Bachelor of Science in Information Technology

ENTERPRISE JAVA(EJ)

CBCS (NOV-2019) Q.P.Code:53706

1. Attempt any three of the following:

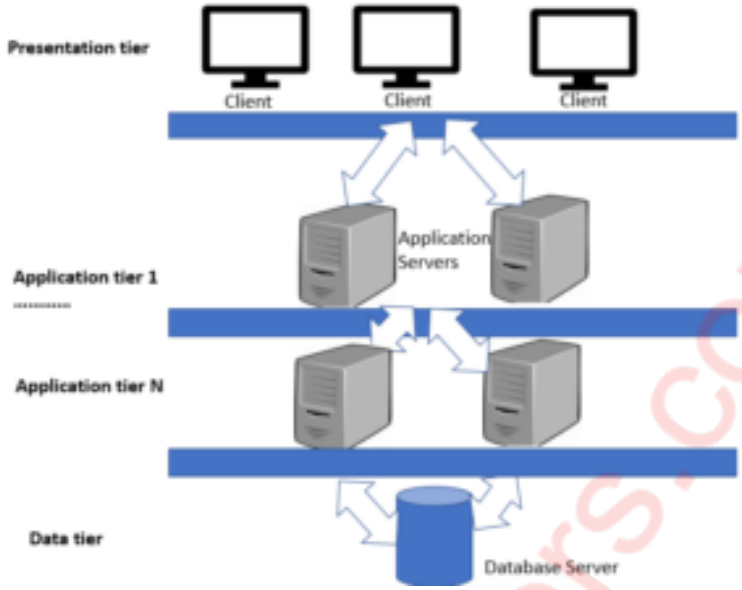
a. What is Java Enterprise Edition (Java EE)? Explain. (5)

- J2EE defines the standard for developing multitier enterprise applications. It stands for **J**ava **2** Platform **E**nterprise **E**dition.
- J2EE is a platform-independent, Java-centric environment from Sun/Oracle for developing, building and deploying Web-based enterprise application online.
- The J2EE platform consists of a set of services, APIs and protocols that provide the functionality for developing multi-tiered, Web-based applications.
- It simplifies enterprise applications by :
    - Standardized modular components.
    - Providing a complete set of services to those modular components.
    - Automatically handling many details of application behaviour without complex programming.
- The Java EE platform is developed through the Java Community Process (JCP), which is responsible for all Java technologies.
- Java EE does not compete with Java SE, but is instead a superset of APIs that builds upon the foundation provide by Java SE and the standard Java Development Kit(JDK) and all Java EE applications run on a Java virtual machine that supports all of the APIs defined by Java SE.

b. Write a note on Multi Tier EE application architecture. (5)

- An extension of 3-tier architecture is nothing but n-tier(MultiTier) architecture.

- Service layer is one machine, data layer in one machine, presentation layer is in morethan one machine. i.e. windows based enterprise application can be 3-tier architecture cannot be 'n' tier as presentation layer runs in only one machine as one process.
- Web-enabling a 3-tier architectural enterprise application is nothing but making it n-tiered.
- The Java EE platform uses a distributed multitiered application model for enterprise applications. Application logic is divided into components according to function, and the application components that make up a Java EE application are installed on various machines depending on the tier in the multitiered Java EE environment to which the application component belongs.
- Tiers separate functionality:
  - Presentation Logic, Business Logic, Data Schema.
- Code that generates input screens and representing pages for the end-user is known as presentation logic.
- Data processing logic according to the business rules of the organization is nothing but business logic.
- Flow control logic is known as application logic.
- Easier upgrade since one tier can be changed without changing the rest.
- Lower deployment and maintenance cost.
- More flexible.
- More extensible(can add functionality).



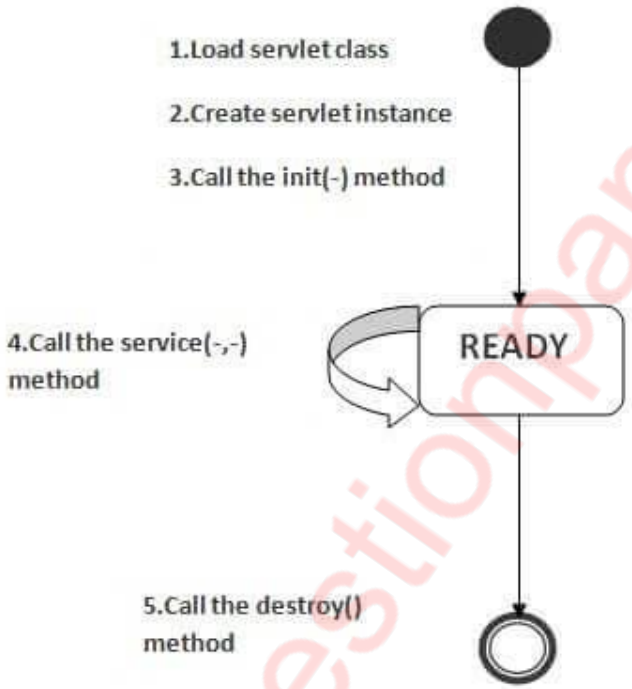
c. List and explain the tasks that Servlet can do. (5)

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

d. Explain the life cycle of servlet.

– A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.



– Now let us discuss the life cycle methods in detail.

➢ **The init() Method**

– The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

– The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.
- The init method definition looks like this –
  public void init() throws ServletException{
  // Initialization code…
  }

➢ **The service() Method**

- The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- Here is the signature of this method –
- public void service(ServletRequest request, ServletResponse response)
  throws ServletException, IOException {
  }
- The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.
- The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

➢ **The doGet() Method**

- A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.
- Public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
  // Servlet code
  }

- **The doPost() Method**

– A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
Public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Servlet code
}
```

- **The destroy() Method**

– The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

– After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this −

```
Public void destroy() {

// Finalization code….

