(2½ Hours)

[Total Marks: 75]

N. B.: (1) **All** questions are **compulsory**.
(2) Make **suitable assumptions** wherever necessary and **state the assumptions** made.
(3) Answers to the **same question** must be **written together**.
(4) Numbers to the **right** indicate **marks**.
(5) Draw **neat labeled diagrams** wherever **necessary**.
(6) Use of **Non-programmable** calculators is **allowed**.

1. Attempt *any three* of the following: 15

a. **What is an enterprise application, and what is the enterprise edition of Java?**

Enterprise application

Enterprise application software is large-scale software that is aimed to support or solve the problems of an entire organization. This large-scale software allows for several different user roles, and the roles define the actions that specific user can perform. An enterprise application is a business application, obviously. As most people use the term, it is a *big* business application. In today's corporate environment, enterprise applications are complex, scalable, distributed, component-based, and mission-critical. They may be deployed on a variety of platforms across corporate networks, intranets, or the Internet. They are data-centric, user-friendly, and must meet stringent requirements for security, administration, and maintenance. In short, they are highly complex systems.

The enterprise software includes many functions and abilities:

- Salespeople can log client information, including personal and product information and client location in the sales pipeline.
- Customer service representatives can track client issues and communication alongside internal responses and updates.
- Internal-facing employees can take client specifications to build the product.
- Managers and C-level executives can track client and employee performance.

Enterprise Edition

The Java Platform, Enterprise Edition (Java EE) is a collection of Java APIs owned by Oracle that software developers can use to write server-side applications. It was formerly known as Java 2 Platform, Enterprise Edition, or J2EE.

The J2EE is not a product, rather it is a specification defining a server side Java framework. Vendors are responsible for using this J2EE spec in order to make compliant J2EE Servers. This allows IT developers to write their Java Business Logic using the J2EE APIs. They do not have to worry about implementing the surrounding framework.

It combines a number of technologies in one architecture with a comprehensive Application Programming Model and Compatibility Test Suite for building enterprise-class server-side applications.

Java EE applications are hosted on application servers, such as IBM's WebSphere, Oracle's GlassFish or Red Hat's WildFly server, all of which run either in the cloud or within a corporate data center. While Java EE apps are hosted on the server side, examples of Java EE clients include an internet of things (IoT) device, smartphone, RESTful web service, standard web-based application, WebSocket or even microservices running in a Docker container.

Java EE provides collection of:

- Standardized components that facilitates the crafting of commercial applications
- Standard interfaces that define how the various application modules interconnect
- Standard services that define how the different software modules communicate

b. **Define Java EE containers with the various Java Container types.**

Container Types:

1 Enterprise JavaBean (EJB) Container:
EJB container is a server-side component architecture. It is mainly used for modular construction of enterprise application. An EJB container provides a run-time environment for enterprise beans within the application server. The container handles all aspects of an enterprise bean's operation within the application server. EJB container acts as an intermediary between the user-written business logic within the bean and the rest of the application server environment. It manages the execution of enterprise beans for J2EE applications. Enterprise beans and their container run on the Java EE server. The EJB container provides local and remote access to enterprise beans. Enterprise bean can be divided into session beans, entity beans, and message-driven beans depend on the data. Session beans represent transient objects and processes and typically are used by a single client. Entity beans represent persistent data, typically maintained in a database. Message-driven beans asynchronously pass messages to application modules and services.

2 Web Container:
A Web application runs within a Web container of a Web server which is also known as web servlet container. The Web container provides the runtime environment through components. Web container manages the execution of JSP page and servlet components for Java EE applications. A web container provides the same services as a JSP container. Web components and their container run on the Java EE server. Web server is a server which is capable of handling HTTP request send by a client and respond back with a HTTP response.

3 Application Client Container:
To host application components, application clients, the application client container is used. It runs on the client computer and it can be interacted with each other. It helps developers to create robust Java applications that also have access to Java EE resources such as data sources or EJBs. This container manages the execution of application client components. Application clients and their container run on the client. Application server can handle all application operations between users and databases

4 Applet Container:
Applet container includes support for the applet programming model. The applet container is a mixture of web browser and java plug in on client machine. The applet container uses the sandbox security model, preventing applets from accessing system resources and causing harm. Applet container ensures security and portability if it runs in an applet.

c. **Explain the life cycle of a servlet application.**

The life cycle of a Servlet is managed by the container in which the Servlet has been deployed. When a request is mapped to a Servlet, the container performs the following steps.

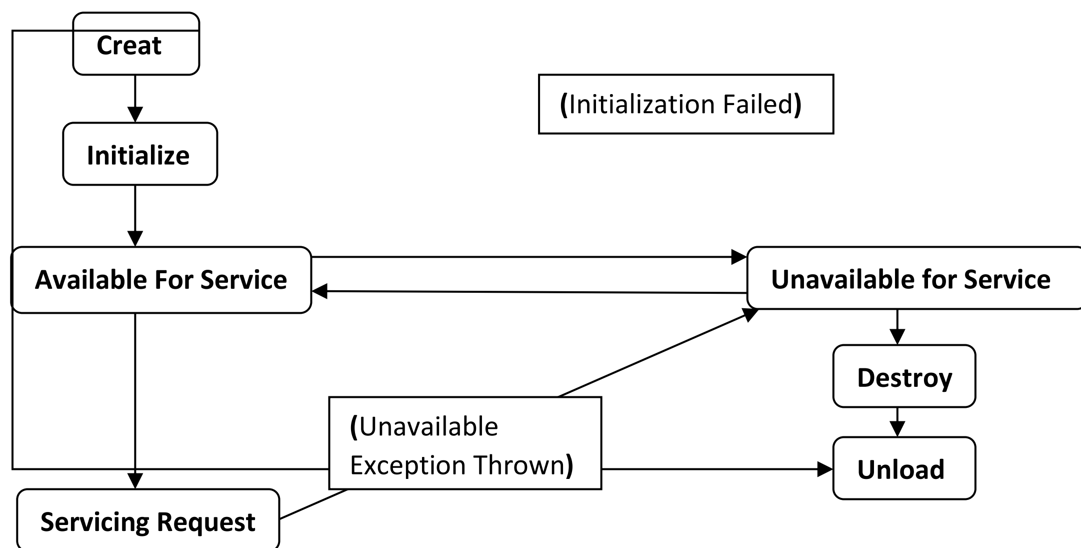
1) If an occurrence of the Servlet does not exist, the Web container …

- Loads the Servlet class.
- Creates an instance of the Servlet class.
- Initializes the Servlet instance by calling the init method. Initialization is covered in Initializing a Servlet.

2) Invokes the service method, passing a request and response object. Service methods are discussed in the section Writing Service Methods.

If the container needs to remove the Servlet, it finalizes the Servlet by calling the Servlets destroy method.

The following are the life cycle methods of a Servlet instance:



We will look into the each method in detail.

