CBCS SCHEME

USN | | | | | | | | | | |

20MCA33

Third Semester MCA Degree Examination, Jan./Feb. 2023
Advances in Java

Time: 3 hrs. Max. Marks: 100

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Module-1

1 a. Define Servlet. Explain life cycle of a Servlet with a neat diagram. **(10 Marks)**

b. Define Cookies. Write a JAVA Servlet program using Cookies to remember user preferences. **(10 Marks)**

OR

2 a. Explain 4 different ways in which a session can be tracked with suitable examples. **(10 Marks)**

b. Briefly explain any 5 HTTP status codes during server responses. **(05 Marks)**

c. Explain HTTP Request headers in brief. **(05 Marks)**

Module-2

3 a. Explain different kinds of main tags used in JSP with example for each. **(10 Marks)**

b. Explain different ways in which you can invoke JAVA code from JSP. **(10 Marks)**

OR

4 a. Explain the need, benefits and advantages of JSP over competing technologies. **(10 Marks)**

b. Write a JSP program that adds two numbers entered through HTML form and display the result **(10 Marks)**

Module-3

5 a. List all the attributes of Page directives tags in JSP and explain any five with an example for each. **(10 Marks)**

b. Write a JSP program to get student information through HTML form. Create a JAVA Bean class, populate Bean and display the same information through another JSP. **(10 Marks)**

OR

6 a. Define JAVA Bean and state its advantages. Explain the features of JAVA Bean. **(08 Marks)**

b. Write a note on JAR files. **(06 Marks)**

c. Write a JSP program to include an applet along with necessary applet code. **(06 Marks)**

Module-4

7 a. With an example, explain the essential steps of JDBC program. **(10 Marks)**

b. Explain in brief basic and advanced JDBC data types. **(10 Marks)**

OR

8 a. What are annotations? Discuss built-in annotations with an example. (10 Marks)

b. Explain the types of statements objects in JDBC with an example. (10 Marks)

Module-5

9 a. Explain the following terms of container services in EJB:

(i) Transactions

(ii) Security.

(iii) Naming and Object stores (10 Marks)

b. What is stateless session bean? Explain the life cycle of stateless session bean with a neat diagram. (10 Marks)

OR

10 a. What is message driven bean? Explain the life cycle of message driven bean. (10 Marks)

b. Write short notes on :

(i) Stateful session bean.

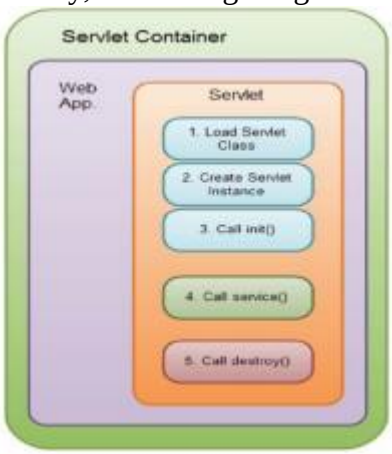
(ii) Singleton session bean. (10 Marks)

* * * * *

1.a Define Servlet? Explain the life cycle of a Servlet with a neat diagram.
Java Servlets are programs that run on a Web or Application server

 Act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
 Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
 Servlets are server side components that provide a powerful mechanism for developing web applications.

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 details.
The init() method :

 The init method is designed to be called only once.
 It is called when the servlet is first created, and not called again for each user request. 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.
 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.
Signature of service method:

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException
{
}
```

 The service () method is called by the container and service method invokes doGe, 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...
}
```

1.b. Define Cookies? Write a Servlet program using cookies to remember user preferences?
Cookies are small bits of textual information that a web server sends to a browser and that the
browser later returns unchanged when visiting the same web site or domain
Servlet1.java

```
package j2ee.prg4;
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
* Servlet implementation class store
*/
@WebServlet("/store")
public class store extends HttpServlet {
private static final long serialVersionUID = 1L;
/**
* @see HttpServlet#HttpServlet()
*/
public store() {
super();
// TODO Auto-generated constructor stub
}
/**
* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
*/
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Setting the HTTP Content-Type response header to text/html
response.setContentType("text/html;charset=UTF-8");
// Returns a PrintWriter object that can send character text to the client.
PrintWriter out=response.getWriter();
try
{
//Requesting input color from html page and storing in String variable s1
String s1=request.getParameter("color");
//Checking the color either RED or Green or Blue
if (s1.equals("RED")||s1.equals("BLUE")||s1.equals("GREEN"))
{

// Creating cookie object ck1 and storing the selected color
Cookie ck1=new Cookie("color",s1);
//adding the cookie to the response
response.addCookie(ck1);
//writing the output in the html format
out.println("<html>");
out.println("<body>");
out.println("You selected: "+s1);
out.println("<form action='retrieve' method='post'>");
out.println("<input type='Submit' value='submit'/>");
out.println("</form>");
out.println("</body>");
```

```
out.println("</html>");
}

}
finally
{
//Closing the output object
out.close();
}
}
}
```

retrieve.java

```
package j2ee.prg4;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
* Servlet implementation class retrieve
*/
@WebServlet("/retrieve")
public class retrieve extends HttpServlet {
private static final long serialVersionUID = 1L;
/**
* @see HttpServlet#HttpServlet()
*/
public retrieve() {
super();

// TODO Auto-generated constructor stub
}
/**
* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
*/
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

// Setting the HTTP Content-Type response header to text/html
response.setContentType("text/html;charset=UTF-8");
// Returns a PrintWriter object that can send character text to the client.
PrintWriter out=response.getWriter();
try
{
//Requesting all the cookies and stored in cookie array ck[]
Cookie ck[]=request.getCookies();
```

```
out.println("<html>");
out.println("<head>");
out.println("<title>servlet</title>");
out.println("</head>");
// Getting the value from cookie and setting the HTML form

background color

out.println("<body bgcolor="+ck[0].getValue()+">");
//Getting the value from cookie and displaying the color name in

HTML form

out.println("You selected color is: "+ck[0].getValue()+"</h1>");
out.println("</body>");
out.println("</html>");

}
finally
{
//closing the printwriter object out
out.close();
}
}
}
```