}
```

e. Write a servlet code to display Square and Square root of numbers between 25 and 365 in tabular form. (5)

```
import java.io.*;
import javax.servlet.*;
import javax.servelt.http.*;
public class Test extends HttpServlet{
public void doGet(HttpServletRequest hreq, HttpServletResponse hres) throws
ServletException, IOException{
PrintWriter pw = hres.getWriter();
out.println("<table border=1>”);
for(int i=25;i<=365;i++){
out.println(“<tr><td>”+(i*i)+”</td><td>”+Math.sqrt(i)=”</td></tr>”);
}
Out,println(“</table>”);
}
```

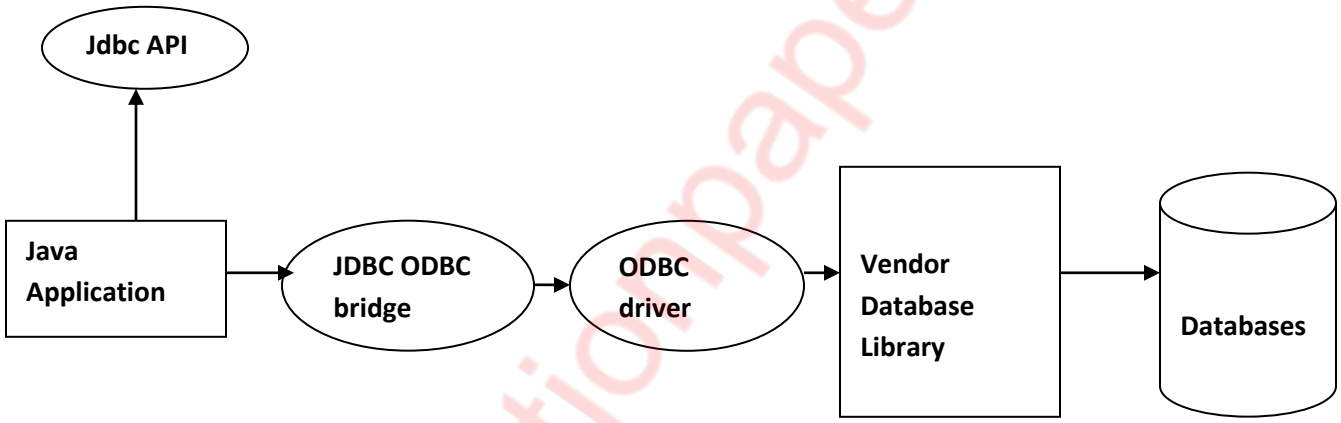
}

f. List and explain four types of JDBC drivers. (5)

– JDC driver implementation vary because of the wide variety of operating systems and hardware platforms in which java operates. Sun has divided the implementation types into four categories. Type 1,2,3 and 4, which is explained below:

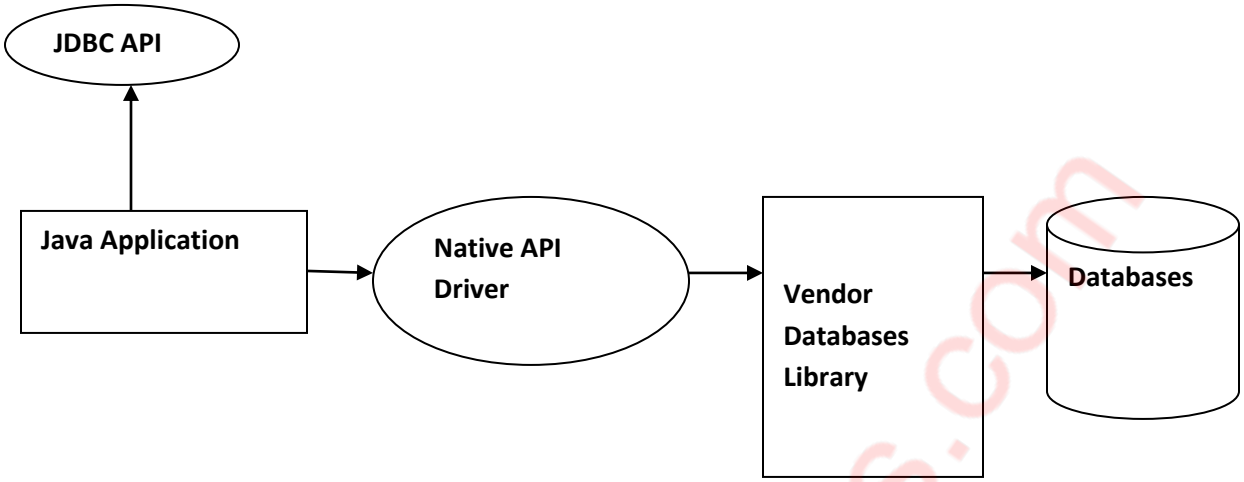
a) Type 1: JDBC-ODBC Bridge Driver

– In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.



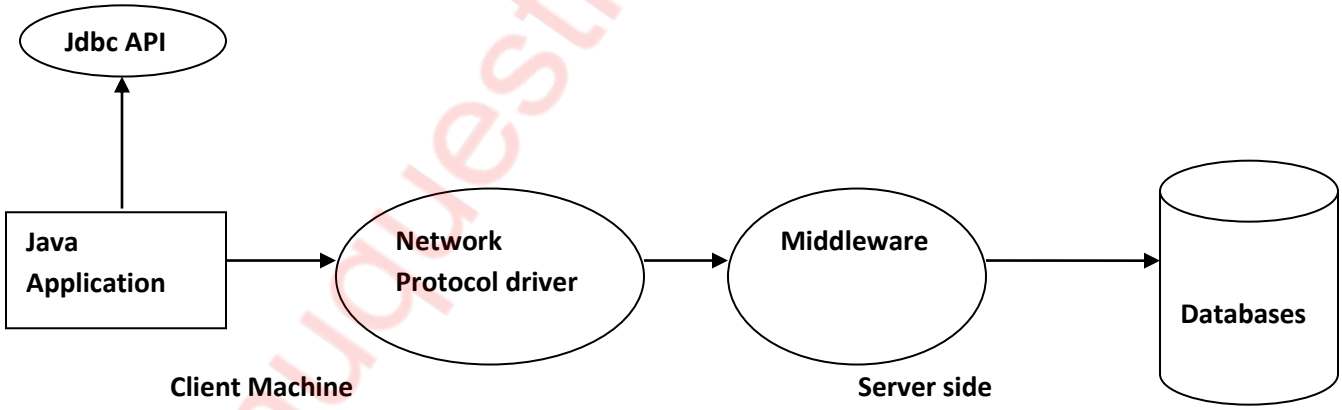
b) Type 2: JDBC-Native API

– In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor specific driver must be installed on each client machine.



c) Type 3: JDBC-Net pure Java

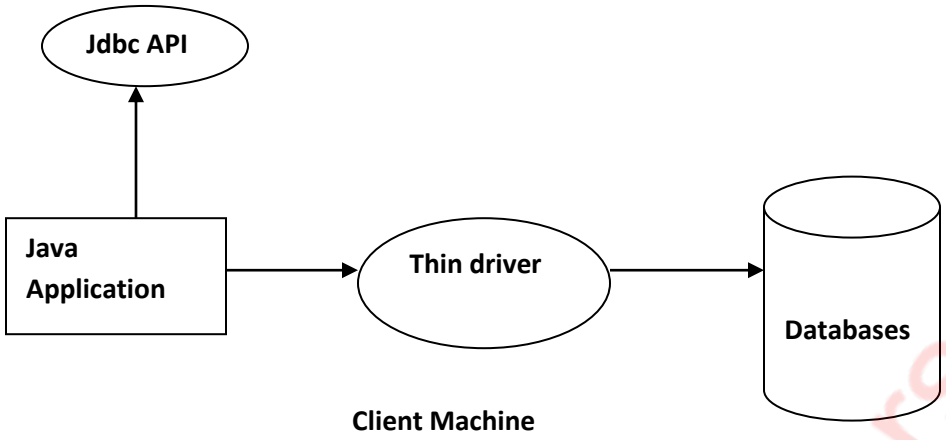
– In a Type 3 driver, a three-tier approach is used to acess databases. The JDBC clients use standard networks sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.



d) Type 4: 100% Pure Java (Thin Driver)

– In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket communication. This is the highest

performance driveravailable for the database and is usually provided by the vendor itself.



2. Attempt any three of the following:

a. Explain Cookie Class with its constructor and any five methods.

- A cookie is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as comment, path and domain qualifiers, a maximum age, and version number.
- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie in the cache of the browser. After that if request it sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.
- **Constructor of Cookie Class**
  - **Cookie()** – To construct a cookie.
  - **Cookie(String name, String value)** – To construct a cookie with a specified name and value.

1) Methods of Cookie Class

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() –** To return the value of the cookie.
4. **Public void setName(String name) –** To changes the name of the cookie.
5. **Public void setValue(String value) –** To changes the value of the cookie.

b. Write a servlet program to create a session and display the following:

i. Session ID ii. New or Old iii. Creation Time