1: init( ) : This method is called once for a Servlet instance. When first time Servlet is called, Servlet container creates instance of that Servlet and loaded into the memory. Future requests will be served by the same instance without creating the new instance. init( ) method is used for inilializing Servlet variables which are required to be passed from the deployment descriptor web.xml. The *ServletConfig* is passed as the parameter to init( ) method which stores all the values configured in the web.xml.

2: service( ): This method is called for the each request. This is the entry point for the every Servlet request and here we have to write our business logic or any other processes. This method takes *HttpServletRequest* and *HttpServletresponse* as the parameters.

3: destroy( ): This method will be called once for an instance. It is used for releasing any resources used by the Servlet instance. Most of the times it could be database connections, Fill IO operations, etc. destroy( ) is called by the container when it is removing the instance from the Servlet container.

d. **Write a short note on javax.Servlet.Servlet package.**

Defines methods that all servlets must implement. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.
To implement this interface, you can write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet.
This interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server. These are known as life-cycle methods and are called in the following sequence:

1. The servlet is constructed, then initialized with the init( ) method.
2. Any calls from clients to the service method are handled.
3. The servlet is taken out of service, then destroyed with the destroy method, then garbage collected and finalized.

In addition to the life-cycle methods, this interface provides the getServletConfig, which servlet can use to get any startup information, and the getServletInfo , which allows servlet to return basic information about itself, such as author, version, and copyright.

Methods
1) public abstract void destroy( ): This method called when the server is preparing to unload a Servlet. It is used to clean up any outstanding resources such as database connections, threads, file handles etc.
2) public abstract ServletConfig getServletConfig( ): This method returns the *ServletConfig* object passed to the init( ) method.
3) public abstract String getServletInfo( ):This method returns a programmer defined String that describes the Servlet.
4) public abstract void init(ServletConfig config) throws ServletException: This method called by the server when the Servlet is first loaded. The init( ) method should store the *ServletConfig* object for retrieval by getServletConfig( ).This method can also be used to perform any one time actions required by the Servlet, such as creating database connections.
5) public abstract void service(ServletRequest req, ServletResponse res) throws ServletException, IOException : This method is called to handle a single client request. A Servlet receives request information through the *ServletRequest* object and sends data back to the client through the *ServletResponse* object.

e. **What is a JDBC Statement object? Explain its 3 types.**
Once a connection is obtained we can interact with the database. Connection interface defines methods for interacting with the database through the established connection. To execute SQL statements, we need to instantiate a Statement object from our connection object by using the createStatement( ) method. As shown below.
Statement tStmt = tDbConn.createStatement( );
A statement object is used to send and execute SQL statements to a database.

There are 3 kinds of Statements

I. Standard Statement: It executes simple SQL queries without parameters. Thisstatement creates an SQL Statement object. A query that return the data can be executed using executeQuery( ) Statement.

Syntax:

Statement tStmt= tDbConn.createStatement( );

Example:

ResultSet trs= tStmt.executeQuery("Select * from Registration");

II. Prepared Statement: It execute a precompiled SQL queries with or without parameters. IT returns a new *PreparedStatement* object. *PreparedStatement* objects are Precompiled SQL statements.

Syntax:

PreparedStatement tPStmt= tDbConn.prepareStatement("Insert in to ....");

III. Callable Statement: It executes a call to a database stored procedure. It returns a new *CallableStatement* object. *CallableStatement* objects are SQL stored procedure call statements.

Syntax: CallableStatement tCStmt=tDbcon.prepareCall("{call getEmployee(?,?)}")

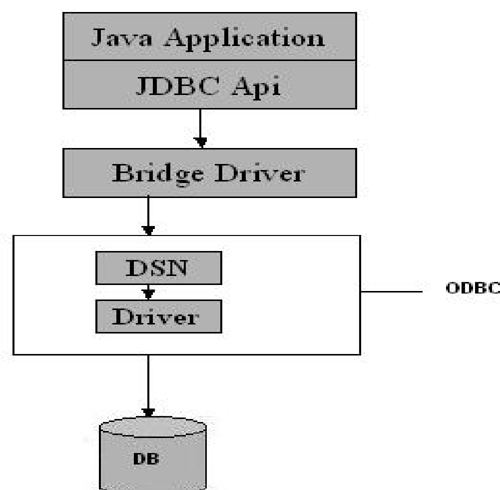
f. **List and explain each of the four JDBC driver types.**

The JDBC API defines the Java interfaces and classes that programmers use to connect to databases and send queries. A JDBC driver implements these interfaces and classes for a particular DBMS vendor. The JDBC driver converts JDBC calls into a network or database protocol or into a database library API call that makes communication with the database. This translation layer provides JDBC applications with database autonomy. In the case of any back-end database change, only we need to just replace the JDBC driver & some code modifications are required. "The Java program that uses the JDBC API loads the specified driver for a particular DBMS before it actually connects to a database. After that the JDBC DriverManager class then sends all JDBC API calls to the loaded driver".

There are four types of JDBC drivers as follows:

1) Type 1- JDBC-ODBC Bridge plus ODBC driver:

This driver converts JDBC API calls into Microsoft Open Database Connectivity (ODBC) calls that are then passed to the ODBC driver. The ODBC binary code must be loaded on every client computer that uses this type of driver.

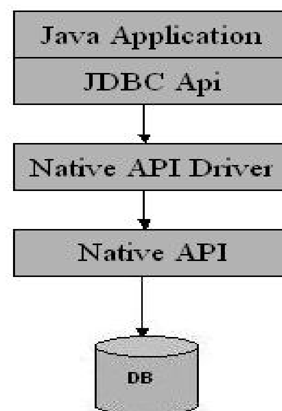


Type 1: JDBC-ODBC Bridge

Advantage: The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.
Disadvantages:

- The Bridge driver is not coded completely in Java; So Type 1 drivers are not portable.
- It is not good for the Web Application because it is not portable.
- It is comparatively slowest than the other driver types.
- The client system requires the ODBC Installation to use the driver.

2) Type 2 - Native-API/ Partly Java driver:
This driver converts JDBC API calls into DBMS precise client API calls. Similar to the bridge driver, this type of driver requires that some binary code be loaded on each client computer.



Native API/Partly Java Driver

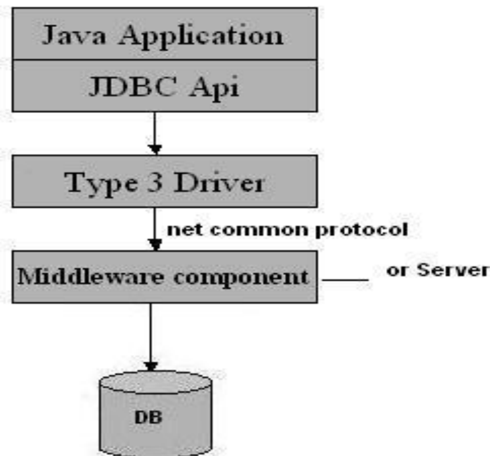
Advantage: This type of divers are normally offer better performance than the JDBC-ODBC Bridge as the layers of communication are less than that it and also it uses Resident/Native API which is Database specific.
Disadvantage:

- Mostly out of date now.
- It is usually not thread safe.
- This API must be installed in the Client System; therefore this type of drivers cannot be used for the Internet Applications.
- Like JDBC-ODBC Bridge drivers, it is not coded in Java which cause to portability issue.
- If we modify the Database then we also have to change the Native API as it is specific to a database.

