BSC-IT

SEM -5 Enterprise Java

APR-2019 **Q.P.Code: 57842**

Q1 Attempt any three of the following: (15)

a) What are the challenges of Enterprise Application Development? (5)

- The major challenge is to maintain the value of existing business with the Internet's hyper-competitive pace.
- To achieve a competitive edge, time and Quality plays an imp. Role.
- To maintain the value of effective business, time and quality plays an important role.
- For quick and customized enterprise application development, following factors should be considered by application developer:

Application should

- Programming productivity
- respond to demand
- Integrate with existing system
- Freedom to choose
- Maintain security
- provide a standard environment to develop Enterprise Application

b) List and explain Java Container types. (5)

The Java EE server provides services to Java EE components in the form of a container.

- The EJB Container: Java EE server components that are coded to hold business logic are called Enterprise JavaBeans[EJB]. The Java EE server's EJB container provides local and remote access to EJB.
  The EJB container:
  • Creates new instance of EJBs
  • Manages their life cycle
  • Provides services such as:
    • Transaction
    • Security
    • Concurrency
    • Distribution
    • Naming service
    • The possibility to be involved asynchronously
- The Web Container: A Java EE container that hosts web applications is a web container. It is responsible for instantiating and invoking servlets and supporting the HTTP and HTTPS protocols. It is the container used to feed web pages to client browsers.

- The Application Client Container: It provides services required for the execution of application client components. Application clients and their container run on the client. It includes a set of Java classes and libraries, security management, naming services to Java SE applications etc.
- Applet container: Applet container is provided by the majority of the web browsers for the execution of Applet. The container prevents any code downloaded to the local computer from accessing local system resources such as processes or files.

c) What are the alternatives to CGI? Explain in detail. (5)

CGI programs provided a relatively simple way to create Web applications that accepts user input, queries a database and returns relevant results back to the Web browser. But there are certain disadvantages of using CGI like Lack of scalability and reduced speed Platform dependent, A process consumes a lot of server side resources in multi-user situation, Difficult for beginners to program modules in this environment, Less efficient etc.

The alternatives to CGI is Servlets. Servlets

- Are loaded into memory once and run from memory thereafter
- Are created as a thread, not as a process
- Are powerful object-oriented abstraction of HTTP
- Are portable across multiple platforms
- Are simple to design and implement
- Accept client's requests and efficiently run the requests within the secure and reliable scope of a Java Virtual Machine
- Are robust and scalable CGI Replacement
- Provide direct Database access using different Database drivers.
- Are supported by several Web Servers.

d) Write a short note on Part and WebConnection interfaces. (5)

Part interface:

It represents a part or form item that was received within a multipart/formdata POST request. Its frequently used methods are:

InputStream getInputStream()throws IOException

Gets the content of this part as an InputStream

String getSubmittedFileName() Gets the file name specified by the client

WebConnection interface:

This interface encapsulates the connection for an upgrade request. It allows the protocol handler to send service requests and status queries to the container. It has following two methods:

ServletInputStream getInputStream() throws IOException Returns an input stream for this web connection for reading binary data.

ServletOutputStream getOutputStream()throws IOException Returns an output stream for this web connection for writing binary data.

e) Explain Deployment descriptor file with its elements. (5)

The web.xml file is a standard Java EE deployment descriptor, specified in the Java Servlet specification
It can be changed without making any change in the source code
It is used to run the application
The elements of web.xml deployment descriptor file are:

I. <web-app>
II. <servlet>
III. <session-mapping>
IV. <session-config>
V. <welcome-file-list>

1. <web-app>
   All entries are wrapped within a pair of opening and closing <web-app> elements

2. <servlet>
   Used to register and configure Servlet details within the opening and closing of <servlet> elements
   <servlet-name> - defines the name of a servlet
   <servlet-class> - actual servlet class name

3. <session-mapping>
   It is used to define mapping between servlet and the corresponding url pattern
   <servlet-name> - name of the servlet
   <url-pattern> - corresponding url details

4. <session-config>
   Used to specify session information
   <session-timeout> - used to define session timeout in minutes

5. <welcome-file-list>
   It consists of a set of filenames when user accesses the path of a WAR subdirectory. It consists of:
   <welcome-file> - it consists of either index.html or index.jsp

f) Explain RowSet and its type in JDBC. (5)

A JDBC RowSet object holds tabular data in a way that makes it more flexible and easier to use than a result set.

Oracle has defined five RowSet interfaces for some of the more popular uses of a RowSet, and standard reference are available for these RowSet interfaces.
These versions of the RowSet interface and their implementations have been provided as a convenience for programmers. Programmers are free to write their own versions of the javax.sql.RowSet interface, to extend the implementations of the five RowSet interfaces, or to write their own implementations. The following interfaces extend RowSet interface:

• JdbcRowSet: It is a connected rowset that makes a JDBC driver look like a JavaBeans component.
• CachedRowSet: it is a container for rows of data that caches its rows in memory, which makes it possible to operate without always being connected to its data source.
• WebRowSet: It is a disconnected rowset that caches its data in memory in the same manner as a CachedRowSet object. As a consequence, a WebRowSet object fills a real need by making it easy for web services to send and receive data from a database in the form of an XML document.
• JoinRowSet: It provides a mechanism for combining related data from different RowSet objects into one JoinRowSet object, which represents an SQL JOIN . In other words, a JoinRowSet object acts as a container for the data from RowSet objects that form an SQL JOIN relationship.
• FilteredRowSet: It provides the reference implementation which may be extended if required. Alternatively, a vendor is free to implement its own version by implementing this interface.