Index.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to the url store and the post method is used -->
<form action="store" method="post">
<!-- Display the Radio button with three option -->

RED:<input type="radio" name="color" value="RED"/><br>
GREEN:<input type="radio" name="color" value="GREEN"/><br>
BLUE:<input type="radio" name="color" value="BLUE"/><br>
<input type="submit" value="submit"/>
</form>
</body>
</html>
```

2.a. Explain 4 different ways in which session can be tracked with suitable examples.

Hidden Form:

```
<INPUT TYPE="HIDDEN" NAME="session" VALUE="a1234">
```

This entry means that, when the form is submitted, the specified name and valare automatically

included in the GET or POST data. This hidden field can be used tostore information about the
session but has the major disadvantage that it only works if every page is dynamically generated
by a form submission. Clicking on a regular hypertext link does not result in a form submission,
so hidden form fields cannot support general session tracking, only tracking within a specific series ooperations such as checking out at a store.

Cookies

Cookies are small bits of textual information that a web server sends to a browser and that the browser later returns unchanged when visiting the same web site or domain

Sending cookies to the client:

1.Creating a cookie object

• Cookie():constructs a cookie.
• Cookie(String name, String value)constructs a cookie with a specified name and value.

EX:

```
Cookie ck=new Cookie("user",”mca");
```

2.Setting the maximum age

setMaxAge() is used to specify how long (in seconds) the cookie should be valid.

Ex:cookie.setMaxAge(60*60*24);

3.Placing the cookie into the HTTP response headers.

We use response.addCookie to add cookies in the HTTP response header as follows:

```
response.addCookie(cookie);
```

Reading cookies from the client:

1. Call request.getCookies(). This yields an array of cookie objects.
2. Loop down the array, calling getName on each one until you find the cookie of interest.

Ex:

```
String cookieName=“userID”;
Cookie[] cookies=request.getCookies();
If(cookies!=null)
{
for(int i=0;i<cookies.length;i++){
Cookie cookie=cookies[i];
if(cookieName.equals(cookie.getName())){
doSomethingwith(cookie.getValue());
}}}
```

Session Tracking:

1. Accessing the session object associated with the currentrequest.

Call request.getSession to get an HttpSessionobject, which is a simple hash table for storing user-
specific data.

2. Looking up information associated with a session.

Call getAttribute on the HttpSession object, cast the return value to the appropriate type, and check whether the result is null.

3.Storing information in a session.

Use setAttribute with a key and a value.

4.Discarding session data.
Call removeAttribute to discard a specific value. Call invalidate to discard an entire session. Call
logout to log the client out of the Web server and invalidate all sessions associated with that user.

2.b. Briefly explain any 5 HTTP status codes during server responses.

200 (OK)
– Everything is fine; document follows.
– Default for servlets.
204 (No Content)
– Browser should keep displaying previous document.
301 (Moved Permanently)
– Requested document permanently moved elsewhere (indicated in Location header).
– Browsers go to new location automatically.
– Browsers are technically supposed to follow 301 and 302 (next page) requests only when the
incoming request is GET, but do it for POST with 303. Either way, the Location URL is retrieved
with GET.
– Servlets should use sendRedirect, not setStatus, when setting this header. See example.
• 401 (Unauthorized)
– Browser tried to access password-protected page without proper Authorization header.
• 404 (Not Found)
– No such page. Servlets should use sendError to set this.
– Problem: Internet Explorer and small (< 512 bytes) error pages. IE ignores small error page by
default.

2.c. Explain HTTP request headers in brief

When a browser requests for a web page, it sends lot of information to the web server which can
not be read directly because this information travel as a part of header of HTTP request. You can
check HTTP Protocol for more information on this.

Header Description

Accept This header specifies the MIME types that the browser or other clients can handle. Values of image/png orimage/jpeg are the two most common possibilities.

Accept-Charset This header specifies the character sets the browser can use to display

the information. For example ISO-8859-1.

Accept-Encoding This header specifies the types of encodings that the browser knows how to handle. Values of gzip orcompress are the two most common possibilities.

Accept-Language This header specifies the client's preferred languages in case the servlet

can produce results in more than one language. For example en, en-us, ru, etc.

Authorization This header is used by clients to identify themselves when accessing

password-protected Web pages.

Connection This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser

to retrieve multiple files with a single request. A value of Keep-Alive means that persistent connections should be used

Content-Length This header is applicable only to POST requests and gives the size of

the POST data in bytes.

Cookie This header returns cookies to servers that previously sent them to the

browser.

Host This header specifies the host and port as given in the original URL.

If-Modified-Since This header indicates that the client wants the page only if it has been changed after the specified date. The server sends a code, 304 which means Not Modifiedheader if no newer result is available.

If-Unmodified- This header is the reverse of If-Modified-Since; it specifies that the Since operation should succeed only if the document is older than the

specified date.

Referer This header indicates the URL of the referring Web page. For example, if you are at Web page 1 and click on a link to Web page 2, the URL of Web page 1 is included in the Referer header when the browser requests Web page 2.

User-Agent This header identifies the browser or other client making the request and can be used to return different content to different types of browsers.

3.a. Explain different kinds of main tags in JSP with example for each.

1) Expression Tag: (<%= ... %>)

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Syntax two forms:

<%= expr %>

<jsp:expression> expr </jsp:expression> (XML form)
2) Scriptlet Tag (<% ... %>)
A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.
Embeds Java code in the JSP document that will be executed each time the JSP page is processed.
Code is inserted in the service() method of the generated Servlet
Syntax two forms:
<% any java code %>
<jsp:scriptlet> ... </jsp:scriptlet>. (XML form)
Example
– <% if (Math.random() < 0.5) { %>

Have a <B>nice</B> day! <% } else { %>
Have a <B>lousy</B> day! <% } %>
• Representative result
– if (Math.random() < 0.5) { out.println("Have a <B>nice</B> day!"); } else {
out.println("Have a <B>lousy</B> day!");
}
3) Declaration Tag (<%! ... %>)
A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.
Code is inserted in the body of the servlet class, outside the service method.
o May declare instance variables.
o May declare (private) member functions.
Syntax two forms:
<%! declaration %>
<jsp:declaration> declaration(s)</jsp:declaration>
Example for declaration of Instance Variable:
<html>
<body>
<%! private int accessCount = 0; %>
 Accesses to page since server reboot:
<%= ++accessCount %>
</body></html>
4) Directive Tag (<%@ ... %>)
Directives are used to convey special processing information about the page to the JSP container.
The Directive tag commands the JSP virtual engine to perform a specific task, such as importing a Java package required by objects and methods.
Directive Description
<%@ page ... %> Defines page-dependent attributes, such as
scripting language, error page , and buffering
requirements.
<%@ include ... %> Includes a file during the translation phase.
<%@ taglib ... %> Declares a tag library, containing custom actions, used in the page
The page directive is used to provide instructions to the container that 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 page directive:
<%@ page attribute="value" %>
You can write XML equivalent of the above syntax as follows:
<jsp:directive.page attribute="value" />

3.b. Explain different ways in which you can invoke java code from JSP.

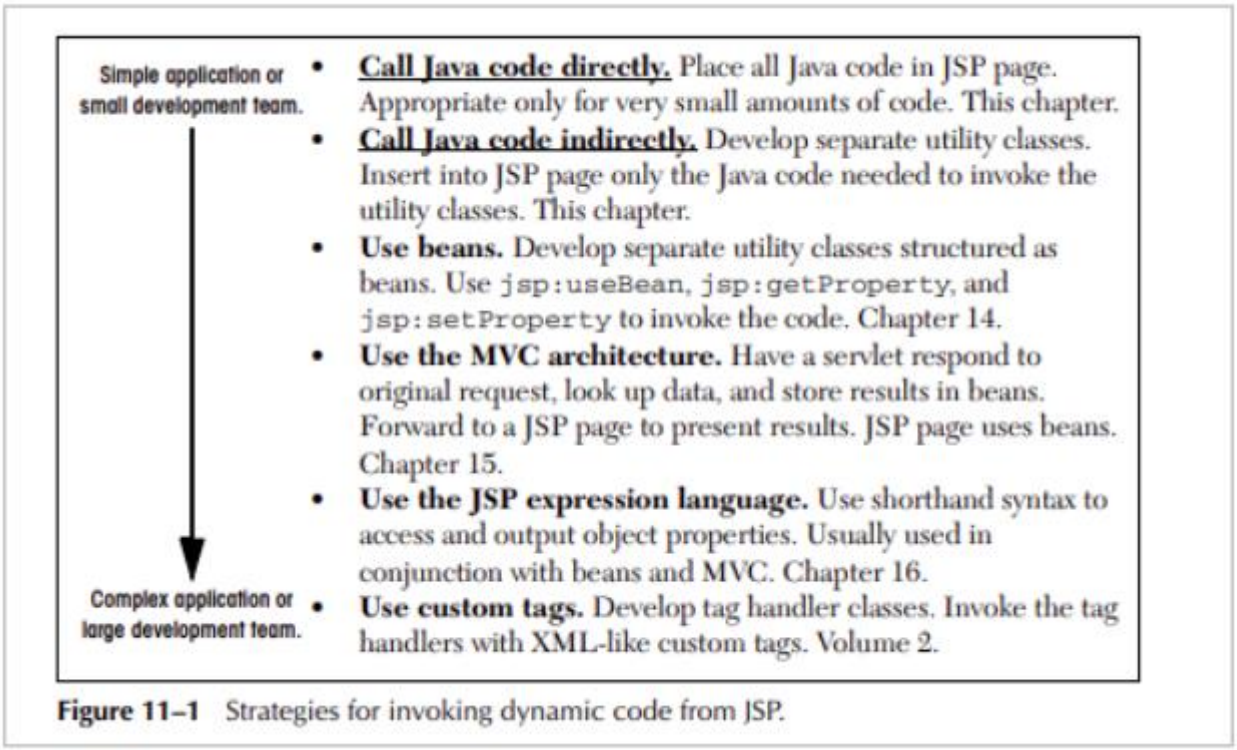


Figure 11–1 Strategies for invoking dynamic code from JSP.

4.a. Explain the need, benefits and advantages of JSP over competing technologies.

Need of JSP:
 JSP provides an easier way to code dynamic web pages.
 JSP does not require additional files like, java class files, web.xml etc

 Any change in the JSP code is handled by Web Container (Application server like tomcat),
and doesn't require re-compilation.
 JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.
Advantages of JSP:
Extension to Servlet
JSP technology is the extension to servlet technology. We can use all the features of 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.
Easy to maintain
JSP can be easily managed because we can easily separate our business logic with presentation
logic. In servlet technology, we mix our business logic with the presentation logic.

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.
Less code than Servlet
In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code.
Moreover, we can use EL, implicit objects
Benefits of JSP:
JSP pages are translated into servlets. So, fundamentally, any task JSP pages can perform could also be accomplished by servlets. However, this underlying equivalence does not mean that servlets and JSP pages are equally appropriate in all scenarios. The issue is not the power of the technology, it is the convenience, productivity, and maintainability of one or the other. After all, anything you can do on a particular computer platform in the Java programming language you could also do in assembly language. But it still matters which you choose. JSP provides the following benefits over servlets alone:
• It is easier to write and maintain the HTML. Your static code is ordinary HTML: no extra backslashes, no double quotes, and no lurking Java syntax.
• You can use standard Web-site development tools. For example, we use Macromedia Dreamweaver for most of the JSP pages in the book. Even HTML tools that know nothing about JSP can be used because they simply ignore the JSP tags.
• You can divide up your development team. The Java programmers can work on the dynamic code. The Web developers can concentrate on the presentation layer. On

large projects, this division is very important. Depending on the size of your team and the complexity of your project, you can enforce a weaker or stronger separation between the static HTML and the dynamic content.

4.b. Write a JSP program to add 2 numbers entered through HTML form and display the result.

index.html