3) Type 3 - JDBC-Net/Pure Java Driver:
This driver sends JDBC API calls to a middle tier Net Server that translates the calls into the DBMS precise Network protocol. The translated calls are then sent to a particular DBMS.
Advantage:

- This type of drivers is the most efficient amongst all driver types.
- This driver is totally coded in Java and hence Portable. It is suitable for the web Applications.
- This driver is server based, so there is no need for any vendor database library to be present on client machines.

- With this type of driver there are many opportunities to optimize portability, performance, and scalability.
- The Net protocol can be designed to make the client JDBC driver very small and fast to load.
- This normally provides support for features such as caching, load balancing etc.
- Provides facilities for System administration such as logging and auditing.
- This driver is very flexible allows access to multiple databases using one driver.

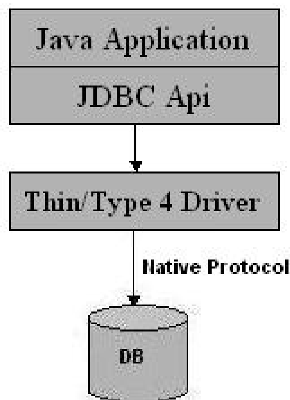


JDBC-Net/Pure Java Driver

Disadvantage: This driver requires another server application to install and maintain. Traversing the recordSet may take longer, since the data comes through the back-end server.

4) Type 4 - Native-protocol/Pure Java driver:

This driver converts JDBC API calls directly into the DBMS precise network protocol without a middle tier. This allows the client applications to connect directly to the database server.



Native-protocol/Pure Java driver

Advantage:

- The Performance of this type of driver is normally quite good.
- This driver is completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web.
- In this driver number of translation layers are very less i.e. type 4 JDBC drivers don't need to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server.
- We don't need to install special software on the client or server.

- These drivers can be downloaded dynamically.

Disadvantage: With this type of drivers, the user needs a different driver for each database.

2. Attempt *any three* of the following: 15

a. **Explain two methods of RequestDispatcher interface.**

METHODS OF REQUESTDISPATCHER

RequestDispatcher is used in two cases.

1. To include the response (output) of one Servlet into another (that is, client gets the response of both Servlets).
2. To forward the client request to another Servlet to honour (that is, client calls a Servlet but response to client is given by another Servlet).

There are two methods exist: RequestDispatcher include( ) and RequestDispatcher forward( ).

Include ( )

The *include(ServletRequest request, ServletResponse response)* method includes the content of a resource in the response. In essence, this method enables programmatic server-side includes. The ServletResponse object has its path elements and parameters remain unchanged from the caller's. The included servlet cannot change the response status code or set headers; any attempt to make a change is ignored.

The request and response parameters must be either the same objects as were passed to the calling servlet's service method or be subclasses of the ServletRequestWrapper or ServletResponseWrapper classes that wrap them.

forward ( )

The *forward(ServletRequest request, ServletResponse response)* method forwards a request from a servlet to another web component(servlet/JSP file/HTML file) on the server. This method allows one Servlet to do preliminary processing of a request and another resource to generate the response.

The forward( ) method should be called before the response has been committed to the client (before response body output has been flushed). If the response already has been committed, this method throws an IllegalStateException. Uncommitted output in the response buffer is automatically cleared before the forward.

The request and response parameters must be either the same objects as were passed to the calling servlet's service method or be subclasses of the ServletRequestWrapper or ServletResponseWrapperclasses that wrap them.

b. **Where are cookies used? Describe any four important methods of cookie class.**

Cookies can be used by web servers to identity and track users as they navigate different pages on a website, and to identify users returning to a website. The cookies can be used for following purposes:

- to recognise your computer when you visit the website
- to track you as you navigate the website, and to enable the use of any e-commerce facilities

- to improve the website's usability
- to analyse the use of the website
- in the administration of the website
- to personalise the website for you, including targeting advertisements which may be of particular interest to you
- to build a profile of your web surfing

Secure websites use cookies to validate a user's identity as they browse from page to page; without cookies, login credentials would have to be entered between before every product added to cart or wish list. Cookies enable and improve:

- Customer log-in
- Persistent shopping carts
- Wish lists
- Product recommendations
- Custom user interfaces (i.e. "Welcome back, Steve")
- Retaining customer address and payment information (only up to permitted.)

Some important methods of Cookie class are:

1. getComment( ): Returns the comment describing the purpose of this cookie, used at client side.
2. getDomain( ): Returns the domain name for the cookie. We use setDomain( ) method to set the domain name for cookie, if domain name is set then the cookie will be sent only to that particular domain requests.
3. getMaxAge( ): Returns the maximum age in seconds. We can use setMaxAge( ) to set the expiration time of cookie.
4. getName( ) : Returns the name of the cookie, can be used at both browser and server side. There is no setter for name, we can set name once through constructor only.
5. getPath( ): Returns the path on the server to which the browser returns this cookie. We will see it's example where the cookie will be sent to specific resource only. We can use setPath( ) to instruct browser to send cookie to a particular resource only.
6. 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. We can use setSecure( ) method to instruct browser to send cookie only over secured protocol.
7. getValue( ): returns the value of the cookie as String. There is also setValue( ) method to change the value of cookie.
8. getVersion( ) : Returns the version of the protocol this cookie complies with. There is also a setter method for version.
9. isHttpOnly( ): Checks whether this Cookie has been marked as HttpOnly. There is also a setter method that we can use to instruct client to use it for HTTP only.

c. **What is a session? Explain Session Management Rules.**

A session is a group of user interactions with your website that take place within a given time frame. For example a single session can contain multiple page views, events, social interactions, and ecommerce transactions.

The Session simply means a particular interval of time. Session Tracking is a way to maintain state (data) of a user. We need to maintain the state of a user to recognize to particular user.

