit" value="Login">
    </form>
</body>
</html>

//login.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class login extends HttpServlet
{
public void doPost(HttpServletRequest req, HttpServletResponse res)throws
ServletException, IOException
{
    res.setContentType("text/html");
    PrintWriter pw = res.getWriter();
    String s=req.getParameter("p");
     if(s.equals("servlet"))
     {
         RequestDispatcher rd=req.getRequestDispatcher("welcomeservlet");
        rd.forward(req, res);
    }
    else
    {
        pw.println("Sorry username or password error !!!");
    }
pw.close();
}
}

//welcomeservlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class welcomeservlet extends HttpServlet
{
public void doPost(HttpServletRequest req, HttpServletResponse res)throws
ServletException, IOException
    {
    res.setContentType("text/html");
    PrintWriter pw = res.getWriter();
    String s=req.getParameter("u");
    pw.println("Welcome "+s);
    pw.close();
    }
}
```

Q.2(b) Explain different constructors & methods of cookie class. **[5]**

Ans.:
- javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.
- Constructor of Cookie class :

	Constructor	**Description**
1.	Cookie()	constructs a cookie.
2.	Cookie(String name, String value)	constructs a cookie with a specified name and value.

- Methods of Cookie class :

	Method	**Description**
1.	public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
2.	public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
3.	public String getValue()	Returns the value of the cookie.
4.	public void setName(String name)	changes the name of the cookie.
5.	public void setValue(String value)	changes the value of the cookie.

- Other methods required for using Cookies :
  For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are :
  1. **public void addCookie(Cookie ck) :** method of HttpServletResponse interface is used to add cookie in response object.
  2. **public Cookie[ ] getCookies() :** method of HttpServletRequest interface is used to return all the cookies from the browser.
- **Example :**
  Cookie ck=new Cookie("user","sss vidyalankar");//creating cookie object
  response.addCookie(ck);//adding cookie in the response

Q.2(c) Create a servlet that uses cookies to store number of times an user has visited servlet. **[5]**

Ans.:

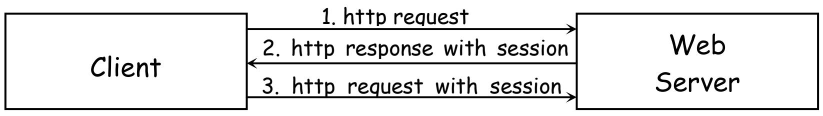
```
//index.jsp
<html>
<head>
<title>JSP Page</title>
    </head>
<body>
    <form method="post" action="VisitServlet">
<h1>Hello World!</h1>
<input type="submit" value="Count">
    </form>
    </body>
</html>

//VisitServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class VisitServlet extends HttpServlet
{
    static int i=1;
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String k=String.valueOf(i);
    Cookie c = new Cookie("visit",k);
    response.addCookie(c);
    int j=Integer.parseInt(c.getValue());
    if(j==1)
    {
    out.println("Welcome");
    }
    else
    {
    out.println("You visited "+i+" times");
    }
    i++;
    }
}
```

Q.2(d) Explain lifecycle of Http Session. **[5]**

Ans.:
- HTTP protocol is a stateless protocol and hence using HTTP it is not possible to store transaction information between client and servlet.
- To store such transaction information, in servlet, we are using the concept of session management.

Working :

1) When a client first time sends a request to a servlet, client sends only the request packet.
2) Servlet processes the request, creates a session memory generate session ID and send that session ID along with a response.
3) Next time if a same client wants to send a request to the same servlet, it sends the request packet along with session ID.
4) Using that session ID, servlet recognizes a particular session memory update that session memory, generate new ID and sends it along with the response.



Q.2(e) Write a program to create to servlet application to upload a file. **[5]**

Ans.:

```
//index.jsp
<html>
    <head>
        <title>File Upload Page</title>
    </head>
    <body>
        <h1>File Upload Application</h1>
    <formaction="FileUploadServlet"enctype="multipart/form–data" method="POST">
        File: <input type="file" name="file" id="file"/><br/><br/>
        Destination: <input type="text" value="" name="destination" /><br/><br/>
        <input type="submit" value="Upload File" name="upload" id="upload"/>
    </form>
    </body>
</html>

//FileUploadServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
@WebServlet(urlPatterns = {"/FileUploadServlet"})
@MultipartConfig
public class FileUploadServlet extends HttpServlet {
    @Override
protected void service(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    final String path = request.getParameter("destination");
    final Part filePart = request.getPart("file");
    final String fileName = retrieveFileName(filePart);
    OutputStream out = null;
    InputStream filecontent = null;
    final PrintWriter writer = response.getWriter();
try {
    out = new FileOutputStream(new File(path + File.separator + fileName));
    filecontent = filePart.getInputStream();
    int read = 0;
    final byte[] bytes = new byte[1024];

    while ((read = filecontent.read(bytes)) != -1)
    {
        out.write(bytes, 0, read);
    }
    response.sendRedirect("SucessfulUploadServlet");
    } catch (FileNotFoundException fne) {
        writer.println("You either did not specify a file to upload or are trying to upload a
file to a protected or nonexistent location.");
        writer.println("<br/> ERROR: " + fne.getMessage());
    } finally {
    if (out != null)
    out.close();
    if (filecontent != null)
    filecontent.close();
    if (writer != null)
        writer.close();
        }
    }
    private String retrieveFileName(final Part part) {
      for (String content : part.getHeader("content-disposition").split(";")) {
        if (content.trim().startsWith("filename")) {
          return content.substring(content.indexOf('=') + 1).trim().replace("\"", "");
        }
    }
    return null;
    }
}
// SucessfulUploadServlet.java
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SucessfulUploadServlet extends HttpServlet
{
@Override
protected void service(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
{
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Sucessful File Upload Page</title>");
        out.println("</head>");
```

```
            out.println("<body>");
            out.println("<h1>File Uploaded Successfully</h1>");
            out.println("<a href='index.jsp'>Click here to upload more files</a>");
            out.println("</body>");
            out.println("</html>");
        }
    }
```

Q.2(f) Write a program to create a servlet application to download a file. [5]

Ans.:

```
//index.jsp
<html>
  <head>
    <title>File Download Page</title>
  </head>
  <body>
    <h1>File Download Application</h1>
    Click <a href="FileDownloadServlet?filename=Chapter2.pdf">Chapter2</a>
    <br/><br/>
    Click <a href="FileDownloadServlet?filename=EJava.pdf">Enterprise Java</a>
  </body>
</html>
// FileDownloadServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
public class FileDownloadServlet extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
  String fileToDownload = request.getParameter("filename");
  System.err.println("Downloading file now...");
  downloadFile(request, response, fileToDownload);
}
private void downloadFile(HttpServletRequest request, HttpServletResponse response,
String fileName) throws ServletException, IOException
 {
    int lenght = 0;
    ServletOutputStream outputStream = response.getOutputStream();
      ServletContext context = getServletConfig().getServletContext();
      response.setContentType((context.getMimeType(fileName) != null) ?
      context.getMimeType(fileName) : "application/pdf");
      response.setHeader("Content-Disposition", "attachment; filename=\"" + fileName);
      InputStream inputStream = context.getResourceAsStream("/" + fileName);
      byte[ ] bytes = new byte[1024];

      while((inputStream != null) && ((length = inputStream.read(bytes)) != -1)) {
        outputStream.write(bytes, 0, length);
      }
      outputStream.flush();
    }
}
```

Q.3Attempt the following (any THREE) **[15]**

Q.3(a) Distinguish between JSP and Servlets. **[5]**

Ans.:

	Servlet	**JSP**
1.	Servlet is plat form independent.	JSP is platform dependent but gets converted to their respective servlet.
2.	Servlet is faster in application development.	JSP is slower in application development.
3.	Servlet is difficult for web designers to use.	JSP is easy for web designers to use.
4.	Servlet uses web.xml configuration file.	JSP does not use web.xml configuration life.
5.	Servlet are purely written in Java.	JSP are purely written in scripting language.
6.	They do not use any tag.	They use html and JSP tags

Q.3(b) Explain different types of JSP tags with example. **[5]**

Ans.: ***JSP Tags/Element :***

Following are different JSP tags which are used to embed Java code inside JSP :

1) Directive Tag : It has following 3 types:

(i) Page Directive Tag : If we want to perform certain operations on the entire JSP page then those operation can be performed using page direction tag.

Example :