```
<html>
    <head>
        <title>Enter two numbers to add up</title>
    </head>

    <body>
        <form action="./add.jsp">
            First number: <input type="text" name="t1"/>
            Second number: <input type="text" name="t2"/>
            <input type="submit" value="SUBMIT" />
        </form>
    </body>

/html>
```

add.jsp

```
<html>
    <head>
        <title>Enter two numbers to add up</title>
    </head>

    <body>
    <%= "<h1> The sum is
"+(Integer.parseInt(request.getParameter("t1"))+Integer.parseInt(request.getParame
ter("t2")))+"</h1>"%>
    </body>
</html>
```

5.a. List all the attributes of Page directive tag. Explain any 5 of them with examples.

1)import

The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.

2. Example of import attribute

```
<html>
<body>
 <%@ page import="java.util.Date" %>
Today is: <%= new Date() %>
 </body>
</html>
```

2. errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example of errorPage attribute

```
//index.jsp
 <html>
 <body>
<%@ page errorPage="myerrorpage.jsp" %>
<%= 100/0 %>
 </body>
 </html>
```

3.isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

```
//myerrorpage.jsp
 <html>
<body>
 <%@ page isErrorPage="true" %>
Sorry an exception occured!<br/>
The exception is: <%= exception %>
 </body>
  </html>
```

4.contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of
the HTTP response.The default value is "text/html;charset=ISO-8859-1".
Example of contentType attribute

```
 <html>
<body>
<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

5. buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.
Example of buffer attribute

```
<html>
 <body>
<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

6.Autoflush
The autoFlush attribute controls whether the output buffer should be automatically flushed when it is full (the default) or whether an exception should be raised when the buffer overflows (autoFlush="false"). Use of this attribute takes one of the following two forms.

```
<%@ page autoflush=“false” %>
```

7. Write a JSP Program which uses jsp:include and jsp:forward action to display a Webpage.
ndex.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to login.jsp and the get method is used -->
<form method="get" action="login.jsp">
UserName : <input type="text"
name ="name"><br> Password
: <input type="password" name
="pass"><br>
<input type="Submit" value ="Submit"/><br>
</form>
</body>
</html>
```

login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
//Getting the input name from the html form
and storing in String ‘uname’ String uname =
request.getParameter("name");
//Getting the input pass from the html form
and storing in String ‘upass’ String upass =
request.getParameter("pass");
if(uname.equals("admin") && upass.equals("admin"))
{

%>
<jsp:forward page="main.jsp"></jsp:forward>
<%
}
else
{
out.println("Wrong Credentials Username and
Password"+"<br>"); out.println("Enter Corrects Username
and Password.. Try again" +"<br><br>");%>
<jsp:include page="index.jsp"></jsp:include>
<%
}%>
</body>
</html>
```

main.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
// Getting the input name from the html form
```

```
and storing in String ‘un’--> String
un=request.getParameter("name");
// Getting the input pass from the html form
and storing in String ‘pw’--> String
pw=request.getParameter("pass");
%>
<h1>welcome:<%=un%></h1>
<h1>your user name is:<%=un%></h1>
<h1>your password is:<%=pw%></h1>
</body>
</html>
```

5.b Write a JSP program to get Student information through HTML form. Create a Java bean class,populate bean and display the same information through JSP

student.java

```
package program8;
public class stud
{
	public String sname;
	public String rno;
	//Set method for Student name
	public void setsname(String name)
	{
		sname=name;
	}
	//Get method for Student name
	public String getsname()
	{
		return sname;
	}
	//Set method for roll no
	public void setrno(String no)
	{
		rno=no;
	}
	//Get method for roll no
	public String getrno()
	{
		return rno;
	}
}
```

display.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- Using the studb bean -->
<jsp:useBean id ="studb" scope = "request" class =
"program8.stud"></jsp:useBean> Student Name :
<jsp:getProperty name="studb" property="sname"/><br/>
Roll No. : <jsp:getProperty name="studb" property="rno"/><br/>
</body>
</html>
```

first.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- Create the bean studb and set the property -->
<jsp:useBean id="studb" scope="request" class="program8.stud"></jsp:useBean>
<jsp:setProperty name="studb" property='*'/>
<jsp:forward page="display.jsp"></jsp:forward>
</body>
</html>
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to first.jsp -->
```

<form action="first.jsp">
Student Name : <input type="text"
name = "sname"> Student Roll no :
<input type="text" name = "rno">
<input type = "submit" value="Submit"/>
</form>
</body>
</html>

6.a. Define Java bean State its advantages. Explain the features of Java Bean.

Software components are self-contained software units developed according to the motto "Developed them once, run and reused them everywhere". Or in other words, reusability is the main concern behind the component model.

A software component is a reusable object that can be plugged into any target software application.

You can develop software components using various programming languages, such as C, C++, Java, and Visual Basic.

- A "Bean" is a reusable software component model based on sun's java bean specification that can be manipulated visually in a builder tool.
- The term software component model describe how to create and use reusable software components to build an application
- Builder tool is nothing but an application development tool which lets you both to create new beans or use existing beans to create an application.
- To enrich the software systems by adopting component technology JAVA came up with the concept called Java Beans.
- Java provides the facility of creating some user defined components by means of Bean programming.
- We create simple components using java beans.
- We can directly embed these beans into the software.

Advantages of Java Beans:

- The java beans posses the property of "Write once and run anywhere".
- Beans can work in different local platforms.
- Beans have the capability of capturing the events sent by other objects and vice versa enabling object communication.
- The properties, events and methods of the bean can be controlled by the application developer.(ex. Add new properties)
- Beans can be configured with the help of auxiliary software during design time.(no hassle at runtime)
- The configuration setting can be made persistent.(reused)
- Configuration setting of a bean can be saved in persistent storage and restored later.

6.b. Write a note on JAR files

JAR files are packaged with the ZIP file format, so you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking. These tasks are among the most common uses of JAR files, and you can realize many JAR file benefits using only these basic features.

Creating a JAR File

The basic format of the command for creating a JAR file is:

jar cf jar-file input-file(s)

The options and arguments used in this command are:

The c option indicates that you want to create a JAR file.