Session Management Rules

- Session represents a conversation between client and server.
- Information or state must be stored.
- Session must execute on behalf of single client. They cannot be shared by more than one client at the same time.
- Each HTTP request must carry an identifier.
- Can be both transaction-aware and use security.
- Use an up-to-date web-server framework to generate and manage the session identifier token, as this will guarantee values that defy prediction.
- Take every precaution to ensure that the session identifier remains confidential to the application.
- Session needs to have a timeout.

d. **What is Non-Blocking I/O? How it works?**

Non-blocking I/O operations allow a single process to serve multiple requests at the same time. Instead of the process being blocked and waiting for I/O operations to complete, the I/O operations are delegated to the system, so that the process can execute the next piece of code.
The main idea of these technologies is to avoid threads blocking. That's important because blocked threads waste resources, threads' memory and processor time during threads context switching. So in some cases it's possible to increase servlet performance without any costs.
Blocking also makes your server vulnerable to thread starvation. Consider a server with 200 threads in its thread pool. If 200 requests for large content are received from slow clients, then the entire server thread pool may be consumed by threads blocking to write content to those slow clients. Asynchronous IO allows the threads to be reused to handle other requests while the slow clients are handled with minimal resources.

How it works?
The basic flow for a servlet that calls an external REST service using an async HTTP client (for example AsyncHttpClient) looks like this:

1. First of all the servlet where we want to provide **async** support should have @WebServlet annotation with asyncSupported value as true.
2. Since the actual work is to be delegated to another thread, we should have a thread pool implementation. We can create thread pool using Executors framework and use servlet context listener to initiate the thread pool.
3. We need to get instance of AsyncContext through ServletRequest.startAsync( ) method. AsyncContext provides methods to get the ServletRequest and ServletResponse object references. It also provides method to forward the request to another resource using dispatch( ) method.
4. We should have a Runnable implementation where we will do the heavy processing and then use AsyncContext object to either dispatch the request to another resource or write response using ServletResponse object. Once the processing is finished, we should call AsyncContext.complete( ) method to let container know that async processing is finished.

5. We can add AsyncListener implementation to the AsyncContext object to implement callback methods, we can use this to provide error response to client in case of error or timeout while async thread processing. We can also do some clean-up activity here.

e. **Write a servlet code to download a file.**

Sample code it may be differ from student to student

index.html

```
<a href="servlet/DownloadServlet">download the jsp file</a>
```

DownloadServlet.java

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class DownloadServlet extends HttpServlet {

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

response.setContentType("text/html");
PrintWriter out = response.getWriter();
String filename = "home.jsp";
String filepath = "e:\\";
response.setContentType("APPLICATION/OCTET-STREAM");
response.setHeader("Content-Disposition","attachment; filename=\"" + filename +
"\"");

FileInputStream fileInputStream = new FileInputStream(filepath + filename);

int i;
while ((i=fileInputStream.read()) != -1) {
out.write(i);
}
fileInputStream.close();
out.close();
}

}
```

web.xml file

This configuration file provides informations to the server about the servlet.

```
<web-app>

<servlet>
<servlet-name>DownloadServlet</servlet-name>
<servlet-class>DownloadServlet</servlet-class>
</servlet>
```

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

</web-app>
```

f. **Which things needs to be carefully checked and understood while writing file uploading code in servlet?**

1. DiskFileItemFactory is default Factory class for FileItem. When Apache commons read multipart content and generates FileItem, this implementation keep file content either in memory or in disk as temporary file, depending upon threshold size. By default DiskFileItemFactory has threshold size of 10KB and generates temporary files in temp directory, returned by System.getProperty("java.io.tmpdir"). Both of these values are configurable. You may get permission issues if user account used for running Server doesn't have sufficient permission to write files into temp directory.
2. Choose threshold size carefully based upon memory usage, keeping large content in memory may result in java.lang.OutOfMemory exception, while having too small values may result in lots of temporary files.
3. Apache commons file upload also provides FileCleaningTracker to delete temporary files created by DiskFileItemFactory. The FileCleaningTracker deletes temporary files as soon as corresponding File instance is garbage collected. It accomplish this by a cleaner thread which is created when FileCleaner is loaded. If you use this feature, than remember to terminate this Thread when your web application ends.
4. Keep configurable details e.g. upload directory, maximum file size, threshold size etc. in config files and use reasonable default values in case they are not configured.
5. It's good to validate size, type and other details of Files based upon your project requirement e.g. you may want to allow upload only images of certain size and certain types e.g. JPEG, PNG etc.

3. Attempt *<u>any three</u>* of the following: 15

a. **What is the use of Java Server Pages? Give the difference between JSP and Servlets.**

JavaServer Pages (JSP) is a technology used to develop interactive Web pages. JSP was developed by Sun Microsystems and is an improved version of Java servlets.
JSP may be developed in a simplified manner and has a wide range of applications. As with most server-based technologies, JSP separates business logic from the presentation layer.
JSPs are normal HTML pages with embedded Java code. To process a JSP file, developers need a JSP engine, which is connected to a Web server. The JSP page is then compiled into a servlet, which is handled by the servlet engine. This phase is known as translation. The servlet engine then loads the servlet class and executes it to create dynamic HTML, which is then sent to the browser. When the next page is requested, the JSP page is precompiled into the servlet and executed, unless the JSP page is

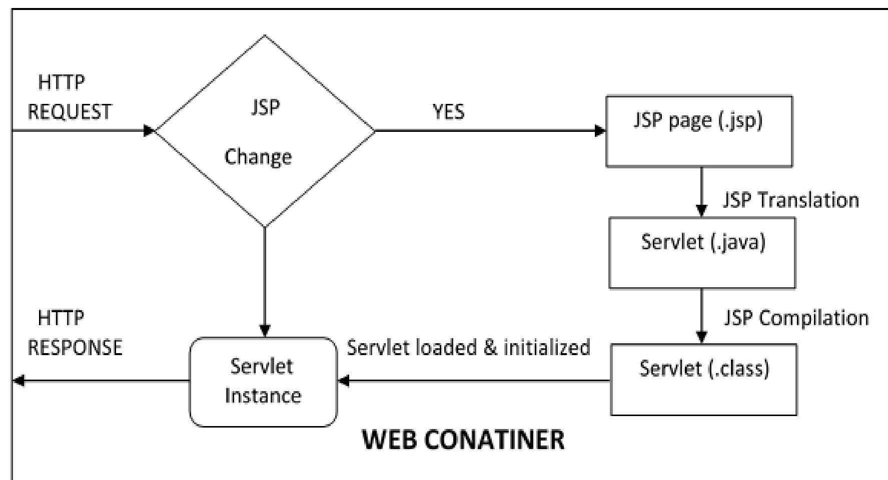
changed. When used with Java DataBase Connectivity (JDBC), JSP provides a dynamic way to create database-driven websites.