Q2 Attempt any three of the following: (15)

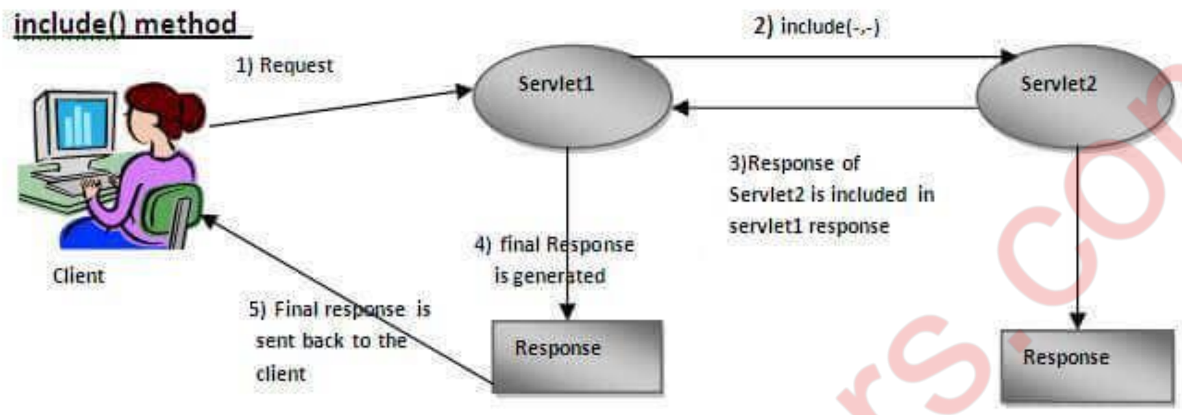
a) Explain methods of RequstDispatcher interface. (5)

RequestDispatcher interface :

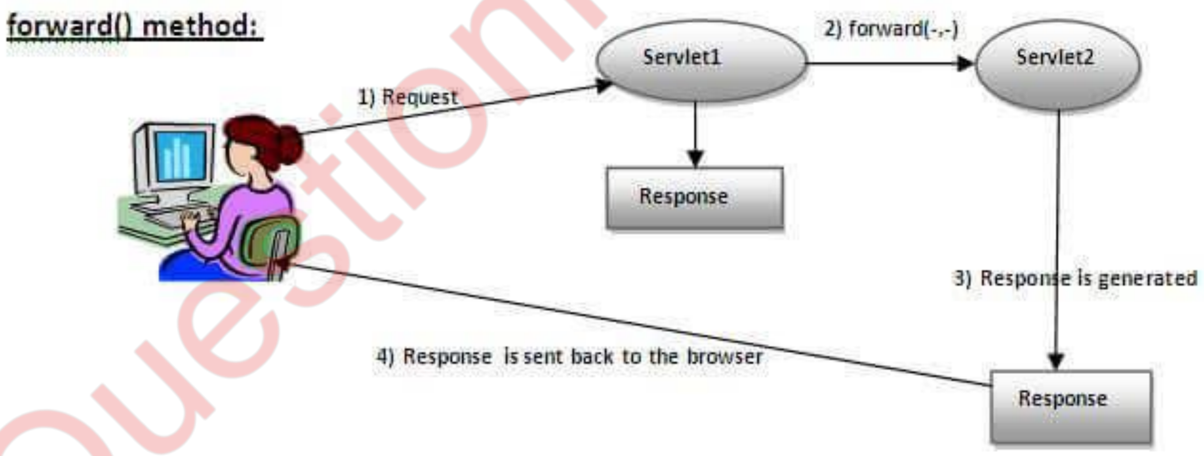
- It is used for dispatching a request.
- It encapsulates a reference to a web resource located at the path specified within the scope of the same servlet context.
- It defines an object, which receives requests from a visitor's browser and sends the requests to any other resource on the web server.
- The servlet engine creates the RequestDispatcher object, which is used as a wrapper around a web server resource located through the path specified or given by a particular name.
- We can get the object of RequestDispatcher interface by using the method: getRequestDispatcher(), there are two ways of using this method:
- javax.servlet.ServletContext.getRequestDispatcher()
- javax.servlet.ServletRequest.getRequestDispatcher()

Methods of RequestDispatcher :

- This interface has two methods:
- include() : it allows including the content produced by another resource such as servlet, JSP or HTML file in the calling servlet's response.


include() method
1) Request
2) include(-,-)
Servlet1
Servlet2
3)Response of Servlet2 is included in servlet1 response
4) final Response is generated
5) Final response is sent back to the client
Client
Response
Response

- forward() : it allows forwarding the request to another servlet, a JSP or HTML page on the server. This resource then take over the responsibility for producing the response.
- It is useful when one servlet does some preliminary processing of a request and wants to let another object complete the generation of the response.



b) Explain setting, sending and reading of Cookie in Java servlet. (5)

Cookie can be created using the following contructor:

Cookie(String name, String value): Constructs a cookie with the specified name and value.

A cookie is just a name-value attribute that is issued by the server and stored on a client machine. It is returned by the client whenever it is accessing a certain group of URLs on a specified server. Following methods are some methods belongs to Cookie class.

Cookies are created, stored and fetched using following methods:

- Cookie.setMaxAge(int expiry): sets the maximum age in seconds for this Cookie.
- HttpServletResponse.addCookie(): Adds the specified cookie to the response. This method can be called multiple times to set more than one cookie.
- HttpServletRequest.getCookies(): Returns an array containing all of the Cookie objects the client sent with this request

c) What is session tracking? What are the ways to track the sessions? (5)

HTTP is a stateless protocol that provides no way for a server to recognize that a sequence of request come from the same client. (not even IP Address). Following are various techniques for session tracking:
1) HTTPSession
2) Cookies
3) URL rewriting
4) Hidden form fields

1) HttpSession Interface :

HttpSession Interface provides a way across more than one-page request or visit to a website and to store information about that user.
With the help of the HttpSession interface, a session can be created between an HTTP client and an HTTP server. This session can persist for a specified time period, across more than one connection or page request from the user.
A Servlet obtains an HttpSession object using
HttpSession getSession(boolean) of HttpservetRequest.