The f option indicates that you want the output to go to a file rather than to stdout. jar-file is the name that you want the resulting JAR file to have. You can use any

filename for a JAR file. By convention, JAR filenames are given a .jar extension, though this is not required. The input-file(s) argument is a space-separated list of one or more files that you want to include in your JAR file. The input-file(s)argument can contain the wildcard * symbol. If any of the "input-files" are directories, the contents of those directories are added to the JAR archive recursively.

The c and f options can appear in either order, but there must not be any space between them. This command will generate a compressed JAR file and place it in the current directory. c - creates a new or empty archive pm the Std output

t - lists the table of contents from std output

X file – it extracts all files or just the named files

f - The argument following this option specifies a JAR file to work v

- It generates verbose output on stderr

m - It includes manifest information from a specified manifest file

0 - it indicates ‘store only’ without using ZIP compression

M - it specifies that a manifest file should not be created for the entries

u - It updates an exisiting JAR file by adding files or changing the

6.b. Write a JSP program to include an applet along with necessary applet code?

```
import
java.awt.*
; import
java.apple
t.*;
```

```
import
java.awt.e
vent.*;
public class DemoApplet extends Applet
{
     public void paint(Graphics g)
     {
          setBackgroun
          d(Color.pink);
          setForeground
          (Color.black);
          g.drawString("Welcome JSP-Applet",100,100);
     }
}
```

AppletJsp.jsp

```
<html>
<body>
<jsp:plugin type="applet" code="DemoApplet.class" width="400" height="400">
   <jsp:fallback>
       <p>Unable to load applet</p>
   </jsp:fallback>
</jsp:plugin>
</body>
</html>
```

7.a. With an example explain the essential steps of JDBC program.

The following 5 steps are the basic steps involve in connecting a Java application with

Database using JDBC.

Register the Driver

Create a Connection

Create SQL Statement

Execute SQL Statement

Closing the connection

Register the Driver

Class.forName() is used to load the driver class explicitly.

Example to register with JDBC-ODBC Driver

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Create a Connection

getConnection() method of DriverManager class is used to create a connection.

Syntax

getConnection(String url)

getConnection(String url, String username, String password)

getConnection(String url, Properties info)

Example establish connection with Oracle Driver

Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:XE","username","password");

Create SQL Statement

createStatement() method is invoked on current Connection object to create a SQL Statement.

Syntax

public Statement createStatement() throws SQLException

Example to create a SQL statement

Statement s=con.createStatement();

Execute SQL Statement

executeQuery() method of Statement interface is used to execute SQL statements.

Syntax

public ResultSet executeQuery(String query) throws SQLException

Example to execute a SQL statement

ResultSet rs=s.executeQuery("select * from user");

while(rs.next())

{

System.out.println(rs.getString(1)+" "+rs.getString(2));

}

Closing the connection

After executing SQL statement you need to close the connection and release the session.

The close() method of Connection interface is used to close the connection.

Syntax

```
public void close() throws SQLException
```

Example of closing a connection

```
con.close();
```

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class

Class.forName("oracle.jdbc.driver.OracleDriver");
//step2 create the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
//step3 create the statement object
Statement stmt=con.createStatement();
//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
//step5 close the connection object
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

7.b. Explain the brief basic and advanced data types?

Basic

1. **CHAR, VARCHAR, and LONGVARCHAR**

CHAR

Represents a small, fixed-length character string

SQL CHAR type corresponding to JDBC CHAR is defined in SQL-92 and is supported by all the major databases

CHAR(12) defines a 12-character string.

All the major databases support CHAR lengths up to at least 254 characters. To retrieve the data from CHAR, ResultSet.getString method will be used.

VARCHAR
VARCHAR represents a small, variable-length character string
It takes a parameter that specifies the maximum length of the string. VARCHAR(12) defines a string whose length may be up to 12 characters.

All the major databases support VARCHAR lengths up to 254 characters.

When a string value is assigned to a VARCHAR variable, the database remembers the length of the assigned string and on a SELECT, it will return the exact original string.
To retrieve the data from VARCHAR, ResultSet.getString method will be used.

LONGVARCHAR

LONGVARCHAR represents a large, variable-length character string No consistent SQL mapping for the JDBC LONGVARCHAR type.

All the major databases support some kind of very large variable-length string supporting up to at least a gigabyte of data, but the SQL type names vary
These methods are getAsciiStream and getCharacterStream, which deliver the data stored in a LONGVARCHAR column as a stream of ASCII or Unicode characters.

2. BINARY, VARBINARY, and LONGVARBINARY

BINARY
Represents a small, fixed-length binary value

SQL BINARY type corresponding to JDBC BINARY is defined in SQL-92 and is supported by all the major databases
BINARY(12) defines a 12-byte binary value.
To retrieve the data from BINARY, ResultSet.getBytes method will be used.

VARBINARY
VARBINARY represents a small, variable-length binary value

It takes a parameter that specifies the maximum binary bytes. BINARY(12) defines a 12-byte binary type.
BINARY values are limited to 254 bytes.
To retrieve the data from BINARY, ResultSet.getBytes method will be used

LONGVARBINARY
LONGVARBINARY represents a large, variable-length byte value No consistent SQL mapping for the JDBC LONGVARBINARY type.

JDBC LONGVARBINARY stores a byte array that is many megabytes long, however, the method getBinaryStream is recommended

3. BIT

The JDBC type BIT represents a single bit value that can be zero or one. SQL-92 defines an SQL BIT type.

Portable code may use the JDBC SMALLINT type, which is widely supported. The recommended Java mapping for the JDBC BIT type is as a Java boolean.

4. TINYINT

The JDBC type TINYINT represents an 8-bit integer value between 0 and 255 that may be signed or unsigned.

Portable code may use the JDBCSMALLINT type, which is widely supported.

Java mapping for the JDBC TINYINT type is as either a Java byte or a Java short. Represents a signed value from -128to 127

16-bit Java short will always be able to hold all TINYINT values.