```
import java.io.*;
import javax.servlet.*;
import javax.servelt.http.*;
public class Test extends HttpServlet{
public void doGet(HttpServletRequest hreq, HttpServletResponse hres) throws
ServletException, IOException{
PrintWriter pw = hres.getWriter();
HttpSession hs= hreq.getSession(true);
Out.println("<br>Session ID  "+hs.getId());
Out.println("<br>Is New  "+hs.isNew());
Out.println("<br>CreationTime"+new java.util.Date(hs.getCreationTime()));
}
}
```

C. Write a servlet program GradeServlet.java that accepts Grade through radio buttons from index.html page, if the string is "A", "B", "C" OR "D", the program should dispatch the direct to the Success.html page containing message "Congratulations, You Passed SEM V exam", else display "Try Again" and load index.html page in current servlet.

```
import java.io.*;
import javax.servlet.*;
import javax.servelt.http.*;
public class Test extends HttpServlet{
public void doGet(HttpServletRequest hreq, HttpServletResponse hres)
throws
ServletException, IOException{
PrintWriter pw = hres.getWriter();
String inp = hreq.getParameter("grade");
if(inp.equals("A") || inp.equals("B") || inp.equals("C") || inp.equals("D") )
{
RequestDispatcher rd = hreq.getRequestDispatcher("Success.html");
rd.forward(hreq, hres);
}
else {
out.println("<H1> TRY AGAIN </H1>");
RequestDispatcher rd = hreq.getRequestDispatcher("index.html");
rd.include(hreq, hres);
}
}
```

d. Explain the following w.r.t. working with files in Servlet . (5)

1. **@MultipartConfigure**
2. **fileSize Threshold**
3. **location**
4. **maxFileSize**
5. **maxRequestSize**

1) **@MultipartConfigure** - The @MultipartConfig annotation is used to annotate a servlet class in order to handle multipart/form-data requests and configure various upload settings. When a servlet is annotated by this annotation, we can access all parts via the methods getParts() and an individual part via the method getPart(name) of the HttpServletRequest object, and write the upload file to disk via the method write (fileName) of the part object.
2) **fileSize Threshold -** Specify size threshold when saving the upload file temporarily. If the upload file's size is greater than this threshold, it will be stored in disk. Otherwise the file is stored in memory. Size in bytes.
3) **Location -** Specify directory where upload files are stored.
4) **maxFileSize -** Specify maximum size of an upload file. Size in bytes.
5) **maxRequestSize -** Specify maximum size of a request (including both upload files and other form data). Size in bytes.

e. Explain using a code snippet the onDataAvailable() and onAllDataRead() methods of ReadListener interface. (5)

1) **ReadListener interface** - servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container. This class represents a call-back mechanism that will notify implementations as HTTP request data becomes available to be read without blocking.

2) **onDataAvailable() –** void onDataAvailable() throws IOException

When an instance of the ReadListener is registered with a ServletInputStream, this method will be invoked by the container the first time when it is possible to read data. Subsequently the container will invoke this method if and only if ServletInputStream.isReady() method has been called and has returned false.
Throws:
IOException - if an I/O related error has occurred during processing

- @Override
```
public void setReadListener( ReadListener readListener )
{
this.readListener = readListener;
try
{
readListener.onDataAvailable();
}
catch ( IOException e )
{
}
}
}
```

3) onAllDataRead() – void onDataAvailable() throws IOException

- When an instance of the ReadListener is registered with a ServletInputStream, this method will be invoked by the container the first time when it is possible to read data. Subsequently the container will invoke this method if and only if ServletInputStream.isReady() method has been called and has returned false.
- Throws:
IOException - if an I/O related error has occurred during processing

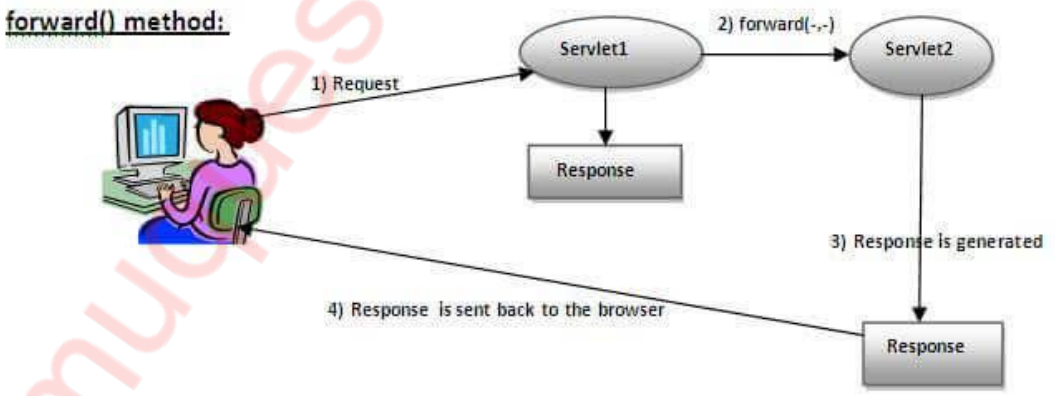
```
@Override
public int read() throws IOException
{
 if ( bytes.length > position )
 {
   return (int) bytes[position++];
 }
```

```
 if ( readListener != null )
 {
  readListener.onAllDataRead();
 }
 return -1;
}
```

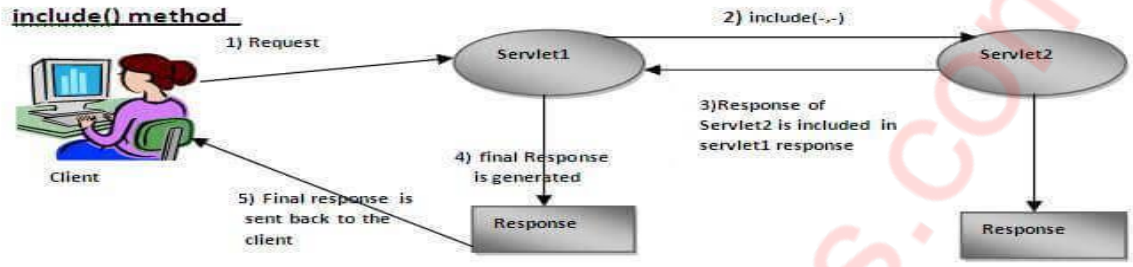
f. What is RequestDispatcher?Explain the two methods from Request Dispatcher. (5)

- The Request Dispatcher interface dispatching the request to another resource it may be html, servlet or jsp.
- This interface also be used to include the content of another resource. It is one of the way of servlet collaboration.
- There are two methods defined in the RequestDispatcher interface. public void forward(ServletRequest request, ServletResponse response)throws ServletException, java. io.

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



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



3. **Attempt any three of the following:**
   a. **Explain the reasons to use JSP.** **(5)**

1) **Compilation** - Java server pages JSP are always compiled before its processed by the server as it reduces the effort of the server to create process.
2) **Fast Development –** No need to recompile and redeploy. If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
3) **Less code than Servlet –** In JSP, we can use many tags such as action tags, JSTL, custom tags, etc, that reduces the code. Moreover, we can use EL, implicit objects, etc.
4) **Easy to maintain –** The business logic with presentation logic is separated in JSP so we can easily managed web application.
   In Servlet technology, we mix our business logic with the presentation logic.
5) **Extension to Servlet** – Jsp technology is the extension to Servlet Technology. We can use all the features of the Servlet in JSP.
   In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP that makes JSP development easy.

b. What are directives? Explain page directive with any of its four attributes. (5)

- A JSP directive affects the overall structure of the servlet class. It usually has the following form −
  <%@ directive attribute = "value" %>
- Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.
- The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.
- There are three types of directive tag –

S.No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

- **JSP - The page Directive**
- The page directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

  Following is the basic syntax of the page directive −

  <%@ page attribute = "value" %>

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.page attribute = "value" />

Attributes

Following table lists out the attributes associated with the page directive –

S.No.	Attribute & Purpose
1	**buffer** Specifies a buffering model for the output stream.
2	**errorPage** Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
3	**isErrorPage** Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
4	**extends** Specifies a superclass that the generated servlet must extend.
5	**import** Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.

c. Develop a simple JSP application to accept values from html page and display on next page. (Name-txt, age-txt, hobbies-checkbox, email-txt, gender-radio button). (5)

- **HTML Form**