2) Providing Continuity with Cookies :

cookie is just a name-value attribute that is issued by the server and stored on a client machine. It is returned by the client whenever it is accessing a certain group of URLs on a specified server. Following methods are some methods belongs to Cookie class.
A cookie is a small piece of data stored on the client-side which servers use when communicating with clients. Cookies are used to identify a client when sending a subsequent request. They can also be used for passing some data from one servlet to another.

3) Hidden Form Fields :

A web server can send a hidden HTML form field along with a unique session ID as follows:
<input type="hidden" name="city" value="Mumbai">
This could be an effective way of keeping track of the session but clicking on a regular (<A HREF…>) hypertext link does not result in a form submission.

4) **URL Rewriting :**
There is an alternative way to send a cookie: URL Rewriting. The cookie data is appended to the URL. You should use URL rewriting only as a fallback for browsers that do not support cookies.

d) Write a short note on creation and validation of sessions. (5)

The following program explains how the session is created. It also explains that till whether the session is created for the first time or not. Also if it is used 10 times then it deletes the session object using its invalidate() method.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class CalculationVisitServlet extends HttpServlet
{
 // private static int counter;
 private int counter;
 public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 HttpSession session=request.getSession(true);
 if(session.isNew())
 {
 out.print("This is the first time you are visiting this page");
 ++counter;
 }
 else
 {
 synchronized(this)
 {
 if(counter==10)
 {
 session.invalidate();
 counter=0;
 request.getSession(false);
 }
 else
 out.print("You have visited this page "+(++counter)+ " times");
 }
 }
 }
 }
```

e) Which points are important while uploading a file? (5)

Following points should be considered while uploading a file:

Set encodiging type attribute as follows:

enctype attribute of <form> tag

multipart/form-data - No characters are encoded.

text/plain- Spaces are converted to "+" symbols, but no special characters are encoded

type attribute value in <input>tag

type="file"

let the user choose one or more files from their device storage. Once chosen, the files can be uploaded to a server using form submission

Part getPart()

Retrieve the name, store location, size, and content-type of the file

File name=--------, StoreLocation=--------, size=--------bytes, isFormField=false, FieldName=filedetails

File getSubmittedFileName()

It returns the selected file name from the FileOpenWindow.

@MultipartConfig

We need to annotate File Upload handler servlet with MultipartConfig annotation to handle

multipart/form-data requests that is used for uploading file to server. MultipartConfig annotation has following attributes:

• **fileSizeThreshold:** We can specify the size threshold after which the file will be written to disk. The size value is in bytes, so 1024*1024*10 is 10 MB.

• **location:** Directory where files will be stored by default, it's default value is "".

• **maxFileSize:** Maximum size allowed to upload a file, it's value is provided in bytes. It's default value is -1L means unlimited.