5. SMALLINT

Represents a 16-bit signed integer value between -32768 and 32767.

The recommended Java mapping for the JDBC SMALLINT type is as a Java short.

6. INTEGER

Represents a 32-bit signed integer value ranging between -2147483648 and 2147483647.

SQL type, INTEGER, is defined in SQL-92 and is widely supported by all the major databases.

Recommended Java mapping for the INTEGER type is as a Java int.\

Advanced Data Types

1. BLOB

- The JDBC type BLOB represents an SQL3 BLOB (Binary Large Object).
- A JDBC BLOB value is mapped to an instance of the Blob interface in the Java programming language.
- A Blob object logically points to the BLOB value on the server rather than containing its binary data, greatly improving efficiency.
- The Blob interface provides methods for materializing the BLOB data on the client when that is desired.

2. CLOB

- The JDBC type CLOB represents the SQL3 type CLOB (Character Large Object).
- A JDBC CLOB value is mapped to an instance of the Clob interface in the Java programming language.
- A Clob object logically points to the CLOB value on the server rather than containing its character data, greatly improving efficiency.

- Two of the methods on the Clob interface materialize the data of a CLOB object on the client.

3. ARRAY

- The JDBC type ARRAY represents the SQL3 type ARRAY.
- An ARRAY value is mapped to an instance of the Array interface in the Java programming language.
- An Array object logically points to an ARRAY value on the server rather than containing the elements of the ARRAY object, which can greatly increase efficiency.
- The Array interface contains methods for materializing the elements of the ARRAY object on the client in the form of either an array or a ResultSet object.

Example : ResultSet rs = stmt.executeQuery("SELECT NAMES FROM STUDENT"); rs.next();
Array stud_name=rs.getArray("NAMES");

4. DISTINCT

- The JDBC type DISTINCT represents the SQL3 type DISTINCT.
- For example, a DISTINCT type based on a CHAR would be mapped to a String object, and a DISTINCT type based on an SQL INTEGER would be mapped to an int.
- The DISTINCT type may optionally have a custom mapping to a class in the Java programming language.
- A custom mapping consists of a class that implements the interface SQLData and an entry in a java.util.Map object.

5. STRUCT

- The JDBC type STRUCT represents the SQL3 structured type.
- An SQL structured type, which is defined by a user with a CREATE TYPE statement, consists of one or more attributes. These attributes may be any SQL data type, built-in or user- defined.
- A Struct object contains a value for each attribute of theSTRUCT value it represents.
- A custom mapping consists of a class that implements the interface SQLData and an entry in a java.util.Map object.

6. REF

- The JDBC type REF represents an SQL3 type REF<structured type>.
- An SQL REF references (logically points to) an instance of an SQL structured type, which the REF persistently and uniquely identifies.

8.a. What are annotations? Discuss built-in annotations with an example.

1. @Retention

@Retention is designed to be used only as an annotation to another annotation.

It specifies the retention policy.

· A retention policy determines at what point annotation should be discarded.

· Java defined 3 types of retention policies through java.lang.annotation.RetentionPolicy enumeration. It has SOURCE, CLASS and RUNTIME.

· Annotation with retention policy SOURCE will be retained only with source code, and discarded during compile time.

· Annotation with retention policy CLASS will be retained till compiling the code, and discarded during runtime.

· Annotation with retention policy RUNTIME will be available to the JVM through runtime.

· The retention policy will be specified by using java built-in annotation @Retention, and we have to pass the retention policy type.

The default retention policy type is CLASS.

2. @Documented

The @Documented annotation is a marker interface that tells a tool that an annotation is to be documented. It is designed to be used only as an annotation to an annotation declaration. By default, annotation are not included in javadoc(is a documentation generator). But if @document is used, it then will be processed by javadoc like toolas and the annotation type information will also be included in generated document .

3. @Target

The @Target annotation specifies the types of declarations to which an annotation can be applied. It is designed to be used only as an annotation to another annotation. @Target takes one argument, which must be a constant from the ElementType enumeration. This argument specifies the types of declarations to which the annotation can be applied. The constants are shown here along with the type of declaration to which they correspond.

Target Constant Annotation Can Be Applied To

ANNOTATION_TYPE Another annotation

CONSTRUCTOR Constructor

FIELD Field

LOCAL_VARIABLE Local variable

METHOD Method

PACKAGE Package

PARAMETER Parameter

TYPE Class, interface, or enumeration

we can specify one or more of these values in a @Target annotation. To specify multiple values, we must specify them within a braces-delimited list. For example, to specify that an annotation applies only to fields and local variables, we can use this @Target annotation: @Target({ ElementType.FIELD, ElementType.LOCAL_VARIABLE }) 4. @Inherited

@Inherited is a marker annotation that can be used only on another annotation declaration. it affects only annotations that will be used on class declarations. @Inherited causes the annotation for a superclass to be inherited by a subclass.

Therefore, when a request for a specific annotation is made to the subclass, if that annotation is not present in the subclass,then its superclass is checked. If that annotation is present in the superclass, and if it is annotated with @Inherited, then that annotation will be returned.

java.lang.annotation.Inherited

```
@Inherited
public @interface MyAnnotation {
}
@MyAnnotation
public class MySuperClass { ... }
public class MySubClass extends MySuperClass { ... }
```

In this example the class MySubClass inherits the annotation @MyAnnotation because MySubClassinherits from MySuperClass, and MySuperClass has a @MyAnnotation annotation.

5. @Override

@Override is a marker annotation that can be used only on methods. A method annotated with @Override must override a method from a superclass. If it doesn't, a compile-time error will result. It is used to ensure that a superclass method is actually overridden, and not simply overloaded.

6. @Deprecated

@Deprecated is a marker annotation. It indicates that a declaration is obsolete and has been replaced by a newer form. This annotation is used to mark a class, method or field as deprecated, meaning it should on longer be used If your code uses deprecated classes, methods or fields the compiler will give you a warning.

```
@Deprecated
Public class MyComponent
{
}
```

The use of the @Deprecated annotation above the class declaration marks the class as deprecated.

