CBCS SCHEME

USN | 1 | C | R | 2 | 0 | M | C | 0 | 3 | 1 |

20MCA33

Third Semester MCA Degree Examination, Feb./Mar. 2022
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. Explain briefly servlet architecture and life cycle method. **(10 Marks)**

b. Write a JAVA servlet program which reads two parameters from the web page say value 1, value 2, which are of type integers, and finds the sum of the two values and return bock the result as a webpage. **(06 Marks)**

c. Write any four difference between GET and PSOT request. **(04 Marks)**

OR

2 a. Explain briefly any five methods of HttpServletRequest. Illustrate it with a simple program. **(10 Marks)**

b. Write a JAVA Servlet program using cookies to remember user preferences. **(06 Marks)**

c. Explain briefly advantages of servlet over CGI. **(04 Marks)**

Module – 2

3 a. Explain the following tags with an example :
i) Declaration ii) Scriptlet iii) Expression iv) Comment. **(10 Marks)**

b. With a neat diagram, explain JSP architecture and life cycle phases of JSP. **(10 Marks)**

OR

4 a. Explain briefly any five JSP implicit object. **(10 Marks)**

b. Explain briefly three life cycle methods of JSP. **(06 Marks)**

c. Write a JSP program to perform arithmetic operation using scriptlet, declaration and expression tag. **(04 Marks)**

Module – 3

5 a. Explain briefly any 10 attributes of JSP page directive tag. **(10 Marks)**

b. Write a JSP program which uses <jsp:include> and <jsp:forward> standard action to display a webpage. **(10 Marks)**

OR

6 a. Explain the following standard action with suitable example.
i) <jsp:useBean> ii) <jsp:plugin> . **(10 Marks)**

b. Write a JSP program to get students information through a HTML and create a JAVA bean class populate bean and display the same information through another JSP. **(10 Marks)**

Module – 4

7 a. Explain briefly any 5 builtin annotation with a suitable example. **(10 Marks)**

b. Explain briefly JDBC routine process. Give an example. **(10 Marks)**

OR

8 a. Discuss any five advanced JDBC data types. **(10 Marks)**

b. Discuss the types of JDBC statements with an example. **(10 Marks)**

Module – 5

9 a. With a neat diagram explain the life cycle of stateful ession bean. **(10 Marks)**

b. Explain briefly stateless, stateful and singleton session bean. **(06 Marks)**

c. List out the differences between stateless and stateful session bean. **(04 Marks)**

OR

10 a. Write a short note on :

i) Dependency injection

ii) Instance pooling

iii) Transactions

iv) Security. **(10 Marks)**

b. With a neat diagram, explain the life cycle of Entity bean. **(10 Marks)**

* * * * *

1.a. Explain briefly servlet architecture and life cycle method.

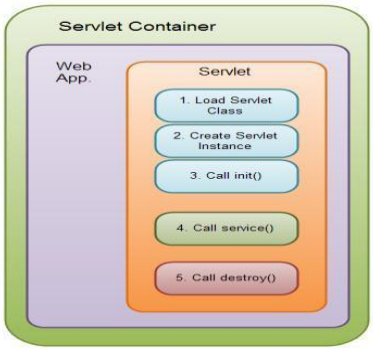
Java Servlets are programs that run on a Web or Application server

 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. Write a java servlet program which reads 2 parameters from the web page which are of type integers and finds the sum of 2 values and return back the result as a web page.

```
package j2ee.prg1;
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class program1
 */
@WebServlet("/pro1")
public class program1 extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public program1() {
        super();
        // TODO Auto-generated constructor stub
    }
	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
```

```
        protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
                // TODO Auto-generated method stub
                // Setting the HTTP Content-Type response header to text/html
                        resp.setContentType("text/html");
                          // Returns a PrintWriter object that can send character text
                    //to the client.
                        PrintWriter pw=resp.getWriter();
                //To retrieve the input values (num1) from HTML page
                        int n1=Integer.parseInt(req.getParameter("num1"));
                       // To retrieve the input values (num2) from HTML page
                        int n2=Integer.parseInt(req.getParameter("num2"));
                        //writing the output in the html format
                        pw.println("<html><body>");
                        pw.println("Sum is "+(n1+n2)+"<br>");
                        pw.println("</body></html>");
        }
}
```