• **maxRequestSize:** Maximum size allowed for multipart/form-data request. Default value is -1L that means unlimited.

Steps to write a program to upload a file

• Allow browsing the file to upload it

• Allows specifying the location where to upload it

• Display the message indicating a successful file upload

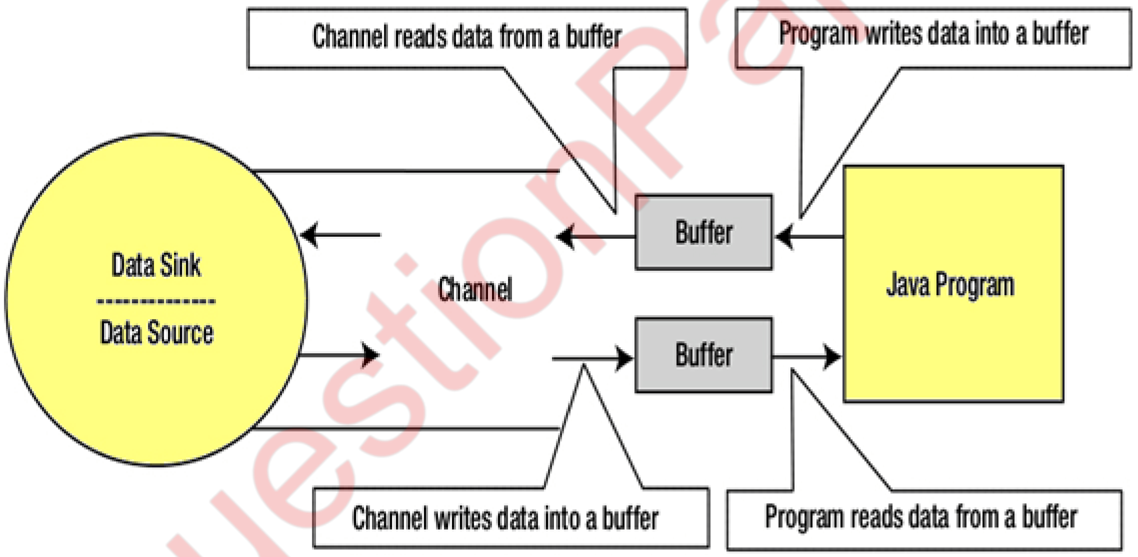
• Display the link to return back to upload more files

f) Explain the working of Non-Blocking I/O. **(5)**

Non-blocking I/O:

- It is Buffer-oriented
- Channels are available for Non-blocking I/O operation
- Selectors are available for Non-blocking I/O operation
- Non blocking IO does not wait for the data to be read or write before returning.
- Java NIO non- blocking mode allows the thread to request writing data to a channel, but not wait for it to be fully written. The thread is allowed to go on and do something else in a mean time.
- java NIO is buffer oriented I/O approach. Data is read into a buffer from which it is further processed using a channel. In NIO we deal with the channel and buffer for I/O operation.
- The major difference between a channel and a stream is:
  - A stream can be used for one-way data transfer.
  - A channel provides a two-way data transfer facility.

Therefore with the introduction of channel in java NIO, the non-blocking I/O operation can be performed. Let's see the interaction between channel, buffers, java program, data source and data sink:


Channel reads data from a buffer
Program writes data into a buffer
Data Sink
Data Source
Channel
Buffer
Buffer
Java Program
Channel writes data into a buffer
Program reads data from a buffer

Channels:

In Java NIO, the channel is a medium that transports the data efficiently between the entity and byte buffers. It reads the data from an entity and places it inside buffer blocks for consumption. Channels act as gateway provided by java NIO to access the I/O mechanism. Usually channels have one-to-one relationship with operating system file descriptor for providing the platform independence operational feature.

Q3 Attempt any three of the following: (15)

a) How JSP functions and executes? (5)

The following steps explain how the web server creates the Webpage using JSP –

1. As with a normal page, your browser sends an HTTP request to the web server.

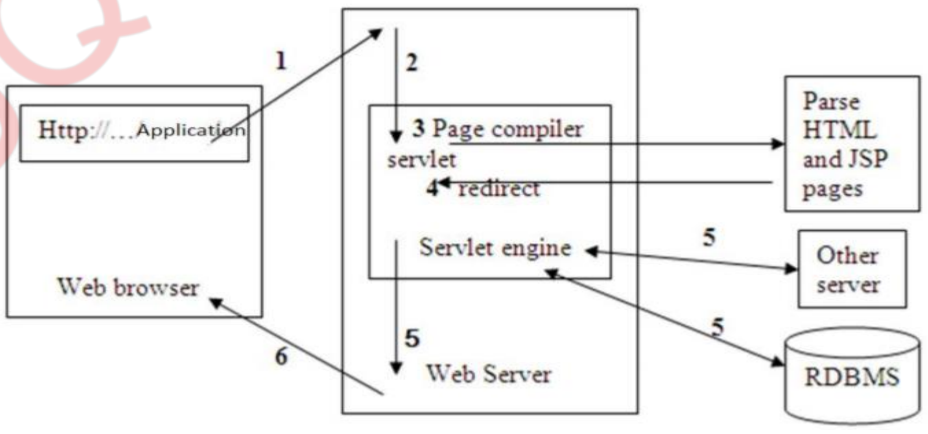
2. The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.

3. The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.

4. The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.

5. A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is further passed on to the web server by the servlet engine inside an HTTP response.

6. The web server forwards the HTTP response to your browser in terms of static HTML content. Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.



b) How to set and access the properties of Java bean in JSP? (5)

The <jsp: useBean> element is used to access the bean in jsp. This element instantiates an object of the bean class specified by class name & binds it to an object with the name specified.
A new object is instantiated only if only if there is no existing one with the same ID & scope.
The useBean action creates a Java object in a specified scope. The object is also made available in the current JSP page as a scripting variable. The syntax for the useBean action is:

```
<jsp:useBean id= "beanobjname"
 class="className " | beanName="className "
 scope="page | request | session | application" >
</jsp:useBean>
```

The setProperty action can be used in conjunction with the useBean action to set the properties of a bean. We can even get the container to populate the bean properties from the request parameters without specifying the property names. In such cases the container will match the bean property names with the request parameter names. The syntax for the setProperty action is shown below:

```
<jsp:setProperty name="BeanObj"
property="*" |
 property="propertyName" |
 [param="paramName" value="value"] />
```

The getProperty action can be used in conjunction with the useBean action to get the value of the properties of the bean defined by the useBean action. For example:

```
<jsp:getProperty name="BeanObj" property="firstName"/>
```

c) What are the types of expressions in EL? (5)

The **Expression Language** (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc.
There are many implicit objects, operators and reserve words in EL.
It can be used for the following cases:
To fetch the parameter from another jsp file as: ${ param.name }
To perform arithmetic operators: e.g. 5*5+4: ${5*5+4}
To perform logical operator as: true and true: ${true and true}
To perform relational operator as: 3.2>=2: ${3.2>=2}
To perform conditional operator as: The result of 10>2 is: "${(10>1)?'greater':'lesser'}"
To perform empty operator as: The Value for the Empty operator is:: ${empty ""}<br>

Types of Expressions

- The unified EL provides 2 types of expressions:
  - Value Expression
  - Method Expression
  - **Value Expression:**
    It is further categorized as:
  - Rvalue expressions: can only read data, but cannot write data. Expressions that use immediate evaluation syntax are always rvalue expressions.
  - Lvalue expressions: can read and write data. Expressions that use deferred evaluation syntax can act as both rvalue and lvalue expressions.
  - **Method Expressions:**
  - A JSF component uses method expressions, Which in turn invokes methods that do some processing on behalf of the component element.
  - For std. components, these methods are necessary for handling events that the components generates as well as validating component data.

  - Using EL Expressions:

  It can be used in 2 situations:
  As attribute values in std. and custom actions.
  In template text such as HTML or non-JSP elements, in the JSP file.

d) Write a short note on restriction, projection and partitioning operators in EL. (5)

Restriction operator – where:
Iterates over the source sequence and yields those elements for which the predicate function returns true. The predicate function accepts two arguments:
The first one represents the element to test
The second, if present, represents the zero-based index of the element within the source sequence.
e.g. employees.where(e->e.salary >=10000)
projection operator - select:
Iterates over the source sequence and yields the results of evaluating the selector Lambda for each element. The selector accepts two arguments:
The first one represents the element to process
The second, if present, represents the zero-based index of the element within the source sequence.
E.g. employees.select(e->e.EmployeeName)
partitioning operator - take:
Iterates over the source sequence and yields a given number of elements from a sequence and
then skips the remainder of the sequence.
e.g. employees.take(100)

partitioning operator - skip:
Iterates over the source sequence and skips a given number of elements from a sequence and then yields the remainder of the sequence.
e.g. employees.skip(2)

e) Explain conditional or flow control statements in XML Tag Library with example. (5)

The <x:if> tag is used for evaluating the test XPath expression. It is a simple conditional tag which is used for evaluating its body if the supplied condition is true.

The syntax used for including the <x:if> tag is:
 <x:if attributes> body content </x:if>

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head>
 <title>x:if Tags</title>
</head>
<body>
<h2>Vegetable Information:</h2>
<c:set var="vegetables">
<vegetables>
 <vegetable>
 <name>onion</name>
 <price>40</price>
</vegetable>
<vegetable>
 <name>Potato</name>
 <price>30</price>
 </vegetable>
</vegetables>
</c:set>
<x:parse xml="${vegetables}" var="output"/>
<x:if select="$output/vegetables/vegetable/price < 100">
 Vegetables prices are very low.
</x:if>
</body>
</html>
```

f) How to query and update data in JSTL? (5)

The <sql:query> tag is used for executing the SQL query defined in its sql attribute or the body. It is used to execute an SQL SELECT statement and saves the result in scoped variable.

Example:
```
<sql:query dataSource="${db}" var="rs">
SELECT * from Students;
</sql:query>
```
The <sql:update> tag is used for executing the SQL DML query defined in its sql attribute or
in the tag body. It may be SQL UPDATE, INSERT or DELETE statements.

Example:
```
<sql:update dataSource="${db}" var="count">
INSERT INTO Students VALUES (154,'Nasreen', 'jaha', 25);
</sql:update>
```

Q4 Attempt any three of the following: (15)

a) Write a short note on enterprise bean server and enterprise bean. (5)

enterprise bean server:

- It is a component transaction server.
- It supports the EJB server side component model fro developing and deploying distributed enterprise level applications in a multi-tiered environment.
- It provides:
  - The framework for creating, deploying and managing middle-tier business logic.
  - An environment that allows the execution of applications developed using EJB components.
- In a three-tier environment:
  - The client provides the user interface logic
  - The business rules are separated to the middle tier
  - The database is the information repository
- An EJB server takes care of:
  - Managing and coordinating the allocation of resources to the applications

- Security
- Threads, Connection pooling

enterprise bean:

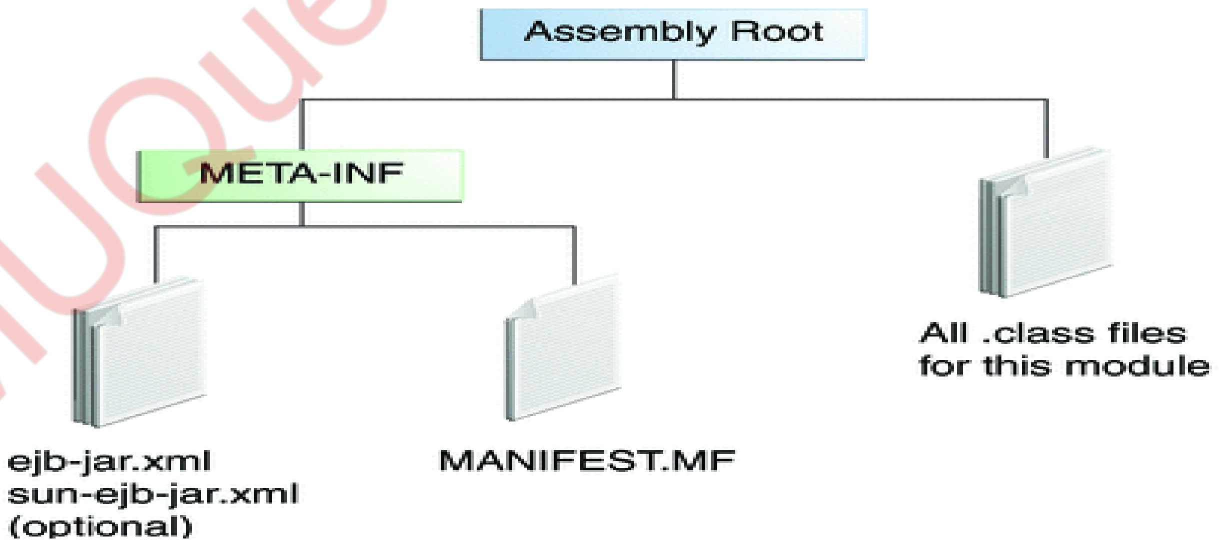
- It is a server-side component that encapsulates the code that fulfills the purpose of the application.
- They can be combined with other components and rapidly produce a custom application.
- The main purpose of introducing EJB was for building distributed components.
- It was designed to solve all the issues n complexities of CORBA.
- This technology is powerful n easy. It helps developers to build business applications that support very large no. of user simultaneously.
- The EJB components can be assembled and reassembled into different distributed applications as per the requirements.

b) Describe packaging of enterprise beans in JAR and WAR modules. (5)

An EJB JAR file is portable and can be used for various applications.