```
<%@ page import ="java.io.*"%>
<%@ page language = "java"%>
<%@ page session = "true"%>
<%@ page content type = "text/html"%>
```

(ii) Include Directive Tag : It is used to include the contents of one JSP into the current JSP.

Example :

```
<%@ include file = "abc.jsp"%>
```

(iii) Taglib Directive Tag : If we want to use some special tags other than html and jsp inside jsp program, then those tags can be used with the help of taglib directive tag.

Example :

```
<% @ taglib uri = " www.w3.org"%>
```

2) Declaration Tag : If we want to declare certain variables inside JSP page then it can be declared using declaration tag.

Example :

```
<%!int a = 0 %>
```

3) Expression Tag : If we want to display a value of the variable or an expression on a browser, then it can be displayed using expression tag.

Example :

```
< % = a % >
```

4) Scriptlets Tag : If we want to write a java code inside jsp page, so that it will get exeulted like a java then it can be written using scriptlets tag.

Example :

```
< %
________
________  } Java code
________
________
% >
```

5) **Comment Tag :** If we want to display a comment inside JSP then it can get displayed using comment tag.
   Example :
   ```
   < % – – comment – –% >
   ```

Q.3(c) Explain different JSP action elements with example. [5]

Ans.: Following are JSP action elements which are used to perform different operations on JSP page :

1) include :
- It is used to include the contents of one JSP into current JSP.
- When we include the contents, the output of both JSP's will get displayed.
  Example :
  ```
  <jsp: include page = "abc.jsp"/>
  ```

2) forward :
- It used to forward the control from current JSP to some other JSP.
- When we forward the control, the output of only forwarded JSP will get displayed.
  Example :
  ```
  <jsp : forward page = "xyz.jsp"/>
  ```

3) useBean :
- It is mainly used to use a particular Bean (a Java class) inside the JSP application.
- By using a Bean we can use or access property of that Bean without the object.
  Example :
  ```
  <jsp : useBean id = "abc"     class = "A">
  ...
  ...
  </jsp : useBean>
  ```

  It has two parameters:
  (i) id :
  - It represents name of Bean

  (ii) class
  - It represents name of Bean class.
  - We can access Bean property, i.e. we can set or give a value to Bean property.

  For that we use "set property" tag. it has 3 parameters:
  (i) name :
  - It represents name of Bean.

  (ii) Property :
  - It represents name of the property.

  (iii) Value :
  - It represents a value given to that property.
  - We can also retrieve a value of the Bean property using "getProperty" tag.

  It has 2 parameters :
  (a) name : It represents name of Bean.
  (b) property : It represents name of the property.

Example :
```
<jsp:useBean    id = "abc"   class = "A">
<jsp: setProperty name = "abc"
          property = "name" value = "sss"/>
<jsp: getProperty name = "abc"
           property = "name"/>
</jsp: useBean>
```

Q.3(d) Create a registration and login.jsp application to register and authenticate the user based on username and password using JDBC. **[5]**

Ans.:
```
//index.jsp
<html>
<body>
    <form method="post" action="login.jsp">
        <b>Username</b>
        <input type="text" name="u" value="" size="10">
        <br>
        <b>Password</b>
        <input type="password" name="p" value="" size="10">
        <br>
        <input type="submit" value="Login">
    </form>
</body>
</html>