The use of the @Deprecated annotation above the fieldclass declaration marks the field as deprecated.

7. @SuppressWarnings specifies that one or more warnings that might be issued by the compiler are to be suppressed. The warnings to suppress are specified by name, in string form. This annotation can be applied to any type of declaration.

@SuppressWarnings

– Makes the compiler suppress warnings for a given methods

– If a method class a deprecated method, or makes an insecure type

case, the compiler may generate a warning.

– You can suppress these warnings by annotating the method containing

the code with the @SuppressWarnings annotation

@ SuppressWarnings

public void methodWithWarning()

{

}

8.b. Explain the types of statements objects in JDBC with an example.

Prepared Statement:

The preparedStatement object allows you to execute parameterized queries.

A SQL query can be precompiled and executed by using the

PreparedStatement object. · Ex: Select * from publishers where pub_id=?

Here a query is created as usual, but a question mark is used as a placeholder

for a value· that is inserted into the query after the query is compiled.

The preparedStatement() method of Connection object is called to

return the· PreparedStatement object.

Ex: PreparedStatement stat; stat= con.prepareStatement("select * from

publisher where pub_id=?")

```
import java.sql.*;

public class JdbcDemo {
   public static void main(String args[]){
    try{
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection("jdbc:odbc:MyDataSource","khutub","");
      PreparedStatement pstmt;
      pstmt= con.prepareStatement("select * from employee whereUserName=?");
      pstmt.setString(1,"khutub");
      ResultSet rs1=pstmt.executeQuery();
      while(rs1.next()){
        System.out.println(rs1.getString(2));
      }
    } // end of try
    catch(Exception e){System.out.println("exception"); }
   } //end of main
   } // end of class
```

Statement Object

- The Statement object is used whenever J2EE component needs to immediately execute a query without first having the query compiled.

Statement Object contains 3 methods:

1. **Execute()** □ **(**used for **DDL** commands like, **Create, Alter, Drop)**
2. **executeUpdate()** □**(**Used for **DML** commands like, **Insert, Update, Delete)**
3. **exceuteQuery()** □ **(**Used for **Select** command**)**

- The **execute()** method is used during execution of DDL commands and also used when there may be multiple results returned.
- The **executeUpdate()** executes INSERT, UPDATE, DELETE, and returns an int value specifying the number of rows affected or 0 if zero rows selected
- The **executeQuery()** method, which passes the query as an argument. The query is then transmitted to the DBMS for processing.
- The executeQuery() □ method executes a simple select query and returns a ResultSet object.
- The ResultSet object contains rows, columns, and metadata that represent data requested by query.

```
importjava.sql.*;

public class StatementDemo {
   public static void main(String args[]){
    try{
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection("jdbc:odbc:MyDataSource","khutub","");
      Statement stmt;
      stmt= con.prepareStatement("select * from employee where Name='abc'");
      ResultSet rs=stmt.executeQuery();
      while(rs.next()){
         System.out.println(rs.getString(1));
      }
    } // end of try
    catch(Exception e){ System.out.println("exception" + e); }
   } //end of main
   } // end of class
```

The Bold line can be replace by any of the statement examples-1, 2 and 3

- The CallableStatement object is used to call a stored procedure from within a J2EE object.
- A Stored procedure is a block of code and is identified by a unique name.
- The type and style of code depends on the DBMS vendor and can be written in PL/SQL, Transact-SQL, C, or other programming languages.
- IN, OUT and INOUT are the three parameters used by the CallableStatement object to call a stored procedure.
- The IN parameter contains any data that needs to be passed to the stored procedure and whose value is assigned using the setxxx() method.
- The OUT parameter contains the value returned by the stored procedures. The OUT parameters must be registered using the registerOutParameter() method, later retrieved by using the getxxx()

- The INOUT parameter is a single parameter that is used to pass information to the stored procedure and retrieve information from the stored procedure.

Example:

```
Connectio
con;  try{
```

- String **query** = "{**CALL LastOrderNumber(?))**}";
- CallableStatement stat = con.prepareCall(query); stat.registerOutPara meter(1 ,Types.VARCHAR); stat**.execute();**
- String lastOrderNumber = stat.getString(1); stat**.close();**
- }

catch (Exception **e){}**

9. Explain the following terms of container services in EJB.

i) Transactions

ii) Security

iii) Naming and Object Store

i) Transaction

ACID Porperties

1. Atomicity

2. Consistency

3. Isolation

4. Durability

Integration of Java Transaction Service

ii)Security:

Includes identification of user(s) or system accessing the application and allowing or denying the access to resources within the application.

• Declarative security in which EJB container manages the security concerns or Custom code can be done in EJB to handle security concern by self.

1. Authentication

2. Authorization

3. User

4. User Groups

5. User Roles

iii)Naming and Object Stores

Provide clients with a mechanism for locating distributed objects or resources.

Naming service much fulfill two requirements:

1. object binding and a lookup API.

Object binding is the association of a distributed object with a natural language name or identifier.

2. A lookup API provides the client with an interface to the naming system;

• Allows us to connect with a distributed service and request a remote reference to a specific object.

Enterprise JavaBeans mandates the use of Java Naming and Directory Interface (JNDI)as a lookup API on Java clients.

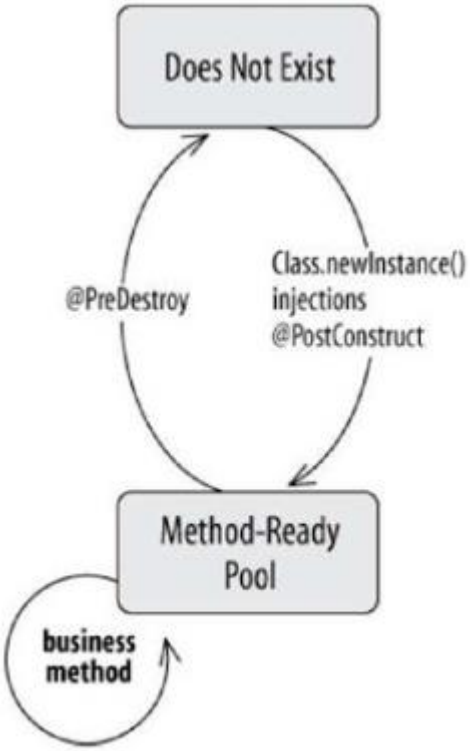
• JNDI supports any kind of naming and directory service.