EJB 3.0 required packaging:

- Web application in a WAR file – Web application archive
- EJB's package in a JAR file – java archive
- WAR and JAR files into an EAR file – Enterprise application archive
- To assemble a Java EE application, package one or more modules into an EAR file, the archive file that holds the application. When deploying the EAR file that contains the enterprise bean's EJB JAR file, you also deploy the enterprise bean to GlassFish Server. You can also deploy an EJB JAR that is not contained in an EAR file.
- Figure shows the contents of an EJB JAR file.



Packaging Enterprise Beans in WAR Modules

- Enterprise beans often provide the business logic of a web application. In these cases, packaging the enterprise bean within the web application's WAR module simplifies deployment and application organization.
- To include enterprise bean class files in a WAR module, the class files should be in the WEB-INF/classes directory.
- To include a JAR file that contains enterprise beans in a WAR module, add the JAR to the WEB-INF/lib directory of the WAR module.
- WAR modules that contain enterprise beans do not require an ejb-jar.xml deployment descriptor. If the application uses ejb-jar.xml, it must be located in the WAR module's WEB-INF directory.
- For example, suppose that a web application consists of a shopping cart enterprise bean, a credit card–processing enterprise bean, and a Java servlet front end. The shopping cart bean exposes a local, no-interface view and is defined as follows:

```
package com.example.cart;
@Stateless
public class CartBean { ... }
```

- The credit card–processing bean is packaged within its own JAR file, cc.jar, exposes a local, no-interface view, and is defined as follows:

```
package com.example.cc;
@Stateless
public class CreditCardBean { ... }
```

- The servlet, com.example.web.StoreServlet, handles the web front end and uses both CartBean and CreditCardBean. The WAR module layout for this application is as follows:

```
WEB-INF/classes/com/example/cart/CartBean.class
WEB-INF/classes/com/example/web/StoreServlet
WEB-INF/lib/cc.jar
WEB-INF/ejb-jar.xml
WEB-INF/web.xml
```

c) How to access no-interface view and local business interface ? (5)

- Accessing No-Interface View/Local Business/ Remote Business Interface:
- Client access to an enterprise bean that exposes a local, no-interface view is accomplished either through dependency injection or JNDI lookup.
    - Client access using dependency injection:
        - To obtain a reference to the no-interface view of an enterprise bean through dependency injection, use the javax.ejb.EJB annotation and specify the enterprise bean's implementation class: @EJB
    - Client access using JNDI lookup:

- To obtain a reference to the no-interface view of an enterprise bean through JNDI lookup, use the javax.naming.InitialContext interface's lookup method:

  IntialContext.lookup ();

Method of accessing the enterprise Bean

Remote Clients

A remote client of an enterprise bean has the following traits (characteristics):

- It can run on a different machine and a different Java virtual machine than the enterprise bean it accesses.
- It can be a web component, an application client, or another enterprise bean.
- To a remote client, the location of the enterprise bean is transparent.
- To create an enterprise bean that has remote access, you must code a remote interface and a home interface. The remote interface defines the business methods that are specific to the bean.
- The home interface defines the bean's life-cycle methods

Local Clients

A local client has these characteristics:

- It must run in the same JVM as the enterprise bean it accesses.
- It can be a web component or another enterprise bean.
- To the local client, the location of the enterprise bean it accesses is not transparent.
- To build an enterprise bean that allows local access, you must code the local interface and the local home interface.
- The local interface defines the bean's business methods, and the local home interface defines its life-cycle.

d) Write a short note on lifecycle of a Message Driven Bean with onMessage() method. (5)

Lifecycle of Message Driven Bean:

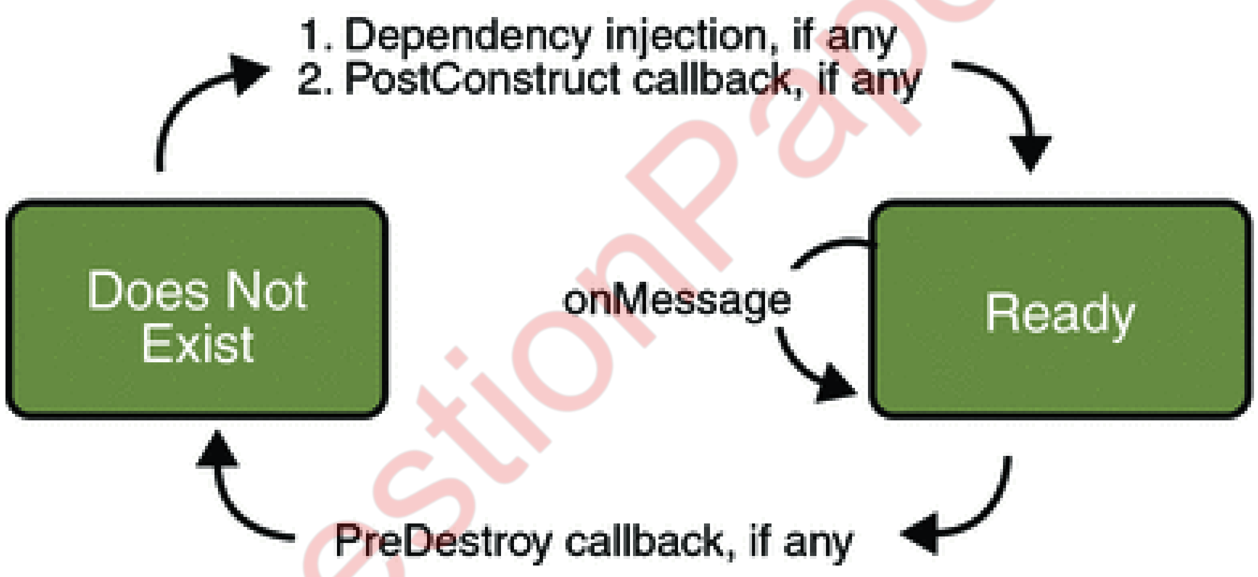
- The EJB container usually creates a pool of MDB instances.
- For each instance, the EJB container performs these tasks:
  - If the MDB uses dependency injection, the container injects these references before instantiating the instance.
  - The container calls the method annotated @PostConstruct(if any).
- Like a stateless session bean, a MDB is never passivated and it has only two states:
- Nonexistent