//login.jsp
<html>
<body>
    <%@page import="java.sql.*" %>
    <%
        String username = request.getParameter("u");
        String password = request.getParameter("p");
        out.println(username);
        out.println(password);
        Class.forName("com.mysql.jdbc.Driver");
        Connection c=DriverManager.getConnection
("jdbc:mysql://localhost/jspdb","root","root");
        PreparedStatement statement = c.prepareStatement("select firstname, password
from registration where firstname =? and password=?");
        statement.setString(1, username);
        statement.setString(2, password);
        ResultSet result = statement.executeQuery();
        if(result.next())
        {
            out.println("Login Successful");
        }
        else
        {
            out.println("username and password are incorrect");
        }
    %>
</body>
</html>
```

Q.3(e) Write a short note on expression language in JSP. [5]

Ans.: **Expression Language in JSP :**

JSP Expression Language (EL) makes it possible to access application data stored in JavaBeans components.

JSP EL allows us to create expressions both (a) arithmetic and (b) logical.
Within a JSP EL expression, we can use integers, floating numbers, strings, boolean values, and null.

Simple Syntax :
When we specify an attribute value in a JSP tag, we simply use a string.

For example :
```
    <jsp:setProperty name = "rectangle" property = "perimeter" value = "100"/>
```
JSP EL allows us to specify an expression for any of these attribute values.

A simple syntax for JSP EL is as follows :
```
    ${expression}
```

Here **expression** specifies the expression itself. The most common operators in JSP EL are **.** and **[]**. These two operators allows us to access various attributes of Java Beans and built-in JSP objects.

For example, the above syntax <jsp:setProperty> tag can be written with an expression like :
```
    <jsp:setProperty name = "rectangle" property = "perimeter"
    value = "${2*rectangle.width+2*rectangle.height}"/>
```
When the JSP compiler sees the **${}** substitutes the value of expression.

We can also use the JSP EL expressions within template text for a tag.
For example, the **<jsp:text>** tag simply inserts its content within the body of a JSP. The following **<jsp:text>** declaration inserts **<h1>Hello!</h1>**into the JSP output :
```
<jsp:text>
    <h1>Hello!</h1>
</jsp:text>
```

We can now include a JSP EL expression in the body of a **<jsp:text>** tag (or any other tag) with the same **${}** syntax you use for attributes. For example :
```
<jsp:text>
    Rectangle Perimeter is: ${2*rectange.width + 2*rectangle.height}
</jsp:text>
```

EL expressions can use parentheses to group subexpressions.
For example, ${(1 + 2) * 3} equals 9, but ${1 + (2 * 3)} equals 7.

To deactivate the evaluation of EL expressions, we specify the **isELIgnored** attribute of the page directive as below :
```
    <%@ page isELIgnored = "true|false" %>
```

The valid values of this attribute are true and false. If it is true, EL expressions are ignored and if it is false, EL expressions are evaluated by the container.

Q.3(f) Create a JSP application to demonstrate the use of JSTL. [5]

Ans.:

```
//index.jsp
<html>
<head>
    <title>If with Body</title>
</head>

<body>
    <c:if test="${pageContext.request.method=='POST'}">
    <c:if test="${param.guess=='Java'}">You guessed it!
    <br />

    <br />

    <br />
    </c:if>

    <c:if test="${param.guess!='Java'}">You are right
    <br />

    <br />

    <br />
    </c:if>
    </c:if>

    <form method="post">Guess what computer language
    I am thinking of?

    <input type="text" name="guess" />

    <input type="submit" value="Try!" />

    <br />
    </form>
</body>
</html>
```

Q.4 Attempt the following (any THREE) [15]

Q.4 (a) Explain different types of EJB. [5]

Ans.: EJBs can be categorized into the following two types :

(i) Session beans : Those are the beans which will get used when the client sends the request and for handling that request that particular EJB is required.

Characteristics :

1) It is a single (short) lived bean.
2) They are transaction oriented.
3) They cannot be create using data from the database but has the ability to change the database.
4) They are synchronous in nature i.e. client and server needs to be connect while accessing them.
5) They can be stateful or stateless.
6) They can be used with the help of home interface.

Types of session beans :

1) Stateful session beans : Those are the session beans which store the information about a client which uses them. They are slower as compared to stateless session beans.
2) Stateless session beans : Those are the session beans which does not store the information about a particular client. They are faster as compared to stateful session beans.
3) Singleton session beans : Those are the session beans which can be used by multiple client simultaneously.

(ii) Message driven beans : Those are the beans which will get used when a particular event will get generated for those beans.

Characteristics :

1) It is single (short) lived beans.
2) They are transaction oriented.
3) They cannot be created using data from the database, but has the ability to change the database.
4) They are a synchronous in nature.
5) They are stateless.
6) They cannot be used with the help of home interface.

Q.4 (b) Create a currency converter application using EJB. **[5]**

Ans.:

```
//index.jsp
<html>
<head>
    <title>Currency Converter</title>
</head>
<body>
    <h1>Currency Converter</h1>
    <hr>
    <p>Enter an amount to convert:</p>
    <form method="post" action="ConvertCurrencyServlet">
    <b> Convert: </b>
    <input type="text" name="amount" value="1" size="10" tabindex="1" />
    <br>
    <b>Enter an amount</b>
    <br>
    <b> From this currency:</b>
    <select name="From" size="3" tabindex="2">
    <option value="USD">America (United States), Dollar (USD)</option>
    <option value="INR">India, Rupee (INR)</option>
    </select>
    <br><br>
    <b> To this currency:</b>
    <select name="To" size="3" tabindex="3">
    <option value="USD">America (United States), Dollar (USD)</option>
    <option value="INR">India, Rupee (INR)</option>
    </select>
    <br><br>
    <input name="cmdSubmit" type="submit" value="Submit" tabindex="4" />
    </form>
</body>
</html>
```

```
// ConvertCurrencyServlet.java (Servlet)
import ejb.CurrencyConverterBean;
import java.io.*;
import java.io.PrintWriter;
import java.math.*;
import javax.ejb.*;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
public class ConvertCurrencyServlet extends HttpServlet
{
    @EJB
    CurrencyConverterBean converterBean;

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String amount = request.getParameter("amount");
        if (amount != null && amount.length() > 0)
        {
            BigDecimal d = new BigDecimal(amount);
            BigDecimal          convertedAmount          =          converterBean.convert
    (request.getParameter("From"), request.getParameter("To"), d);
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Converted Currency</title>");
            out.println("</head>");
            out.println("<body>");
            out.println(amount + " " + request.getParameter("From") + " = ");
            out.println(convertedAmount + " " + request.getParameter("To"));
            out.println("</body>");
            out.println("</html>");
        }
    }
}

// CurrencyConverterBean.java (Stateless Session Bean)
package ejb;
import java.math.*;
import javax.ejb.*;
@Stateless
public class CurrencyConverterBean
{
    final private BigDecimal USD = new BigDecimal("0.0229137");
    final private BigDecimal INR = new BigDecimal("46.589100");
    public BigDecimal convert(String from, String to, BigDecimal amount)
    {
        if(from.equals(to))
        {
            return amount;
        }
```

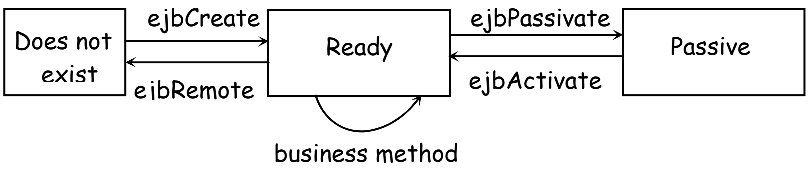
```
            else
            {
                BigDecimal toRate = findRate(to);
                BigDecimal result = toRate.multiply(amount);
                return result.setScale(2, BigDecimal.ROUND_UP);
            }
        }