• It is complex but, the way JNDI is used in Java Enterprise Edition (EE) applications is usually simple.

• Java client applications can use JNDI to initiate a connection to an EJB server and locate a specific EJB.

9.b What is Stateless Session bean ? Explain its lifecycle with a neat diagram

- If EJB is a grammar, session beans are the verbs.
- Session beans contain business methods.
- It can be invoked by local, remote and webservice client.
- It stores data of a particular user for a single session
- To access an application that is deployed on the server, the client invokes the session bean’s methods.
- The session bean performs work for its client, reducing the complexity by executing business tasks inside the server.



- It has only two states: Does Not Exist and Method-Ready Pool.
- The Method-Ready Pool is an instance pool of stateless session bean objects that are not in use.
- This is an important difference between stateless and stateful session beans; stateless beans define instance pooling in their lifecycles and stateful beans do not.
- Figure illustrates the states and transitions an SLSB instance goes through.

The Does Not Exist State:

- **Does Not Exist**. In this state, the bean instance simply does not exist.
- **Ready**. When WebLogic server is first started, several bean instances are created and placed in the Ready pool. More instances might be created by the container as needed by the EJB container.

The Method Ready Pool:

- Stateless bean instances enter the Method-Ready Pool as the container needs them.
- When the EJB server is first started, it may create a number of stateless bean instances and enter them into the Method-Ready Pool.
- When the number of stateless instances servicing client requests is insufficient, more can be created and added to the pool.

Moving from the Does Not Exist State to the Ready State

- 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 by invoking the Class.newInstance() method on the stateless bean class.
- Second, the container injects any resources that the bean's metadata has requested via an injection annotation or XML deployment descriptor.

- **Finally, the EJB container will fire a post-construction event.**
- The bean class can register for this event by annotating a method with @javax.annotation.PostConstruct.
- The @PreDestroy method should close any open resources before the stateless session bean is evicted from memory at the end of its lifecycle.

Moving from the Ready State to Does Not Exist State:

- When the EJB container decides to reduce the number of session bean instances in the ready pool, it makes the bean instance ready for garbage collection

10.a. What is Message Driven Bean? Explain the life cycle of Message Driven Bean?

- MDB are stateless, server-side, transaction-aware components for processing asynchronous message delivered via JMS Asynchronous message is a paradigm in which two or more applications communicate via a message describing a business event.
- A message driven bean (MDB) is a bean that contains business logic. But, it is invoked by passing the message. Hence , it is like JMS Receiver.
- MDB asynchronously receives the message and processes it.
- A message driven bean receives message from queue or topic
- A message driven bean is like stateless session bean that encapsulates the business logic and doesn't maintain state.
- Messaging is a technique to communicate applications or software components.
- JMS is mainly used to send and receive message from one application to another.
- JMS (Java Message Service) is an API that provides the facility to create, send and read messages.
- It provides loosely coupled, reliable and asynchronous communication. JMS is also known as a messaging service.
- There are two types of messaging domains in JMS.

  1. Queue Model - Point-to-Point Messaging Domain

  2. Topic Model -Publisher/Subscriber Messaging Domain

10.b. Write a short note on the following

i) Stateful Session Bean

ii) Singleton

i)Stateful Session Bean:

- A stateful session bean is a type of enterprise bean, which preserve the conversational state with client.
- It Keeps associated client state in its instance variables.
- EJB Container creates a separate stateful session bean to process client's each request.
- As soon as request scope is over, statelful session bean is destroyed.
- Every request upon a given proxy reference is guaranteed to ultimately invoke upon the same bean instance.
- Each SFSB proxy object has an isolated session context, so calls to one session will not affect another.

- Stateful sessions, and their corresponding bean instances, are created sometime before the first invocation upon a proxy is made to its target instance (as shown in Figure).
- They live until the client invokes a method that the bean provider has marked as a remove event, or until the Container decides to remove the session.

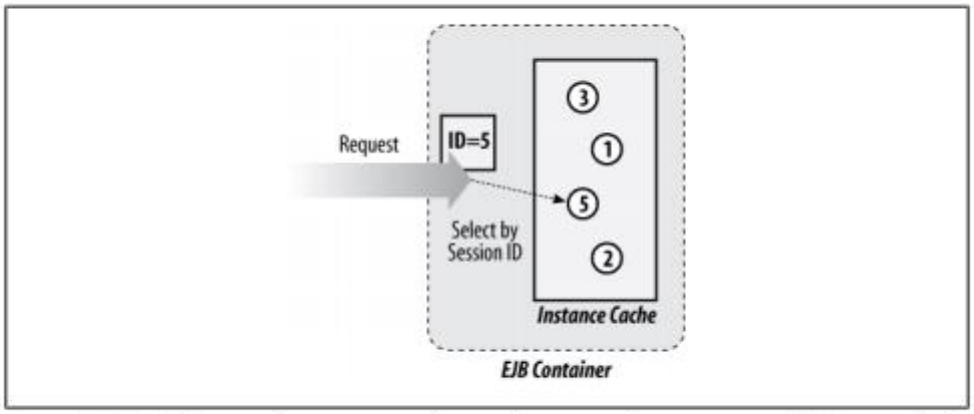


Figure 2-3. Stateful session bean creating and using the correct client instance, which lives inside the EJB Container, to carry out the invocation

ii) Singleton

- **Singleton session beans are appropriate in the following circumstances.**
- State needs to be shared across the application.
- A single enterprise bean needs to be accessed by multiple threads concurrently.
- The application needs an enterprise bean to perform tasks upon application startup and shutdown.
- The bean implements a web service.
- A session bean must be annotated or denoted in the deployment descriptor as a stateless or stateful or singleton session bean. These annotations are component-defining annotations and are applied to the bean class.