JSP V/S SERVLETS
(Any 4 expected)

- Servlet is faster than JSP, While JSP is slower than Servlet because it first translate into java code then compile.
- In Servlet, if we modify the code then we need recompilation, reloading, restarting the server> It means it is time consuming process. While In JSP, if we do any modifications then just we need to click on refresh button and recompilation, reloading, and restart the server is not required.
- Servlet is a pure java code. While JSP is tag based approach.
- In Servlet, there is no such method for running JavaScript at client side. While In JSP, we can use the client side validations using running the JavaScript at client side.
- To run a Servlet you have to make an entry of Servlet mapping into the deployment descriptor file i.e. web.xml file externally. While For running a JSP there is no need to make an entry of Servlet mapping into the web.xml file externally, you may or not make an entry for JSP file as welcome file list.
- Coding of Servlet is harden than JSP. While Coding of JSP is easier than Servlet because it is tag based.
- In MVC pattern, Servlet plays a controller role. While In MVC pattern, JSP is used for showing output data i.e. in MVC it is a view.
- Servlet accept all protocol request. While JSP will accept only http protocol request.
- In Servlet, Service( ) method need to override. While In JSP no need to override service( ) method.
- In Servlet, by default session management is not enabled we need to enable explicitly. While In JSP, session management is automatically enabled.
- In Servlet we do not have implicit object. It means if we want to use an object then we need to get object explicitly form the servlet. While In JSP, we have implicit object support.
- In Servlet, we need to implement business logic, presentation logic combined. While In JSP, we can separate the business logic from the presentation logic by uses JavaBean technology.
- In Servlet, all package must be imported on top of the servlet. While In JSP, package imported anywhere top, middle and bottom.

b. **Explain Life Cycle of a JSP Page.**

The page cant outputs the content directly to the browser it relies on early server side processing process which translates JSP page into the JSP Page class. The page class then will handle all the requests made of JSP.



JSP's life cycle can be grouped into following phases.

1. JSP Page Translation: A java Servlet file is generated from the JSP source file. This is the first step in its dull multiple phase life cycle. In the translation phase, the container validates the syntax of the JSP page and tag files for correctness. The container interprets the standard directives and actions, and the custom actions referencing tag libraries used in the page.
2. JSP Page Compilation: After the page translation the generated java Servlet file is compiled into a java Servlet class. The translation of a JSP page into its implementation class can happen at any time between initial deployment of the JSP page into the JSP container and the receipt and processing of a client request for the target JSP page.
3. Class Loading: The java Servlet class that was compiled from the JSP source is loaded into the container.
4. Execution phase: In this phase the container adjust one or more instances of this class in response to requests and other events. Interface *JspPage* contains jspInit( ) and jspDestroy( ) method. The JSP specification has provided a special interface *HttpJspPage* for JSP pages serving HTTP requests and this interface contains _jspService( ) method.
5. Initialization: jspInit( ) method is called instantly after the instance was created. It is called only once during JSP life cycle.
6. _jspService( ) execution: This method is called for every request of this JSP during its life cycle. This is where it serves the purpose of creation. It has to pass through all the above steps to reach this phase. It passes the request and the response objects. _jspService( ) method cannot be overridden.
7. jspDestroy( ) execution: This method is called when this JSP page is destroyed. With this call the Servlet serves its purpose and submits itself to garbage collection. This is the end of JSP life cycle.

c. **What are directives in JSP? Explain its types.**

JSP directives are JSP elements that send messages to the JSP container. Directives affect the overall structure of the Servlet generated from the JSP page. JSP directives do not produce any output to the generated document. The general format of a JSP directive is:

<%@ directiveType attributes %>

The first word, directiveType, is one of three values: page, include, and taglib.
The second word, attributes, is one or more name-value pairs; the name part is the name of the attribute relevant to the directive type and the value is a quoted string relevant to the attribute name.
There are three types of directives:

- page directive
- include directive
- taglib directive

The page Directive: Using the page directive, we can instruct the Servlet to import classes, define a general error reporting page, or set a content type. The page directive specifies attributes for the page.
Syntax:
<%@ page language="java" %>

The include Directive: The include directive allows us to include code from another file, which is useful for the purpose of reusability.
Syntax:
<% @include file="menu.JSP" %>

The taglib Directive: The JSP "taglib" directive is use to define tag library, which is the collection of tags and it also defines the prefix for the tags.
Syntax:
<%@ taglib uri="URLoFTagLibrary" prefix="tagPrefix" %>

(One short example expected.)

d. **Give an explanation of the jsp:useBean action tag's attributes and usage.**
The JSP:useBean action tag is used to locate or instantiate a bean class.
Attributes and Usage of jsp:useBean action tag

1. id: is used to identify the bean in the specified scope.
2. scope: represents the scope of the bean. It may be page, request, session or application. The default scope is page.
    - page: specifies that you can use this bean within the JSP page. The default scope is page.
    - request: specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
    - session: specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
    - application: specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
3. class: instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
4. type: provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.

5. beanName: instantiates the bean using the java.beans.Beans.instantiate( ) method.

e. **Write a short note on <JSP:plugin> Action.**

The plugin action is used to insert Java components into a JSP page. It determines the type of browser and inserts the <object> or <embed> tags as needed. If the needed plugin is not present, it downloads the plugin and then executes the Java component. The Java component can be either an Applet or a JavaBean.

The plugin action has several attributes that correspond to common HTML tags used to format Java components. The <param> element can also be used to send parameters to the Applet or Bean.

Syntax:

```
<JSP:plugin type="applet" codebase="dirname" code="MyApplet.class"
width="60" height="80">
        <JSP:param name="fontcolor" value="red" />
        <JSP:param name="background" value="black" />

        <JSP:fallback>   Unable to find Java Plugin </JSP:fallback>
</JSP:plugin>
```

We can try this action using some applet if we are interested. A new element, the <fallback> element, can be used to specify an error string to be sent to the user in case the component fails.

f. **What exactly is JSTL? Describe XPath in detail.**

JSTL

The JavaServer Pages Standard Tag Library (JSTL) encapsulates core functionality common to many JSP applications. For example, instead of iterating over lists using a scriptlet or different iteration tags from numerous vendors, JSTL defines a standard tag that works the same everywhere. This standardization lets you learn a single tag and use it on multiple JSP containers. Also, when tags are standard, containers can optimize their implementation.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and tags for accessing databases using SQL. It also introduces the concept of an expression language to simplify page development. JSTL also provides a framework for integrating existing tag libraries with JSTL. JSTL is composed of:

An expression language

Standard Action Libraries (42 actions in four libraries)

Tag Library Validators (2 validators)

XPath

An XPath expression generally defines a pattern in order to select a set of nodes. These patterns are used by XSLT to perform transformations or by XPointer for addressing purpose. XPath specification specifies seven types of nodes which can be the output of execution of the XPath expression. XPath uses a path expression to select node or a list of nodes from an XML document.

The XML tags use XPath as a local expression language; XPath expressions are always specified using attribute select. This means that only values specified for select attributes are evaluated using the XPath expression language. All other attributes are evaluated using the rules associated with the global expression language.