        public BigDecimal findRate(String to)
        {
            BigDecimal returnValue = null;
            if(to.equals("INR"))
            {
                returnValue = INR;
            }
            if(to.equals("USD"))
            {
                returnValue = USD;
            }
            return returnValue;
        }
    }
```

Q.4 (c) Explain lifecycle of stateful session beans. [5]

Ans.: The following figure shows the life cycle of a stateful session bean. It has the following states :

- **Does not exist.** In this state, the bean instance simply does not exist.
- **Ready state.** A bean instance in the ready state is tied to particular client and engaged in a communication.
- **Passive state.** A bean instance in the passive state is passivated to conserve resource.

The various state transitions as well as the methods available during the various states are discussed below :



Moving from the Does Not Exist to the Ready State :

- When a client invokes a create method on a stateful session bean, the EJB container creates a new instance and invokes the callback method public void setSessionContext(SessionContext ctx).
- After the callback method setSessionContext is called, the EJB container calls the callback method ejbCreate that matches the signature of the create method.

The Ready State :

- A stateful bean instance in the ready state is tied to a particular client for the duration of their communication. During this communication the instance can the execute component methods invoked by the client.

Activation and Passivation :

- To more optimally manage resources, the EJB container might passivate an inactive stateful session bean instance by moving it from the ready state to the passive state.
- If after passivation a client application continues the conversation by invoking a business method, the passivated bean instance is reactivated.
- Right after the state has been restored, the callback method ejbActivate is invoked. If your session bean needs to execute some custom logic after activation, you can implement it using this callback method. The caller (a client application) of the session bean instance will be unaware of passivation (and reactivation) having taken place.

Moving from the Ready to the Does Not Exist State :

- When a client application invokes a remove method on the stateful session bean, it terminates the conversation and tells the EJB container to remove the instance.
- Just prior to deleting the instance, the EJB container will call the callback method ejbRemove. If your session bean needs to execute some custom logic prior to deletion, you can implement it using this callback method.

Q.4 (d) Develop simple EJB application to demonstrate servlet hit count using singleton session beans. **[5]**

Ans.:

```
//index.jsp
<html>
   <body>
      <form method="post" action="ServletClient">
         <input type="submit" value="Count">
      </form>
   </body>
</html>

//ServletClient.java (Servlet)
package servlet;

import ejb.*;
import java.io.*;
import javax.ejb.*;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;

public class ServletClient extends HttpServlet
{
   @EJB
   CountServletHitsBean counterBean;

   @Override
   public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
   {
      response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         out.println("<!DOCTYPE html>");
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Servlet ServletClient</title>");
         out.println("</head>");
         out.println("<body>");
```

```
        out.println("<h1>Number of times this servlet is accessed: " +
counterBean.incrementAndGetHitCount() + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}

// CountServletHitsBean.java (Singleton Session Bean)
package ejb;
import javax.ejb.Singleton;
@Singleton
public class CountServletHitsBean
{
    private int hitCount;
    public synchronized int incrementAndGetHitCount()
   {
        return hitCount++;
    }
}
```

Q.4 (e) What different things an interceptor can do with request, also explain defining and interceptor. [5]

Ans.: Every request with passes through each interceptor, An interceptor can,

- Ignore the request
- Process the request data
- Short circuit the request and present the EJB class method from firing

Interceptors have to access the EJB class which is being executed as well as all environmental variables & execution properties.

Defining an interceptors :

- Interceptors methods can be used to intercept either a business method or lifecycle plant.
- An intercaptors that intercepts a business method is typically called an AroundInvoke Method because it can be defined by annotating the method an @AroundInvoke annotation.
- An AroundInvoke method can be defined as :
  - Enterprise bean class
  - Interceptor class
- An Interceptor class is a normal class which does not extend any class or implement any interface.

Q.4 (f) What is JNDI? [5]

Ans.:
- The Java naming & directory interface (JNDI) is an application programming interface (API) that provides naming & directory functionality to applications written using Java programming language.
- It is a Java API for a directory service that allows Java Software clients to discover and look up data and objects via a name.
- JNDI API is used by the Java RMI and Java EE APIs to look up objects in a network.
- The API provides :
  - A mechanism to bind an object to a name.
  - A directory look up interface that allows general queries.
  - An event/interface that allows clients to determine when directory entries have been modified.
  - LDAP extensions to support the additional capabilities of an LDAP service.

Q.5 Attempt the following (any THREE) [15]

Q.5 (a) Explain persistence in java. [5]

Ans.:
- Any enterprise application performs database operations by storing and retrieving vast amounts of data.
- Despite all the available technologies for storage management, application developers normally struggle to perform database operations efficiently.
- Generally, developers use lots of code, or use the proprietary framework to interact with the database, whereas using JPA, the burden of interacting with the database reduces significantly.
- It forms a bridge between object models (Java program) and relational models (database program).

Persistence of Object Oriented Models :
- Relational objects are represented in a tabular format, while object models are represented in an interconnected graph of object format.
- While storing and retrieving an object model from a relational database, some mismatch occurs due to the following reasons:
  - **Granularity :** Object model has more granularity than relational model.
  - **Subtypes :** Subtypes (means inheritance) are not supported by all types of relational databases.
  - **Identity :** Like object model, relational model does not expose identity while writing equality.
  - **Associations :** Relational models cannot determine multiple relationships while looking into an object domain model.
  - **Data navigation :** Data navigation between objects in an object network is different in both models.

Q.5 (b) Why ORM is used? Explain ORM. [5]

Ans.: When we work with an object-oriented system, there is a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java or C# represent it as an interconnected graph of objects.

Consider the following Java Class with proper constructors and associated public function :