```
<HTML>
<form action="Show.jsp">
<br> Enter Name <input type=text name=txtName>
<br> Enter Age <input type=text name=txtAge>
<br> Select Hobbies <input type=checkbox name=txtHob value=Reading> Reading
<input type=checkbox name=txtHob value=Singing> Singing
<br> Select Gender <input type=radio name=txtGender value=Male> Male
<input type=radio name=txtGender value=Female> Female
<input type=radio name=txtGender value=Other> Other
<input type=submit>
</form>
</HTML>
```

- **JSP Code**

```
Your Name<%=request.getParameter("txtName")%>
Your Age<%=request.getParameter("txtAge")%>
<%
Foreach(i in request.getParameter("txtHob")) out.println("<br>"+i);
%>
Gender Selected<%=request.getParameter("txtGender")%>
```

d. Explain the < jsp:useBean> tag with its attribute. Support your answer with suitable code snippet. (5)

- The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.
- Syntax of jsp:useBean action tag

- <jsp:useBean id= "instanceName" scope= "page | request | session | applic ation"
- **class**= "packageName.className" type= "packageName.className"
- beanName="packageName.className | <%= expression >" >

  </jsp:useBean>

- **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.

- **Calculator.java (a simple Bean class)**

```
package com.javatpoint;
public class Calculator{

public int cube(int n){return n*n*n;}
}
```

- **index.jsp file**

```
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

e. List the name of JSP implicit objects. Explain any Two in details. (5)

1. The implicit out objects.
2. The implicit request Object.
3. The implicit response object.
4. The implicit Session Object.
5. The implicit Config Object.

1) The implicit out Objects

- In a scriplet <%...%> you can use the out object to write to the output stream.
- **Methods**

1. Public abstract void print(Object obj) throws IOException

- Print an object. The string produced by the String.valueOf(Object) method is written to the JspWriter's buffer or, if no buffer is used, directly to the underlying writer.
- **Parameters**

  Obj – The Object to be printed.
- **Throws**

  IOException – If an error occurred while writing.

2. Public abstract void clear() throws IOException

- Clear the contents of the buffer. If the buffer has been already been flushed then the clear operation shall throw an IOException to signal the fact that some data has already been irrevocably written to the client response stream.
- **Throws**

  IOException – If an I/O error occurs

3. Public abstract void newLine() throws IOException

- Write a line separator. The line separator string is defined by the system property line. Separator, and is not necessarily a single newline('\n') character.

- **Throws**

  IOException – If an I/O error occurs
- **Example**

  <%

  Out.print("The sum is"); out.print("1+2="+(1+2));

  %>

2) The implicit response object

- Following are some methods associated with response object.

1. Public void addCookie(Cookie cookie)

Adds the specified cookie to the response. This method can be called multiple times to set more than one cookie.

- **Parameters**

  Cookie – the Cookie to return to the client.

2. Public String getContentType()

Returns the content type used for the MIME body sent in this response.

3. Public String getHeader(String name)

Gets the value of the response header with the given name.

- **Parameters**

  Name – the name of the response header whose value to return
- **Returns**

  The value of the response header with the given name, or null if no header with the given name has been set on this response.

f. What is wrong in using JSP scriptlet tag? How JSTL fixes JSP scriptlet shortcomings? (5)

➢ **What is wrong in using JSP scriptlet tag?**

- Scriplets are nothing but java code enclosed within <% and %> /.
  So the problems are:
- Hard to read.
- Hard to maintain.
- Discourage reuse and encapsulation.
- Encourage to put complex login in pages.

➢ **How JSTL fixes JSP scriptlet shortcomings?**

- JSTL stands for JavaServer Pages Standard Tag Lirary. JSTL encapsulates core functionality of JSP application.
- JSTL has support for common, structural tasks such as conditionals and iterations, tags internationalization, tags for manipulating XML documents,

and SQL, tags. It is also provides a framework for integrating existing custom tags with JSTL tags.

4. Attempt any three of the following:

a. Explain benefits of EJB. (5)

- EJBs are reusable components
  - Can be reused in different parts of the system.
  - Can be packaged into libraries and sold.
- EJBs can be combined visually using development IDEs.
  E.g. Visual Age, Visual Café
- EJBs provides convenient abstractions so it do not require you to write:
  - Multi-threaded, multiple access code.
  - Database access code (e.g. JDBC).
  - Network communication code(i.e. it uses RMI) for client/server communication.
  - Network communication code for EJB to EJB communication.
  - Transaction management code.
- EJBs from different businesses can interact easily.
  - This is because of their well-defined interfaces.

b. Write a note on different types of session beans. (5)

- **Types of Session Beans**
- Session beans are of three types: stateful, stateless, and singleton.

- **Stateful Session Beans**
- The state of an object consists of the values of its instance variables. In a *stateful session bean*, the instance variables represent the state of a unique client/bean session. Because the client interacts ("talks") with its bean, this state is often called the *conversational state*.
- As its name suggests, a session bean is similar to an interactive session. A session bean is not shared; it can have only one client, in the same way that

an interactive session can have only one user. When the client terminates, its session bean appears to terminate and is no longer associated with the client.

- The state is retained for the duration of the client/bean session. If the client removes the bean, the session ends and the state disappears. This transient nature of the state is not a problem, however, because when the conversation between the client and the bean ends, there is no need to retain the state.