index.html

```
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to url mapping “pro1“ and the post method is used  -->
<form method="post"  action="pro1">
Number1: <input type="text" name="num1"><br>
Number2: <input type="text" name="num2"><br>
<input type="Submit" value="Accept">
</form>
</body>
</html>
```

1.c. Write any four differences between get and post method.

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.

2.a. Explain briefly any five methods of HTTPServletRequest. Illustrate with a simple program.

There are following methods which can be used to read HTTP header in your servlet program. These methods are available with *HttpServletRequest* object.

· **getCookies**

The getCookies method returns the contents of the Cookie header, parsed and stored in an array of Cookie objects.

• **getAuthType and getRemoteUser**

The getAuthType and getRemoteUser methods break the Authorization header into its component pieces.

• **getContentLength**

The getContentLength method returns the value of the Content-Length header (as an int). **getContentType**

The getContentType method returns the value of the Content-Type header (as a String). • **getDateHeader and getIntHeader**

The getDateHeader and getIntHeader methods read the specified header and then convert them to Date and int values, respectively.

• **getHeaderNames**

Rather than looking up one particular header, you can use the getHeaderNames method to get an Enumeration of all header names received on this particular request. • **getHeaders**

In most cases, each header name appears only once in the request. Occasionally, however, a header can appear multiple times, with each occurrence listing a separate value. • **getMethod**

The getMethod method returns the main request method (normally GET or POST, but things like HEAD, PUT, and DELETE are possible).

• **getRequestURI**

The getRequestURI method returns the part of the URL that comes after the host and port but before the form data. For example, for a URL of
http://randomhost.com/servlet/search.BookSearch,
getRequestURI would return /servlet/search.BookSearch.

• **getProtocol**

Lastly, the getProtocol method returns the third part of the request line, which is generally HTTP/1.0 or HTTP/1.1.

2.b. Write a java program using cookies to remember user preferences.

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>
```

3.a. Explain the following tags with an example.
i)Declaration ii) Scriptlet iii)Expression iv) Comment
Three types of tags:
Scriplet tag
Expression tag
Declaration tag

Scriplet Tag:
In JSP JAVA code can be written inside the JSP page using Scriplet tag
Syntax:

```
<% java source code %>
```

Example:

```
<html>
    <body>
      <%  out.print("Hello world…");%>
    </body>
</html>
```

Expression Tag:

Code placed within expression tag is written to the output stream of the response. So, no need to write out.print() to write data. It is mainly used to print values of variable or method

Syntax:

<%= *Statement* %>

Example:

```
<html>
    <body>
      <%= "Hello world…" %>
    </body>
</html>
```

<% = new java.util.Date() %>

Note: Do not end statement with semicolon (;)

Declaration tag:

Used to declare fields and methods. The code written inside this tag is placed outside the service() method of auto generated servlet .So it doesn't get memory at each request

Syntax:

<%! *Statement* %>

Example:

```
<html>
    <body>
      <%!  int data=60;%>
      <%= "Value is: " + data %>
    </body>
</html>
```

Jsp Comments [Creating Template Text]:

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.

There are a special constructs you can use in various cases to insert comments. They are:

Syntax	Purpose
<%-- comment **--%>**	A JSP comment. Ignored by the JSP engine. So comment **appear** in the **JSP page** but *not* in the **resultant document**
<\% - - - - - \%>	Represents static <% - - - %> literal. You want to have [**<%** --- **%>**] in the output then, you need to put <\% -----\%>.
<!-- HTML Comment -->	HTML comments are passed through to the client.

3.b. With a neat diagram explain JSP architecture and life cycle phases of JSP.

User sends request through internet via web browser to Web server for a JSP page (extension .jsp).

As per the request, the Web server (here after called as JSP container) loads the page.

Then, loaded jsp page will be moved to the JSP Servlet Engine.

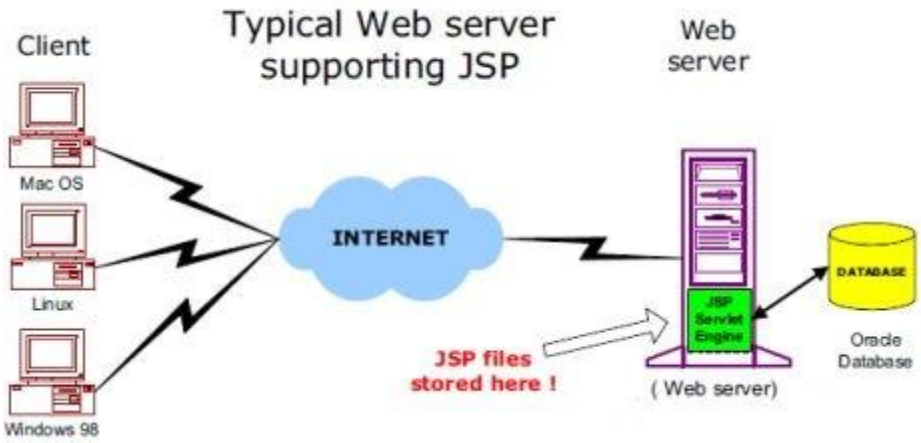
JSP Servlet Engine have two phase to handle the requested JSP page, they are Translation Phase Compilation or Request processing Phase

(I) Translation Phase: JSP Servlet Engine translates the JSP file into Servlet source code file with extension .java. This translation is normally done the first time the page is requested.

(II) Compilation Phase: After translated JSP Servlet Engine compiles the .java file of servlet by compiler and gets converted into .class file.

5. The .class file of Servlet is executed and all the processes that happens in servlet is performed on JSP later like initialization, output of execution is sent to the client as response.

6. In the end, a JSP is just a Servlet.



4.a. Explain briefly any five JSP implicit objects.

request: Reference to the current request
response: Response to the request
session: session associated with current request
application: Servlet context to which a page belongs
pageContext: Object to access request, response, session and application associated with a page
config: Servlet configuration for the page
out: Object that writes to the response output stream
page: instance of the page implementation class (this)
exception: Available with JSP pages which are error pages

4.b. Explain briefly three life cycle methods of JSP.

A JSP life cycle can be defined as the entire process from its creation till the destruction similar to a servlet life cycle with an additional step of compiling a JSP into servlet.
The following are the paths followed by a JSP
Compilation – 3 steps
Parsing jsp
Turning the JSP into a servlet
Compiling the servlet
Initialization
Execution
Destroy
JSP Initialization:

- When a container loads a JSP it invokes the jspInit() method before servicing any requests.
- If you need to perform JSP-specific initialization, override the jspInit() method:

```
public void jspInit()
{
        // Initialization code...
 }
```

- initialization is performed only once
- generally initialize database connections, open files, and create lookup tables in this method.

JSP Execution:

- Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP.

```
void _jspService(HttpServletRequest request, HttpServletResponse response)
{
        // Service handling code...
}
```

- This method is invoked once per request and is responsible for generating the response for that request

JSP Cleanup:

- The destruction phase when a JSP is being removed from use by a container.

- The jspDestroy() method is the JSP equivalent of the destroy method for servlets.
- Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

```
public void jspDestroy()
{
        // Your cleanup code
 }
```

4.c. Write a JSP program to perform arithmetic operation using scriptlet, declaration and expression tag.