```
public class Employee
{
  private int id;
  private String first_name;
  private String last_name;
  private int salary;

  public Employee() {}
  public Employee(String fname, String lname, int salary)
 {
    this.first_name = fname;
    this.last_name = lname;
    this.salary = salary;
  }

  public int getId()
  {
    return id;
  }

  public String getFirstName()
 {
    return first_name;
  }
  public String getLastName()
  {
    return last_name;
  }
  public int getSalary()
  {
    return salary;
  }
}
```

Consider the above objects are to be stored and retrieved into the following RDBMS table :

```
create table EMPLOYEE (
  id INT NOT NULL auto_increment,
  first_name VARCHAR(20) default NULL,
  last_name  VARCHAR(20) default NULL,
  salary     INT  default NULL,
  PRIMARY KEY (id));
```

First problem, what if we need to modify the design of our database after having developed a few pages or our application?

Second, loading and storing objects in a relational database exposes us to the following five mismatch problems :

	Mismatch	**Description**
1	Granularity	Sometimes you will have an object model, which has more classes than the number of corresponding tables in the database.
2	Inheritance	RDBMSs do not define anything similar to Inheritance, which is a natural paradigm in object-oriented programming languages.
3	Identity	An RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).
4	Associations	Object-oriented languages represent associations using object references whereas an RDBMS represents an association as a foreign key column.
5	Navigation	The ways you access objects in Java and in RDBMS are fundamentally different.

The Object-Relational Mapping (ORM) is the solution to handle all the above impedance mismatches.

What is ORM?

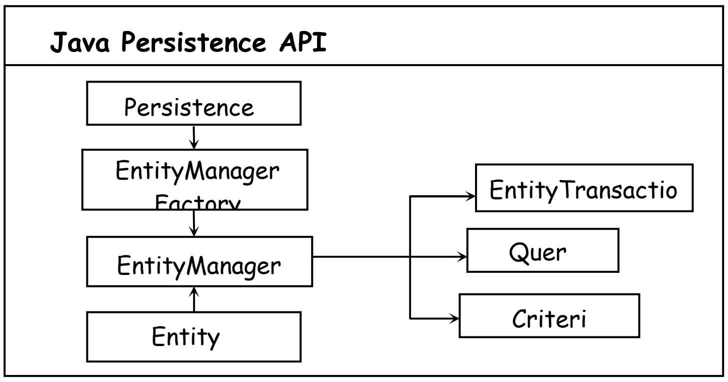
ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C#, etc.

An ORM system has the following advantages over plain JDBC :

1. Let's business code access objects rather than DB tables.
2. Hides details of SQL queries from OO logic.
3. Based on JDBC 'under the hood.'
4. No need to deal with the database implementation.
5. Entities based on business concepts rather than database structure.
6. Transaction management and automatic key generation.
7. Fast development of application.

Q.5 (c) Explain how JPA works. [5]

Ans.: The following diagram shows the class level architecture of JPA. It shows the core classes and interfaces of JPA.

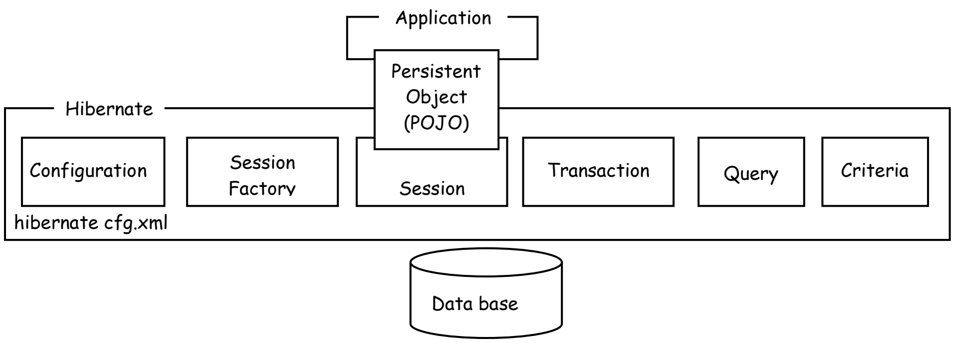


The following table describes each of the units shown in the above architecture :

Class/ Interface	Description
EntityManagerFactory	This is a factory class of EntityManager. It creates and manages multiple EntityManager instances.
EntityManager	It is an Interface, it manages the persistence operations on objects. It works like factory for Query instance.
Entity	Entities are the persistence objects, stores as records in the database.
EntityTransaction	It has one-to-one relationship with EntityManager. For each EntityManager, operations are maintained by EntityTransaction class.
Persistence	This class contain static methods to obtain EntityManagerFactory instance.
Query	It is mainly used to retrieve data from database using SQL queries
Criteria	It is mainly used to retrieve data from database using different methods

Q.5 (d) Explain architecture of hibernate. [5]

Ans.:



1. **Configuration** : This component contains a file 'hibernate.cfg.xml." which is used to establish a connection with the database. It is also used to create session factory component.
2. **Session factory** : This component will get created only once but it is responsible to create different session object.
3. **Session** : It is used to receive a persistent object inside hibernate layer and used to save that object inside the database. For different persistent object, separate session objects will get created. It is also used to create transaction object.
4. **Transaction** : It represent unit of work which means it represent when the operations will get start and when they will get stopped.

5. **Query** : To perform different operation if we want to use different sql queries, then transaction will create query object.
6. **Criteria** : To perform different operation if we want to use java methods, then transaction will create criteria object.

Q.5 (e) Explain structure of guestbook.hbm.xml file (Hibernate mapping file). **[5]**

Ans.:

```
<?xml version="1.0" encoding="UTF–8"?>
<hibernate–mapping>
   <class name="guestbook" table="guestbooktable">
      <property name="name" type="string">
         <column name="username" length="50" />
      </property>
      <property name="message" type="string">
         <column name="usermessage" length="100" />
      </property>
   </class>
</hibernate–mapping>
```

Elements:

<hibernate–mapping>…........</hibernate–mapping>

It is the base tag which is used to write hibernate mapping file, which is used to map POJO class with database table.

<class>…........</class>

It represents name of the class and database table which we want to map with each other. It has 2 parameters:

name– It represents name of the class

table– It represents name of the database table

<property>…........</property>

It is used to write the property which we want to map with database column. It has 2 parameters:

name– It represents name of the property

type– It represents type of the property

<column>…........</column>

It is used to write the database column which we want to map with java class property. It has 2 parameters:

name– It represents name of the column

length– It represents maximum length of a column value

Q.5 (f) Explain structure of hibernate.sfg.xml file (Hibernate configuration file). **[5]**

Ans.:

```
<?xml version="1.0" encoding="UTF–8"?>

<hibernate–configuration>
<session–factory>

<property name="hibernate.dialect">
org.hibernate.dialect.MySQLDialect</property>
<property name="hibernate.connection.driver_class">
com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.url">
jdbc:mysql://localhost/DBname</property>
```

```
<property name="hibernate.connection.username">
root</property>

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

<mapping resource="guestbook.hbm.xml"/>

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

Elements:
- hibernate.dialect : It represents the name of the SQL dialect for the database.
- hibernate.connection.driver_class : It represents the JDBC driver class for the specific database.
- hibernate.connection.url : It represents the JDBC connection to the database.
- hibernate.connection.username : It represents the user name which is used to connect to the database.
- hibernate.connection.password : It represents the password which is used to connect to the database.
- guestbook.hbm.xml : It represents the name of mapping file.

❑ ❑ ❑ ❑ ❑