- **Stateless Session Beans**

- A *stateless session bean* does not maintain a conversational state with the client. When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation. When the method is finished, the client-specific state should not be retained. Clients may, however, change the state of instance variables in pooled stateless beans, and this state is held over to the next invocation of the pooled stateless bean. Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client. That is, the state of a stateless session bean should apply across all clients.
- Because they can support multiple clients, stateless session beans can offer better scalability for applications that require large numbers of clients. Typically, an application requires fewer stateless session beans than stateful session beans to support the same number of clients.
- A stateless session bean can implement a web service, but a stateful session bean cannot.

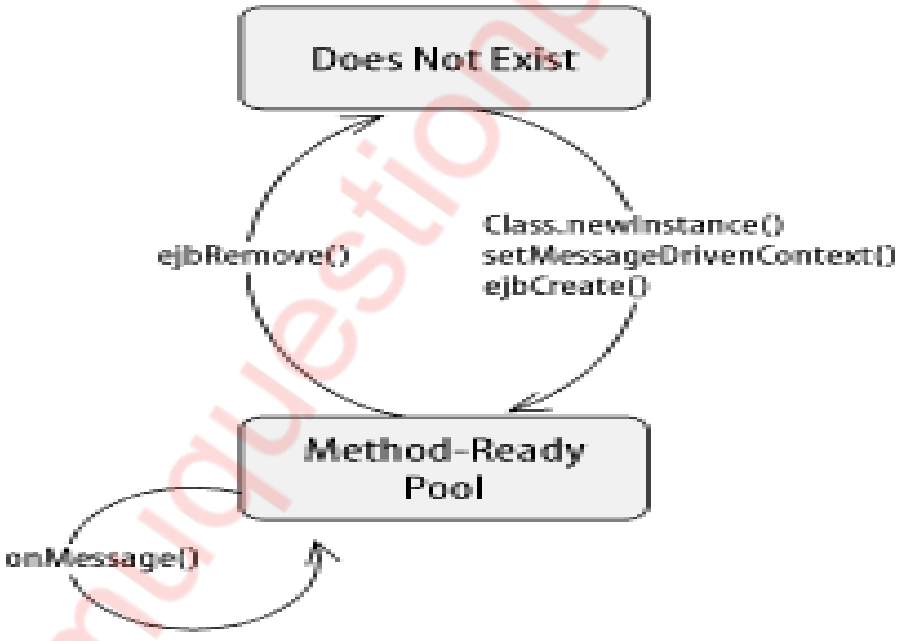
- **Singleton Session Beans**

- A *singleton session bean* is instantiated once per application and exists for the lifecycle of the application. Singleton session beans are designed for circumstances in which a single enterprise bean instance is shared across and concurrently accessed by clients.
- Singleton session beans offer similar functionality to stateless session beans but differ from them in that there is only one singleton session bean per application, as opposed to a pool of stateless session beans, any of which may respond to a client request. Like stateless session beans, singleton session beans can implement web service endpoints.

- Singleton session beans maintain their state between client invocations but are not required to maintain their state across server crashes or shutdowns.
- Applications that use a singleton session bean may specify that the singleton should be instantiated upon application startup, which allows the singleton to perform initialization tasks for the application. The singleton may perform cleanup tasks on application shutdown as well, because the singleton will operate throughout the lifecycle of the application.

c. Explain life cycle of a message driven bean using suitable diagram. (5)

- The Life Cycle of a Message-Driven Bean
- Just as the entity and session beans have well-defined life cycles, so does the MDB bean. The MDB instance's life cycle has two states*:* Does Not Exist *and* Method-Ready Pool *.* The Method-Ready Pool is similar to the instance pool used for stateless session beans. Figure illustrates the states and transitions that an MDB instance goes through in its lifetime.



MDB life cycle

- **Does Not Exist**
- When an MDB instance is in the Does Not Exist state, it is not an instance in the memory of the system. In other words, it has not been instantiated yet.
- **The Method-Ready Pool**
- MDB instances enter the Method-Ready Pool as the container needs them. When the EJB server is first started, it may create a number of MDB instances and enter them into the Method-Ready Pool. (The actual behavior of the server depends on the implementation.) When the number of MDB instances handling incoming messages is insufficient, more can be created and added to the pool.
- **Transitioning to the Method-Ready Pool**
- When an instance transitions from the Does Not Exist state to the Method-Ready Pool, three operations are performed on it. First, the bean instance is instantiated when the container invokes the Class.newInstance( ) method on the MDB class. Second, the setMessageDrivenContext( ) method is invoked by the container providing the MDB instance with a reference to it.

d. Write a stateless session bean code to represent BookInformation. (BookId integer, BookName String, Pages integer, Price double). (5)

- package bookdemo;

```
import javax.ejb.Stateless;

@Stateless

public class BookInfo{

 private int bookId, pages;

private String bookName;

private double price
```

```
public BookInfo(){}

public void setBookId(int i){bookId=i;}

public void setPages(int p){pages=p;}

public void setBookName(String n){bookName=n;}

public void setPrice(double p){price=p;}

public int getBookId(){return bookId;}

public int getPages(){return pages;}

public String getBookName(){return bookName;}

public double getPrice(){return price;}

}
```

e. What is an interceptor? How an interceptor is defined and how aroundInvoke () is added to it? (5)

- Interceptors are used to implement cross-cutting concerns, such as logging, auditing, and security, from the business logic.
- Interceptors are used, as the name suggests, when you want to intercept calls to EJB methods. If you declare an Interceptor for a Bean, every time a method of that Bean is invoked, it will be intercepted with one method of the Interceptor. That means that the execution goes straight to the Interceptor's method. The intercepting method then, can decide whether to call the intercepted EJB method or simply replace it.
- You might find the above behavior resembling the Aspect Oriented Programming philosophy, and you'd be correct. Despite the fact that the implementation of the two technologies is completely different, the truth is they can be used for the same purposes. For example, when you want to log something before of after a Beans method is executed. Or when you want to enforce a specific policy concerning method calls, e.g. authentication, input checking etc. Of course an EJB can have a chain of Interceptors that will intercept the method in a specific order.