```
<HTML>
  <HEAD>
    <TITLE>Addition and Subtraction</TITLE>
  </HEAD>

  <BODY>
    <H1>Addition and Subtraction</H1>
    <%!
        int operand1 = 15, operand2 = 24, sum, difference;
!>
     <% ou.print(“sum:”); %>
        <%= operand1 + operand2 %>
     <% ou.print(“Difference:”); %>
        <%= (operand1 - operand2) %>
<%
        prod = operand1 - operand2;
        quo= operand1/operand2;
        out.println(operand1 + " + " + operand2 + " = " + prod + "<BR>");
        out.println(operand1 + " - " + operand2 + " = " + quo);
     %>
  </BODY>
</HTML>
```

5.a. Explain briefly any 10 attributes of JSP page directive tag.

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

<%@ page attribute="value" %>

Attributes of JSP page directive

import
contentType
extends
info
buffer
language
isELIgnored
isThreadSafe
autoFlush
session
pageEncoding
errorPage

isErrorPage

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.

Example of import attribute

```
<html>
<body>

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

</body>
</html>
```

2)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>
```

3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

4)info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

Example of info attribute

```
<html>
<body>

<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

The web container will create a method getServletInfo() in the resulting servlet.For example:

```
public String getServletInfo() {
  return "composed by Sonoo Jaiswal";
}
```

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)language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

7)isELIgnored

We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later.

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

8)isThreadSafe

Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like:

```
<%@ page isThreadSafe="false" %>
```

The web container in such a case, will generate the servlet as:

```
public class SimplePage_jsp extends HttpJspBase
  implements SingleThreadModel{
.......
}
```

9)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>
```

10)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>
```

5.b. Write a JSP program which uses <jsp:include> and <jsp:forward> standard action to display a web page.

index.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"))
{
%>

<%
}
else
{
<jsp:forward page="main.jsp"></jsp:forward>
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>
```

6.a. Explain the following standard action with suitable example.
a) <jsp:usebean> b) <jsp:plugin>
a)<jsp:usebean>
To instantiate a Bean Class
If bean object of Bean class is already created , it does not create the bean
If object of Bean is not created ,it instantiates the Bean
Syntax:

```
<jsp:useBean id="instancename"
scope="page | request | session |application" class="packagename.classname" type="packagename.classname"
beanName="packagename.classname | <%=expression %>" />
```

Attributes	Description
Id	**To identify the Bean in specified scope**
scope	***Default scope is page*** 1. ***page*- specifies bean can be used within JSP page** 2. ***request* – specifies bean can be used from any JSP page that processes the same request. Wider scope than Page** 3. ***session* - specifies bean can be used from any JSP page in the same session whether processes the same request or not. Wider scope than request** 4. ***application* - specifies bean can be used from any JSP page in the same application. Wider scope than session**
class	**Instantiates the specified bean class (i.e. creates an object of the bean class)**
type	**Provides the bean data type if the bean already exists in the scope.Mainly used with class or beanName attribute other wise no bean is instantiated**

beanName	Instantiates the bean using the java.beans.Beans.instantiate() method

<jsp:plugin>
<jsp:plugin> is used to include components like applet.

2. Attributes of JSP plugin action

Attribute name	Required	Description
type	Yes	Specifies type of the component: applet or bean.
code	Yes	Class name of the applet or JavaBean, in the form of *packagename.classname*.class.
codebase	Yes	Base URL that contains class files of the component.
align	Optional	Alignment of the component. Possible values are: left, right, top, middle, bottom.
archive	Optional	Specifies a list of JAR files which contain classes and resources required by the component.
height, width	Optional	Specifies height and width of the component in pixels.
hspace, vspace	Optional	Specifies horizontal and vertical space between the component and the surrounding components, in pixels.
jreversion	Optional	Specifies JRE version which is required by the component. Default is 1.2.
name	Optional	Name of the component.
title	Optional	Title of the component.
nspluginurl, iepluginurl	Optional	Specifies URL where JRE plugin can be downloaded for Netscape or IE browser, respectively.
mayscript	Optional	Accepts true/false value. If true, the component can access scripting objects (Javascript) of the web page.