- Ready to receive messages

Contain business logic for handling received messages to perform the operations such as

  - Computation/calculation
  - Initiating a step in a workflow
  - Storing data
  - Sending another message
  - Transfer data with another EJB for further processing

- Are messages listeners, consuming messages from JMS destination such as queue or a topic
- May invoke other EJB components to get some help


1. Dependency injection, if any
2. PostConstruct callback, if any
Does Not Exist
onMessage
Ready
PreDestroy callback, if any

onMessage() Method

- In MDB, metadata annotations are used to specify the bean type.
- In this case, @MessageDriven specifies the destination monitored by thie MDB.
- The bean class need not implement the javax.ejb.MessageDrivenBean interface.
- The onMessage() method is activated on the reception of messages sent by a client application to the corresponding JMS destination.

e) How to define interceptor? What is the role of ArroundInvoke method? (5)

Defining an Interceptor

- Interceptor methods can be used to intercept either a business method or lifecycle event.
- An interceptor that intercepts a business method is typically called an AroundInvoke method because it can be defined by annotating the method with an @AroundInvoke annotation.
- An AroundInvoke method can be defined on:
    - Enterprise bean class
    - Interceptor class
- An interceptor class is a normal java class. It does not need to extend any special class or implement any interface.

AroundInvoke Method

- It can be defined by either of the following methods:
    - Annotating the method with an @AroundInvoke annotation.
    - Using an element in the bean's deployment descriptor.

It must satisfy the following requirements:

- One AroundInvoke method is allowed for each class.
- It must have a no argument public constructor.
- It must accept an javax.interceptor.InvocationCOntext object as an argument and return a java.lang.Object object.
- It can throw any application exception that is specified in the business interface or any runtime exception.
- It can be declared private, package private, protected or public.

It must call InvocationContext.proceed() to signal its intention to continue the invocation.

- Applying Interceptor
- Adding an interceptor to an enterprise bean
- Build and run the web application

f) What is dataSource Resource Definition in Java EE? (5)

- Java EE applications use datasource objects when they access relational databases through the JDBC API.
- A datasource has a set of properties that identify the data source that it represents.
- These properties include information such as :
    - The location of the database server
    - The name of the database
    - The network protocol to use to communicate with the server.
- A datasource object also works with a JNDI naming service.
- After registration the application can use the JNDI API to access that datasource object, which then can be used to connect to the datasource it represents.

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 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 Definition Using Deployment Descriptors

Q5 Attempt any three of the following: (15)

a) Explain Java Persistence API with specification. (5)

Java Persistence API with specification:

1. Java Persistence API is a specification, which is released under Java EE 5 specification.
2. It is neither an implementation nor a product. Hence, it cannot be used as it is for persistence. It needs an ORM implementation to work with and persist Java Objects.
3. Technically, JPA is just a set of interfaces [a Specification] and thus requires an implementation.

4. All specifications require vendors or open source projects to implement them. Using JPA therefore requires picking up an implementation [ORM tool such as Hibernate, Toplink, OpenJPA or any other ORM that implements JPA].
5. JPA defines the interface that an implementation has to implement.
6. The whole point of having a standard interface is that users can, in principle, switch between implementations of JPA without changing their code.
7. This way, JPA helps prevent Vendor Lock-In by relying on a strict specification such as JDO and EJB 2.x entities.
8. Currently most of the persistence vendors[ORM tool providers] have released implementations of JPA.
9. Since, the Java Persistence code spec covers several persistence frameworks into a single API, an application written using JPA will work across several implementations [Hibernate, Toplink and so on].

This is very useful, especially when, development begins using the free open source ORM implementations and later on when the need arises, the open source implementation is swapped with a commercial ORM implementation. The switch happens without changing the application code spec.

b) What is Impedance Mismatch? How it can be solved? (5)

First problem, what if we need to modify the design of our database after having developed a few pages or our application? Second, loading and storing objects in a relational database exposes us to the following five mismatch problems –

1 Granularity:

Sometimes you will have an object model, which has more classes than the number of corresponding tables in the database.

2 Inheritance:

RDBMSs do not define anything similar to Inheritance, which is a natural paradigm in object-oriented programming languages.

3 Identity:

An RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).

4 Associations:

Object-oriented languages represent associations using object references whereas an RDBMS represents an association as a foreign key column.

5 Navigation:

The ways you access objects in Java and in RDBMS are fundamentally different.

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

ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C#, etc. An ORM system has the following advantages over plain JDBC –

- Let's business code access objects rather than DB tables.

- Hides details of SQL queries from OO logic.
- Based on JDBC 'under the hood.'
- No need to deal with the database implementation.
- Entities based on business concepts rather than database structure.
- Transaction management and automatic key generation.
- Fast development of application.

c) What is the relationship between JPA, ORM, Database and application? (5)

JPA, ORM, Database and the Application

- JPA is made up of a few classes and interfaces.
- The application communicates with the configured JPA provider to access the underlying data.
- Applications invoke the appropriate methods of the JPA.
- The information about the mapping between the instance variables of classes and the columns of tables in the database is available either in XML and/or POJOs with the help of annotations.