- In Java EE 5, Interceptors were allowed only on EJBs. In Java EE 6, Interceptors became a new specification of its own, abstracted at a higher level so that it can be more generically applied to a broader set of specifications in the platform.

- **Interceptor method in bean class**
- Take a look at EmailMDB.java. It contains this method:

```
@AroundInvoke
public Object mdbInterceptor(InvocationContext ctx) throws Exception
{
System.out.println("*** Intercepting call to EmailMDB.mdbInterceptor()");
return ctx.proceed();
}
```

- This method will wrap the call to EmailMDB.onMessage(). The call to ctx.proceed() causes the next object in the chain of interceptors to get invoked. At the end of the chain of interceptors, the actual bean method gets called.

Similarly EmailSystemBean.java has a method annotated with @AroundInvoke

```
@AroundInvoke
public Object myBeanInterceptor(InvocationContext ctx) throws Exception
{
if (ctx.getMethod().getName().equals("emailLostPassword"))
{
System.out.println("*** EmailSystemBean.myBeanInterceptor - username:
" + ctx.getParameters()[0]);
}
return ctx.proceed();
}
```

f. What is Java Naming and Directory Interface? Explain. (5)

- The Java Naming and Directory Interface (JNDI) is an application programminginterface(API)that provides naming and directory functionality to applications written using the Java programming language. It is defined

to be independent of any specific directory service implementation. Thus a variety of directories –new, emerging, and already deployed can be accessed in a common way.

- JNDI stands for Java Naming and Directory Interface. It is a set of API and service interfaces. Java based applications use JNDI for naming and directory services. In context of EJB, there are two terms.
- **Binding** − This refers to assigning a name to an EJB object, which can be used later.
- **Lookup** − This refers to looking up and getting an object of EJB.
- In Jboss, session beans are bound in JNDI in the following format by default.
- **local** − EJB-name/local
- **remote** − EJB-name/remote
- In case, EJB are bundled with <application-name>.ear file, then default format is as following −
- **local** − application-name/ejb-name/local
- **remote** − application-name/ejb-name/remote

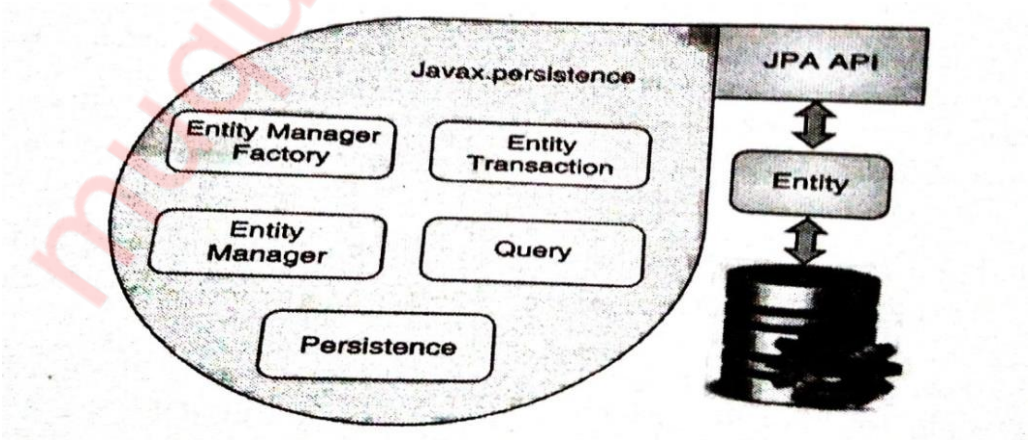
5. Attempt any three of the following:

a. What is Persistence? (5)

- Persistance is one of the fundamental concepts of application development. It allows DATA to outlive the execution of an application that created it. It is one of the most vital place of an application without which all the data is simply lost.
- Mapping Java objects to database tables and vice versa is called *Object*-relational mapping (ORM). The Java Persistence API (JPA) is one possible approach to ORM. Via JPA the developer can map, store, update and retrieve data from relational databases to Java objects and vice versa. JPA can be used in Java-EE and Java-SE applications.

- JPA is a specification and several implementations are available. Popular implementations are Hibernate, EclipseLink and Apache OpenJPA. The reference implementation of JPA is EclipseLink.
- JPA permits the developer to work directly with objects rather than with SQL statements. The JPA implementation is typically called persistence provider.
- The mapping between Java objects and database tables is defined via persistence metadata. The JPA provider will use the persistence metadata information to perform the correct database operations.
- JPA metadata is typically defined via annotations in the Java class. Alternatively, the metadata can be defined via XML or a combination of both. A XML configuration overwrites the annotations.
- The following description is based on the usage of annotations.
- JPA defines a SQL-like Query language for static and dynamic queries.
- Most JPA persistence providers offer the option to create the database schema automatically based on the metadata.
- As a specification, the Java Persistence API is concerned with *persistence*, which loosely means any mechanism by which Java objects outlive the application process that created them. Not all Java objects need to be persisted, but most applications persist key business objects. The JPA specification lets you define which objects should be persisted, and how those objects should be persisted in your Java applications.

b. Explain using suitable diagram architecture of Java Persistence API. (5)



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

Units	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	This interface is implemented by each JPA vendor to obtain relational objects that meet the criteria.