The following table shows the scopes available within an XPath expression to access web application data, are mapped in the similar way as their EL Implicit Objects counterparts:
$foo pageContext.findAttribute("foo")
$param:foo request.getParameter("foo")
$header:foo request.getHeader("foo")
$cookie:foo Maps to cookie values for foo
$initParam:fooapplication.getInitParameter("foo")
$pageScope:foo pageContext.getAttribute("foo", PageContext.PAGE_SCOPE)
$requestScope:foo pageContext.getAttribute("foo", PageContext.REQUEST_SCOPE)
$sessionScope:foo pageContext.getAttribute("foo", PageContext.SESSION_SCOPE)
$applicationScope:foopageContext.getAttribute("foo", PageContext.APPLICATION_SCOPE)

4. Attempt *any three* of the following: 15

a **What is EJB? Explain its advantages.**

Enterprise JavaBeans (EJB) is the server-side component-based middleware standard for Java. The specification aims at unifying application server vendors to support a standard architecture so that compliant business applications are not only platform independent, but vendor independent.

EJB is intended for complex distributed systems that use a purchased application server to handle the plumbing or low-level technical workload. Encapsulation and separation of the technical services layer from the business application layer allows application developers to focus on solving business problems.

ADVANTAGES OF ENTERPRISE BEAN

(Any 4 expected)

1) **Provides Architectural Independence:** EJB prevents developers from the underlying middleware, since EJB is only concern about the Java environment. It also helps the EJB server/container vendor to change and make improvements on the underlying middleware layer without affecting a user's existing enterprise applications.
2) **EJB creates a base for Application Development:** The EJB specification assigns specific roles for project participants with enterprise application development using EJB. The server vendor provides support for complex system services and makes available an organized framework for a Bean to execute in, without assistance from Bean developers.
3) **Provides Distributed Transaction support:** EJB provides transparency for distributed transactions. This means that a client can begin a transaction and then invoke methods on Beans present within two different servers, running on different machines, platforms or JVM.
4) **WORA for server side components:** Since EJB is based on Java technology, both the developer and the user are guaranteed that their components are Write Once, Run Anywhere (WORA). As long as an EJB Server devotedly conforms to the EJB specification, any EJB application should run within that server.
5) **Helps to create Portable and Scalable solutions:** Beans conforming to the EJB API will install and run in a portable fashion on any EJB server.

6) **Provides of vendor specific enhancements:** Since the EJB specification provides a lot of flexibility for the vendors to create their own enhancements, the EJB environment may end being feature rich.
7) **Developer productivity:** The EJB architecture improves the productivity of application developers by standardizing and automating the use of complex infrastructure services such as transaction management and security checking.
8) **Customization**: Enterprise bean applications can be customized without access to the source code. Application behaviour and runtime settings are defined through attributes that can be changed when the enterprise bean is deployed.
9) **Superior transaction management:** Enterprise beans in a EJB server benefit from transaction management services

b **Write a detailed note on the type of Session beans.**

Session Beans

A Session Bean is an Enterprise Bean that is generated for each session from the client and expires when the client exits. The lifecycle of the Session Bean does not exceed the range from the beginning until the end of the usage of the system by the user.

Types of Session Beans

There are three types of session beans:

1) Stateless Session Bean: Stateless Session Beans are business objects that do not have state associated with them. Access to a single bean instance is limited to only one client at a time and thus concurrent access to the bean is banned. In case concurrent access to a single bean is attempted anyway the container simply routes each request to a different instance. Instances of Stateless Session beans are typically pooled. If a second client accesses a specific bean right after a method call on it made by a first client has finished, it might get the same instance.

Example: Sending an Email to customer support may be handled by a stateless bean since this is a one off operation and not part of a multi-step process.

2) Stateful Session Bean: Stateful Session Beans are business objects having state, means they can keep track of which calling client they are dealing with throughout a session and thus access to the bean instance is strictly limited to “only one client at a time”. In the case of concurrent access to a single bean is attempted anyway the container serializes those requests, but via the @AccessTimeout annotation the container can throw an exception instead. Stateful session beans' state may be persisted automatically by the container to free up memory after the client hasn't accessed the bean for some time.

Example: The hotel check out may be handled by a stateful session bean that would use its state to keep track of where the customer is in the checkout process, possibly holding locks on the items the customer is charged for services.

3) Singleton Session Bean: Singleton Session Beans are business objects having a global shared state in a JVM. Concurrent access to the one and only bean instance can be controlled by the container or by the bean itself. Container Managed concurrency can be tuned using the @Lock annotation, that designates whether a read lock or a write lock will be used for a method call. Also the Singleton Session Beans can explicitly request to be instantiated when the EJB container starts up, using the @Startup annotation.

c **Write a short note on Remote and Local Interfaces.**

When you design a Java EE application, one of the first decisions you make is the type of client access allowed by the enterprise beans, either Local or Remote.
The session bean can implement multiple interfaces. Local interface are the type of interface that are used for making local connections to EJB. The @Local annotation is used for declaring interface as Local. The javax.ejb.Local package is used for creating Local interface. Remote interface are the interface that has the methods that relate to a particular bean instance. In the Remote interface we have all get methods as given below in the program. This is the interface where all of the business method go. The javax.ejb.Remote package is used for creating Remote interface.

Whether to allow local or remote access depends on the following factors.

- Tight or loose coupling of related beans: Tightly coupled beans depend on one another. For example, if a session bean that processes sales orders calls a session bean that emails a confirmation message to the customer, these beans are tightly coupled. Tightly coupled beans are good candidates for local access. Because they fit together as a logical unit, they typically call each other often and would benefit from the increased performance that is possible with local access.

- Type of client: If an enterprise bean is accessed by application clients, it should allow remote access. In a production environment, these clients almost always run on machines other than those on which the Glassfish Server is running. If an enterprise bean's clients are web components or other enterprise beans, the type of access depends on how you want to distribute your components.

- Component distribution: Java EE applications are scalable because their server-side components can be distributed across multiple machines. In a distributed application, for example, the server that the web components run on may not be the one on which the enterprise beans they access are deployed. In this distributed scenario, the enterprise beans should allow remote access.

- Performance: Owing to such factors as network latency, remote calls may be slower than local calls. On the other hand, if you distribute components among different servers, you may improve the application's overall performance. Both of these statements are generalizations; performance can vary in different operational environments. Nevertheless, you should keep in mind how your application design might affect performance.

If you aren't sure which type of access an enterprise bean should have, choose remote access. This decision gives you more flexibility.

d **Describe message-driven beans characteristics.**

Message-driven beans have the following characteristics:

- A message-driven bean's instances retain no data or conversational state for a specific client i.e. they are stateless.
- A single message-driven bean can process messages from multiple clients.
- They are invoked asynchronously.
- They can be transaction-aware.
- They do not represent directly shared data in the database, but they can access and update this data.

- A message-driven bean has only a bean class i.e. unlike a session bean, the clients don't access message-driven beans through interfaces.
- They don't have the remote or local interfaces that define client access.

e **What is a naming service?**

A fundamental facility in any computing system is the *naming service*--the means by which names are associated with objects and objects are found based on their names. When using almost any computer program or system, you are always naming one object or another. For example, when you use an electronic mail system, you must provide the name of the recipient. To access a file in the computer, you must supply its name. A naming service allows you to look up an object given its name.