6.b. Write a Java JSP program to get student information through a HTML and create JavaBean class, populate Bean and display the same information through an other 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>
```

7.a. Explain briefly any 5 built-in annotations with a suitable example.

Built in Annotations:

Java defines many built-in annotations.

These four are the annotations imported from **java.lang.annotation**: **@Retention**, **@Documented**, **@Target**,and **@Inherited**.

@Override, **@Deprecated**, and **@SuppressWarnings** are included in **java.lang**.

@Retention

@Retention is designed to be used only as an annotation to another annotation. It specifies the retention policy.

@Documented

The **@Documented** annotation is a marker interface that tells a tool that an annotation is tobe documented. It is designed to be used only as an annotation to an annotation declaration.**@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** takesone argument, which must be a constant from the **ElementType** enumeration. This argumentspecifies 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 Con structor
FIELD Field
LOCAL_VARIABLE L ocal variable
METHOD M ethod
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 })

@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.

@Override

@Override is a marker annotation that can be used only on methods. A method annotatedwith **@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.

@Deprecated

@Deprecated is a marker annotation. It indicates that a declaration is obsolete and has beenreplaced by a newer form.

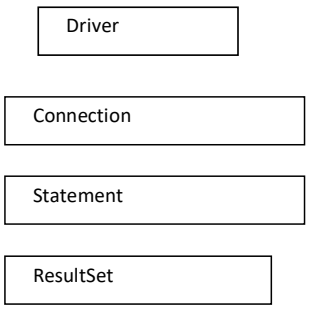
@SuppressWarnings

@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.

7.b. Explain briefly JDBC routine process. Give an example.

1. Seven Basic Steps in Using JDBC

1. Load the Driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement Object
5. Execute a query
6. Process the results
7. Close the Connection



1. Load the JDBC driver

When a driver class is first loaded, it registers itself with the driver Manager Therefore, toregister a driver, just load it!
Example:

```
String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
    Class.forName(driver);                    Or
    Class.forName(sun.jdbc.odbc.JdbcOdbcDriver);
```

2. Define the Connection URL

The purpose of loading and registering the JDBC driver is to bring the JDBC driver into theJava Virtual Machine (JVM).

jdbc : subprotocol : source

- each driver has its own subprotocol

each subprotocol has its own syntax for the source

jdbc:msql://host[:port]/database

Ex:
jdbc:msql://foo.nowhere.com:4333/accou nting

3. Establish the Connection

- DriverManager Connects to given JDBC URL with given **user name** and **password**
- **Throws** java.sql.SQLException
- **returns** a Connection object
- A Connection represents a session with a specific database.
- The connection to the database is established by **getConnection()**, which requests accessto the database from the DBMS.
- A Connection object is returned by the getConnection() if access is granted; else getConnection() throws a SQLException.
- If username & password is required then those information need to be supplied to accessthe database.

  String url = jdbc : odbc : Employee;

  Connection c = DriverManager.getConnection(url,userID,password);

- Sometimes a DBMS requires extra information besides userID & password to grantaccess to the database.
- This additional information is referred as properties and must be associated with Properties or Sometimes DBMS grants access to a database to anyone without using username or password.

  Ex: Connection c = DriverManager.getConnection(url) ;

---

4. Create a Statement Object

A Statement object is used for executing a static SQL statement and obtaining the resultsproduced by it.

Statement stmt = con.createStatement();

This statement creates a Statement object, *stmt* that can pass SQL statements to the DBMSusing connection, *con*.

5. Execute a query

Execute a SQL query such as SELECT, INSERT, DELETE, UPDATE Example

String SelectStudent= "select * from STUDENT";

6. Process the results

- A ResultSet provides access to a table of data generated by executing a Statement.
- Only one ResultSet per Statement can be open at once.
- The table rows are retrieved in sequence.
- A ResultSet maintains a cursor pointing to its current row of data.
- The 'next' method moves the cursor to the next row.

7. Close the Connection

connection.close();

- Since opening a connection is expensive, postpone this step if additional databaseoperations are expected

8.a. Discuss any 5 advanced JDBC 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 programminglanguage.
- 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 isdesired.

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 programminglanguage.
- A Clob object logically points to the CLOB value on the server rather than containing its characterdata, 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 programminglanguage.
- An Array object logically points to an ARRAY value on the server rather than containing theelements of the ARRAY object, which can greatly increase efficiency.
- The Array interface contains methods for materializing the elements of the ARRAY object on theclient 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, anda 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 ina 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 ofone 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 ina 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 theREF persistently and uniquely identifies.
- In the Java programming language, the interface Ref represents an SQL REF.

7. JAVA_OBJECT

- The JDBC type JAVA_OBJECT, makes it easier to use objects in the Java programming language asvalues in a database.
- JAVA_OBJECT is simply a type code for an instance of a class defined in the Java programming language that is stored as a database object.
- The JAVA_OBJECT value may be stored as a serialized Java object, or it may be stored in some vendor-specific format.
- The type JAVA_OBJECT is one of the possible values for the column DATA_TYPE in the ResultSet objects returned by various DatabaseMetaData methods, including getTypeInfo, getColumns, and getUDTs.
- Values of type JAVA_OBJECT are stored in a database table using the method PreparedStatement.setObject.
- They are retrieved with They are retrived with the methods ResultSet.getObject or CallableStatement.getObject and updated with the ResultSet.updateObject method.

For example, assuming that instances of the class Engineer are stored in the column ENGINEERS in the table PERSONNEL, the following code fragment, in which stmt is a Statement object, prints out the names of all of the engineers.

8.b. Discuss the types of JDBC statements with an example.

- 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.

Example-1:

```
Statement stmt = con.createStatement();
ResultSet res = stmt.executeQuery("select * from Employee");
```

OR

Example-2:

Example-3:

Statement stmt = con.createStatement();
stmt.**executeUpdate**(“Insert into employee values(„12345“,“sk“,98453));
stmt.**executeUpdate**(“update employee set Mobile=89706 where Mobile=12345);
OR

Statement stmt = con.createStatement();
stmt.**execute**(“Drop table Employee”);
stmt.**execute**(“Create table Employee (name varhcar(10), age Number(3))”);

```
import java.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'");
        System.out.println(rs.getString(1));
      }
    } // end of try
    catch(Exception e){ System.out.println("exception" + e); }
```

replace

by any of the

1.6. PreparedStatement Object

- 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));
```

1.6. CallableStatement

- 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.

```
Connection
con; try{
String query = "{CALL
LastOrderNumber(?))}"; CallableStatement
stat = con.prepareCall(query);
stat.registerOutParameter( 1

,Types.VARCHAR); stat.execute();
String lastOrderNumber =
stat.getString(1); stat.close();

}
```

9.a. With a neat diagram explain the life cycle of stateful session bean.

Below Figure illustrates the stages that a session bean passes through during its lifetime. The client initiates the life cycle by invoking the create method.The EJB container instantiates the bean and then invokes the setSessionContext and ejbCreate methods in the session bean. The bean is now ready to have its business methods invoked.

While in the ready stage, the EJB container may decide to deactivate, or passivate, the bean by moving it from memory to secondary storage. (Typically, the EJB container uses a least-recently-used algorithm to select a bean for passivation.) The EJB container invokes the bean's ejbPassivate method immediately before passivating it. If a client invokes a business method on the bean while it is in the passive stage, the EJB container activates the bean, moving it back to the ready stage, and then calls the bean's ejbActivate method.