The above classes and interfaces are used for storing entities into a database as a record. They help programmers by reducing their efforts to write codes for storing data into a database so that they can concentrate on more important activities such as writing codes for mapping the classes with database tables.

c. Write a JSP code to add guest feedback using JPA in GuestBook table in database. (Make suitable assumptions) (5)

– AddFeedBack.jsp

```
<%@page import="java.util.*,javax.persistence.*,mypack.GuestBook" %>

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<%!

private EntityManagerFactory entityManagerFactory;

private EntityManager entityManager;

private EntityTransaction entityTransaction;

%>

<%

entityManagerFactory =
Persistence.createEntityManagerFactory("JPAApplication1PU");

entityManager = entityManagerFactory.createEntityManager();

String submit = request.getParameter("btnSubmit");

try {

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

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

String messageDate = new java.util.Date().toString();

GuestBook gb = new GuestBook();

gb.setVisitorName(guest);
```

```
gb.setMessage(message);

gb.setMessageDate(messageDate);

entityTransaction = entityManager.getTransaction();

entityTransaction.begin();

entityManager.persist(gb);

entityTransaction.commit();

} catch (RuntimeException e) {

if(entityTransaction != null) entityTransaction.rollback();

throw e; }

try {

guestbook = entityManager.createQuery("SELECT g from GuestBook

g").getResultList();

        } catch (RuntimeException e) { }

entityManager.close();%>
```

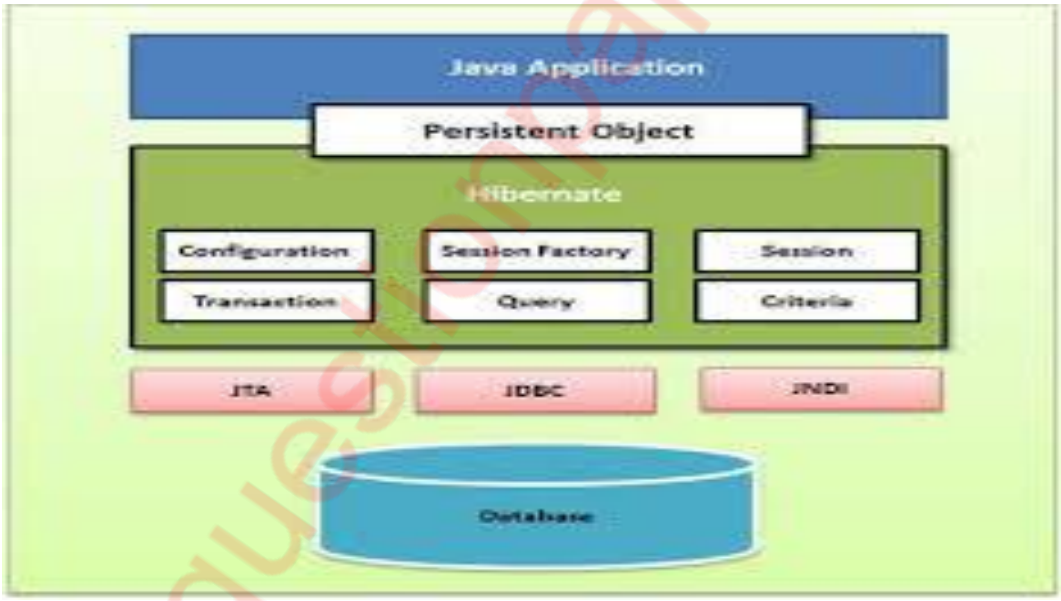
d. What is Hibernate? (5)

- Hibernate is an Object-Relational Mapping(ORM) solution for JAVA. It is an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java application.
- 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.
  - Enterprise JavaBeans Entity Beans
  - Java Data Objects
  - Castor
  - TopLink

- Spring DAO
- Hibernate, and many more

- Hibernates takes care of mapping Java classes to database tables using XML, files and without any line of code. If there is a change in the database or in any tables, then all that you need to change are the XML file properties.
- Provides simple APIs(classes and methods) for storing and retrieving Java objects directly to and from the database.
- Hibernate supports Inheritence, Association relations, and Collections.
- Abstract away the unfamiliar SQL types and provides a way to work around familiar Java Objects.
- Hibernate does not require an application server to operate.

e. Explain using suitable diagram architecture of Hibernate. (5)



- Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.
- Following section gives brief description of each of the class objects involved in Hibernate Application Architecture.

- **Configuration Object**
- The Configuration object is the first Hibernate object you create in any Hibernate application. It is usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate.
- The Configuration object provides two keys components −
- **Database Connection** – This is handled through one or more configuration file supported by Hibernate. These files are hibernate.properties and hibernate.cfg.xml.
- **Class Mapping Setup** − This component creates the connection between the Java classes and database tables.
- **SessionFactory Object**
- Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.
- The SessionFactory is a heavyweight object; it is usually created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So, if you are using multiple databases, then you would have to create multiple SessionFactory objects.
- **Session Object**
- A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.
- The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.
- **Transaction Object**
- A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).
- This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

- **Query Object**
- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.
- **Criteria Object**
- Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

f. Write a JSP code to add visitor's feedback using Hibernate in FeedBack table in database. (Make suitable assumptions). (5)

- <%@page import="org.hibernate.*, org.hibernate.cfg.*, mypack.*" %>

```
<%! SessionFactory sf;
org.hibernate.Session hibSession; %>
<%
sf = new Configuration().configure().buildSessionFactory();
hibSession = sf.openSession();
Transaction tx = null;
GuestBookBean gb = new GuestBookBean();
try{
tx = hibSession.beginTransaction();
String username = request.getParameter("name");
String usermsg = request.getParameter("message");
String nowtime = ""+new java.util.Date();
gb.setVisitorName(username);
gb.setMsg(usermsg);
gb.setMsgDate(nowtime);
hibSession.save(gb);
tx.commit();
out.println("Thank You for your valuable feedback....");
}catch(Exception e){out.println(e);}
hibSession.close();%>
```