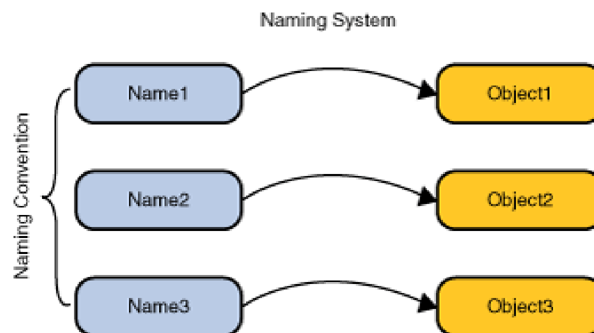
Naming service provides a mechanism to name objects and to retrieve objects by name. If you want to associate an object with a particular name, you have to bind it. Once the object is bound, it can be accessed through the lookup operations and through the operations for working with names (rename, rebind, unbind.).

Some objects cannot be stored directly so they are put in the system as references.

A naming service's primary function is to map people friendly names to objects, such as addresses, identifiers, or objects typically used by computer programs.

For example, the Internet Domain Name System (DNS) maps machine names to IP Addresses:

www.test.com ==> 192.0.2.5



A file system maps a filename to a file reference that a program can use to access the contents of the file.

c:\MyDoc\autocad.exe ==> File Reference

These two examples also illustrate the wide range of scale at which naming services exist--from naming an object on the Internet to naming a file on the local file system.

f **Write a note on DataSource Resource Definitio in Java EE 7.**

DataSource resources are used to define a set of properties required to identify and access a database through the JDBC API. These properties include information such as the URL of the database server, the name of the database, and the network protocol to use to communicate with the server. DataSource objects are registered with the Java Naming and Directory Interface (JNDI) naming service so that applications can use the JNDI API to access a DataSource object to make a connection with a database.

Java EE 7 provides the option to programmatically define DataSource resources for a more flexible and portable method of database connectivity.

The name element uniquely identifies a DataSource and is registered with JNDI. The value specified in the name element begins with a namespace scope. Java EE 7 includes the following scopes:

- java:comp: Names in this namespace have per-component visibility.
- java:module: Names in this namespace are shared by all components in a module, for example, the EJB components defined in an a ejb-jar.xml file.
- java:app: Names in this namespace are shared by all components and modules in an application, for example, the application-client, web, and EJB components in an .EAR file.
- java:global: Names in this namespace are shared by all the applications in the server.
- You can programmatically declare DataSource definitions using one of the following methods:
- Creating DataSource Resource Definitions Using Annotations
- Creating DataSource Resource Definitions Using Deployment Descriptors

5. **Attempt *any three* of the following:** **15**

a. **Explain following:**

I. JOIN condition Using ON
II. Entity Listeners Using CDI

JOIN condition Using ON

JOIN expressions in JPA are already a bit different from JOIN in standard SQL. It is possible to use JOIN only when a mapping between entities already exists, and is not always necessary due to lazy loading of related collections using implicit joins. JPA provides an additional feature to standard ON keyword.

The **ON** keyword makes possible to filter results after every join, leading to smaller result after each successive join. However, one limitation still remains even when using JOIN with ON – entities can still be joined only when they are mapped together as related entities.

It makes possible to relate unrelated entities in the ON condition, making it possible to JOIN an unrelated entity to other entities already in the query. Therefore, it does not require that fields are mapped as related. This is convenient especially if we need the join condition only for a single report and we don't want to update our mappings.

Example:

```
SELECT p FROM Person p LEFT JOIN Person p2 ON p2.city = p.city
SELECT e FROM Employee e LEFT JOIN e.address ON a.city = :city
```

Entity Listeners Using CDI

Entity Listeners allow to handle cross-cutting lifecycle events in a non-persistent listener class. In JPA 2.1, entity listeners will support dependency injection through CDI. The usual lifecycle **callback** methods of **@PrePersist**, **@PostPersist**, **@PreUpdate** and **@PreRemove** can be used for entities. The entity listeners can also be annotated with **@PostConstruct** and **@PreDestroy** for their own lifecycle.

Example:

```
@Entity
@EntityListeners(Alarm.class)
public class Customer {
@Id private Integer id;
private String name;
...
```

```
}
public class Alarm {
  @PostPersist
  public void alert(Customer c) {
    ...
  }
}
```

b. **Where Does Java Persistence API Fit In?**

There are several persistent frameworks and ORM options in Java. A persistent framework is an ORM service that stores and retrieves objects into a relational database. Such as: Enterprise JavaBeans Entity Beans, Java Data Objects, Castor, TopLink, Spring DAO, Hibernate and many more.

These tools helps developer to focus on object model instead of dealing with mismatch of object and table columns. As downside every tool mentioned above comes with their own API. So its means Code Specs are tied with specific vendor which gives problem to switch to another tool with previous code.

A) Java Persistence API: The Java Persistence API (JPA) defines the management of persistence and object/relational mapping within Java Enterprise Edition (Java EE) and Java Standard Edition (Java SE) environments.

The Java Persistence API (JPA) represents a simplification of the persistence programming model. JPA manages persistence and object/relational mapping within the Java EE specification for Enterprise Java Beans 3.0. The JPA specification defines the object/relational mapping within its own guidelines instead of relying on vendor-specific mapping implementations. These features make applications that use JPA easier to implement and manage.

JPA combines the best features from previous persistence mechanisms such as Java Database Connectivity (JDBC) APIs, Object Relational Mapping (ORM) frameworks, and Java Data Objects (JDO). Creating entities under JPA is as simple as creating Serializable classes. JPA supports the large data sets, data consistency, concurrent use, and query capabilities of JDBC. JPA allows the use of advanced object-oriented concepts such as inheritance. JPA avoids vendor lock-in because it does not rely on a strict specification like JDO and EJB 2.x entities.

B) Java Persistence API as A Specification: The JPA specification defines the object-relational mapping internally, rather than relying on vendor-specific mapping implementations. JPA is based on the Java programming model that applies to Java Enterprise Edition (Java EE) environments, but JPA can function within a Java SE environment for testing application functions.

The JPA specification explicitly defines the object-relational mapping, rather than relying on vendor-specific mapping implementations. JPA standardizes the important task of object-relational mapping by using annotations or XML to map objects into one or more tables of a database. To further simplify the persistence programming model:

- The **EntityManager** API can persist, update, retrieve, or remove objects from a database.

- The **EntityManager** API and **object-relational mapping metadata** handle most of the database operations without requiring to write JDBC or SQL code to maintain persistence.
- **JPA provides a query language**, extending the independent EJB querying language (also known as **JPQL**), that you can use to retrieve objects without writing SQL queries specific to the database that you are working with.