ORM:
Object Relational Mapping is concept/process of converting the data from Object oriented language to relational DB and vice versa. For example in java its done with the help of reflection and jdbc.

Application:
Its the implementation of above concept.

Database:
Database can have number of tables. Each table can have number of attributes. Amongst the number of table some relations can be established with foreign key and primary key. One row is allotted for one record related to one object.

JPA:

- Its the one step above ORM. Its high level API and specification so that different ORM tools can implement so that it provides the flexibility to developer to change the implementation from one ORM to another (for example if application uses the JPA api and implementaion is hibernate.
- In future it can switch to IBatis if required. But on the other if application directly lock the implementation with Hibernate without JPA platform, switching is going to be herculean task)

d) Write a short note on functions in JPQL and Downcasting in JPQL. (5)

JPQL Functions

Funcrion	Descrption	Example
ABS	absolute value	ABS(e.salary - e.manager.salary)
CONCAT	concatenates two or more string values	CONCAT(e.firstName, ' ', e.lastName)
CURRENT_DATE	the current date on the database	CURRENT_DATE
CURRENT_TIME	the current time on the database	CURRENT_TIME
LENGTH	the character/byte length of the character or binary value	LENGTH(e.lastName)
LOCATE	the index of the string within the string, optionally starting at a start index	LOCATE('-', e.lastName)
LOWER	convert the string value to lower case	LOWER(e.lastName)
SQRT	computes the square root of the number	SQRT(o.result)
SUBSTRING	the substring from the string, starting at the index, optionally with the substring size	SUBSTRING(e.lastName, 0, 2)
UPPER	convert the string value to upper case	UPPER(e.lastName)

JPA 2.1 introduced the TREAT operator to JPQL which you can use to downcast anentity within your query.

You can, for example, create an entity model with Authors who have written differentkinds of Publications, like Books and BlogPosts. It's pretty obvious that Publication isthe super class of Book and BlogPost and that you have to model a relationshipbetween Author and Publication.

```
List<Object[]> result = em.createQuery(

 "SELECT a, p FROM Author a JOIN a.publications p

WHERE treat(p AS Book).title LIKE '%Java%'"). getResultList();
```

e) How hibernate works? (5)

An xml mapping document is created, which informs hibernate about:

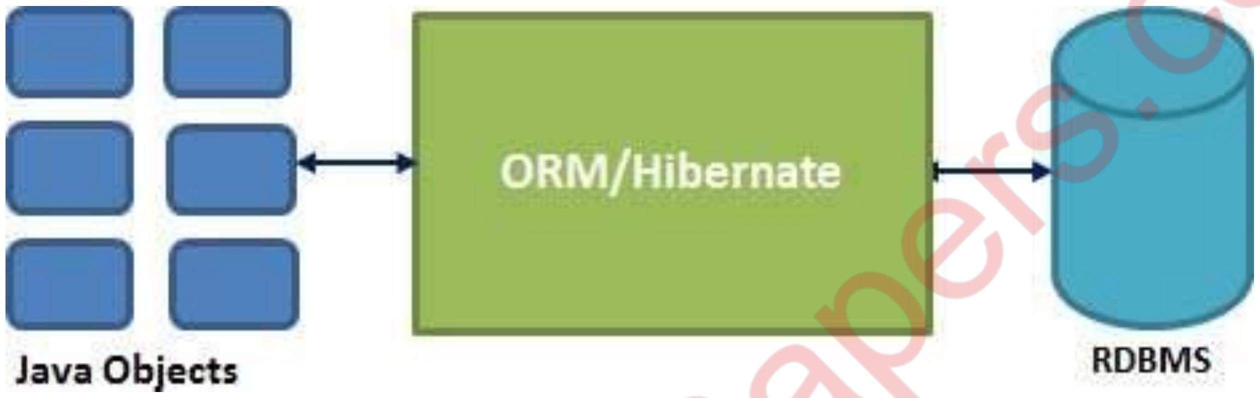
- The classes needed to store the data.
- How the classes are related to the tables & columns in the database.

- The mapping document is compiled, when the application starts up.
- It provides the framework with all the necessary information.
- During the runtime, hibernate reads the xml mapping document and dynamically builds java classes to manage the translation between the database and java objects.
- A session factory is created from the compiled collection of mapping document.
- The session class provides the interface between the persistence data store and the application.
- All the database interaction is done via a simple, native API that hibernate provides.

Components of Hibernate:

- **Connection Management**: provides efficient management of the database connection. Database connection is the most expensive part of an application that allows interacting with the database.
- **Transaction Management**: provides ability to execute more than one database statements at a time.
- **Object Relational Mapping**: is a technique of mapping the data representation from an object model to a relational model. This part is used to perform CRUD operations from the underlying database tables.

Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns.



f) Explain the structure of hibernate.cfg.xml. **(5)**

One of the most required configuration file in Hibernate is hibernate.cfg.xml file. By default, it is placed under src/main/resource folder. hibernate.cfg.xml file contains database related configurations and session related configurations.

Database configuration includes jdbc connection url, DB user credentials, driver class and hibernate dialect.

For example,

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration

DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

 <session-factory>

 <property

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

 <property
```

```
name="hibernate.connection.url">jdbc:mysql://localhost:3306/java2novice</property>
 <property name="hibernate.connection.username">root</property>
 <property name="hibernate.connection.password">root</property>
 <property
name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
 <property name="show_sql">false</property>
 </session-factory>
</hibernate-configuration>
```