JPA is designed to operate both inside and outside of a Java Enterprise Edition (Java EE) container. When you run JPA inside a container, the applications can use the container to manage the persistence context. Applications that are designed for container-managed persistence do not require as much code implementation to handle persistence, but these applications cannot be used outside of a container. Applications that manage their own persistence can function in a container environment or a Java SE environment.

c. **Why is there a need for Object Relational Mapping (ORM)?**

ORM performs the rather amazing task of managing the application's interactions with the database. Once used an ORM's tools to create mappings and objects for use in an application, those objects completely manage the application's data access needs. You won't have to write any other low-level data access code. You could still write low-level data access code to supplement the ORM data objects, but this adds a significant layer of complexity to an application that we've rarely found necessary when using a robust ORM tool. It is better to keep the application simpler and more maintainable.

Benefits of using an ORM for development of data based applications:

1. **Productivity:** The data access code is usually a significant portion of a typical application, and the time needed to write that code can be a significant portion of the overall development schedule. When using an ORM tool, the amount of code is unlikely to be reduced, but the ORM tool generates 100% of the data access code automatically based on the data model you define, in few moments.
2. **Application design:** A good ORM tool designed by very experienced software architects will implement effective design patterns that almost force you to use good programming practices in an application. This can help support a clean separation of concerns and independent development that allows parallel, simultaneous development of application layers.
3. **Code Reuse:** If you create a class library to generate a separate DLL for the ORM generated data access code, you can easily reuse the data objects in a variety of applications. This way, each of the applications that use the class library need have no data access code at all.
4. **Application Maintainability:** All of the code generated by the ORM is presumably well-tested. We need to make sure that the code does what we need, but a widely used ORM is likely to have code banged on by many developers at all skill levels. Over the long term, we can refactor the database schema or the model definition without affecting how the application uses the data objects.

d. **Write a short note on Functions in JPQL and Downcasting in JPQL.**

Functions in JPQL

JPA supports a set of database functions which you can use to perform small operations and transformations within a query. This is often easier and faster than doing it in the Java code. SQL supports a lot more database functions than JPQL. Specific database vendors also provide their own set of functions. Users and libraries can also define their

own database functions. JPA 2.0 JPQL provided no mechanism to call database specific functions.
JPA 2.1 introduced ***function( )*** to call database functions which are not directly supported by the standard. The syntax is very easy. You need to provide the name of the function as the first parameter and then all parameters of the custom function. In following example, the name of the function is "calculate" and I provide the numbers 1 and 2 as parameters.
Example:

```
Author a = em.createQuery("SELECT a FROM Author a WHERE a.id =
            function('calculate', 1, 2)", Author.class).getSingleResult( );
```

Downcasting in JPQL
JPQL will be extended to cast in the FROM and WHERE clause. The format of this will use the keyword "**TREAT**" and be part of the join clause. The operator TREAT can be used to cast an entity to its subclass value, i.e. downcast related entities with inheritance. This is typically used in JOIN queries where we want to impose a restriction in WHERE clause involving a subclass field. The following is an example:
Example:

```
select e from Employee e join TREAT(e.projects AS LargeProject) lp
where lp.attribute = value
```

e. **What is Hibernate? Explain the features of Hibernate.**
Introduction to Hibernate
Hibernate is an open source object/relational mapping tool for Java. It provides help for mapping from Java classes to database tables; it also provides data query and retrieval facilities and can significantly reduce development time otherwise spent with manual data handling in SQL and JDBC. Hibernate is most useful with object-oriented domain modes and business logic in the Java-based middle-tier. It allows transparent persistence that enables the applications to switch to any database. Hibernate can be used in Java Swing applications, Java Servlet-based applications, or J2EE applications using EJB session beans. Hibernates is used to relieve the developer from 90 to 95 percent of common data persistence related programming tasks.
Hibernate allows us develop persistent classes using common Java language, including association, inheritance, polymorphism, composition and the Java collections framework. Hibernate is Free Software. The LGPL license is sufficiently flexible to allow the use of Hibernate in both open source and commercial projects. Hibernate is available for download at http://www.hibernate.org/.

Features of Hibernate

- Hibernate is Free Resource.
- Hibernate provides full-featured query facilities as *Hibernate Query Language*, *Query API*, and *native SQL dialect* of the database.
- It provides Filters for working with historical, regional or permissible data.
- It provides full support for projection/aggregation and sub selects.
- Hibernate allows *Runtime performance monitoring* using JMX or local Java API.
- Hibernate applications are scalable due to its dual-layer architecture & can be used in the clustered environments.
- Hibernate reduces the development timings as it supports inheritance, polymorphism, composition and the Java Collection framework.

- Hibernate supports the automatic generation of primary key for our data application.
- It supports Java generics, which basically boils down to allowing type safe collections.
- Hibernate3 will support all four operations as methods of the Session interface.
- Hibernate's XML binding enables data to be represented as XML and POJOs interchangeably.
- Hibernate support the EJB3 draft specification for POJO persistence and annotations.

f. **Explain the components of Hibernate configurations.**

1) SessionFactory (org.hibernate.SessionFactory): It is a thread safe, absolute cache of compiled mappings for a single database. It is a factory for "*org.hibernate.Session*" instances & a client of "*org.hibernate.connection.ConnectionProvider*". This optionally maintains a second level cache of data that is reusable between transactions at a process or cluster level.

2) Session (org.hibernate.Session): It is a single threaded, short-lived object representing a conversation between the application and the persistent store. It wraps a JDBC **java.sql.Connection**. It is a Factory for "*org.hibernate.Transaction*". It maintains a first level cache of persistent the application's persistent objects and collections; this cache is used when navigating the object graph or looking up objects by identifier.

3) Persistent objects and collections: It is a short lived, single threaded objects containing persistent state and business function. These can be ordinary JavaBeans or POJOs. They are associated with exactly one "*org.hibernate.Session*". Once the **org.hibernate.Session** is closed, they will be disconnected and free to use in any application layer.

4) Transient and detached objects and collections: It is an instances of persistent classes that are not currently associated with a **org.hibernate.Session**. They may have been instantiated by the application and not yet persisted, or they may have been instantiated by a closed **org.hibernate.Session**.

5) Transaction (org.hibernate.Transaction): It is a single threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC, JTA or CORBA transaction. An "*org.hibernate.Session*" might span several **org.hibernate.Transactions** in some cases. Transaction separation in the underlying API or **org.hibernate.Transaction** is never optional.

6) ConnectionProvider (org.hibernate.connection.ConnectionProvider): It is a factory for, and pool of, JDBC connections. It abstracts the application from underlying **javax.sql.DataSource** or **java.sql.DriverManager**. It is not open to the elements of application, but it can be extended or implemented by the developer.

7)TransactionFactory (org.hibernate.TransactionFactory): It is a factory for "*org.hibernate.Transaction*" instances. It is not open to the elements of the application, but it can be extended or implemented by the developer.