At the end of the life cycle, the client invokes the remove method and the EJB container calls the bean's ejbRemove method. The bean's instance is ready for garbage collection.

Your code controls the invocation of only two life cycle methods-the create and remove methods in the client. All other methods in Figure are invoked by the EJB container. The ejbCreate method, for example, is inside the bean class, allowing you to perform certain operations right after the bean is instantiated. For instance, you may wish to connect to a database in the ejbCreate method.

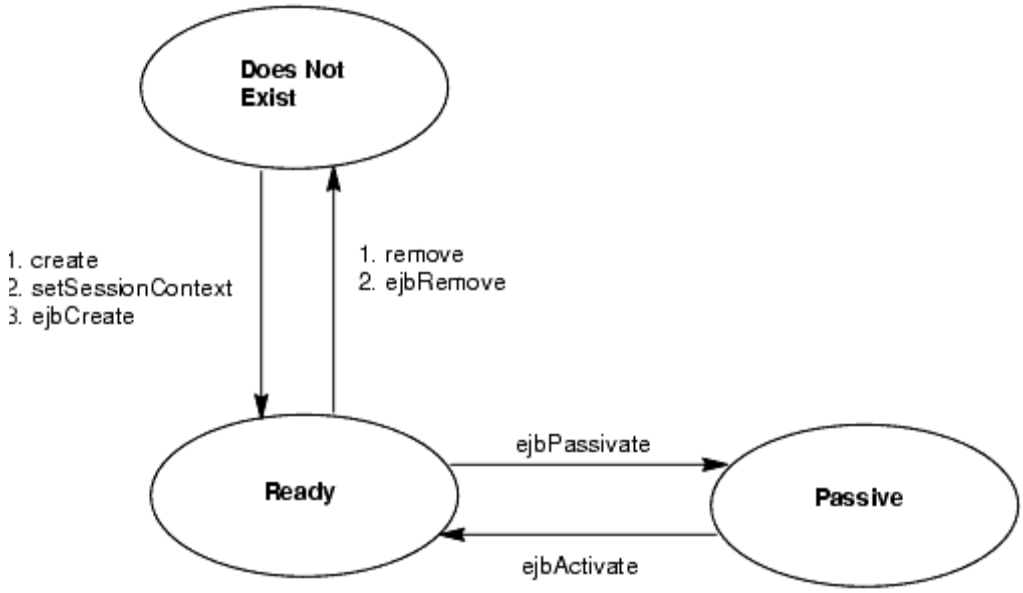


Figure Life Cycle of a Stateful Session Bean

9.b. Explain briefly stateless, stateful and singleton session bean.

There are 3 types of session bean.

1) Stateless Session Bean: It doesn't maintain state of a client between multiple method calls.

2) Stateful Session Bean: It maintains state of a client across multiple requests.

3) Singleton Session Bean: One instance per application, it is shared between clients and supports concurrent access.

Stateless session beans (SLSBs) Stateless session beans are useful for functions in which state does not need to be carried from invocation to invocation. The Container will often create and destroy instances. This allows the Container to hold a much smaller number of objects in service, hence keeping memory footprint down.

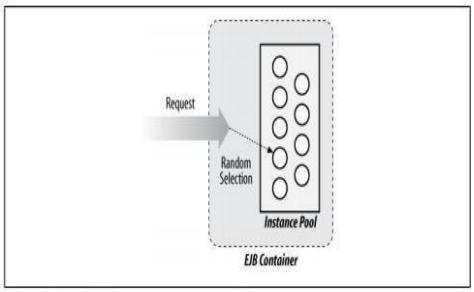


Figure 2-2. An SLSB Instance Selector picking an instance at random

Stateful session beans (SFSBs) Stateful session beans differ from SLSBs in that every request upon a given proxy reference is guaranteed to ultimately invoke upon the same bean instance. SFSB invocations share conversational state. 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 (Figure 2-3). 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.

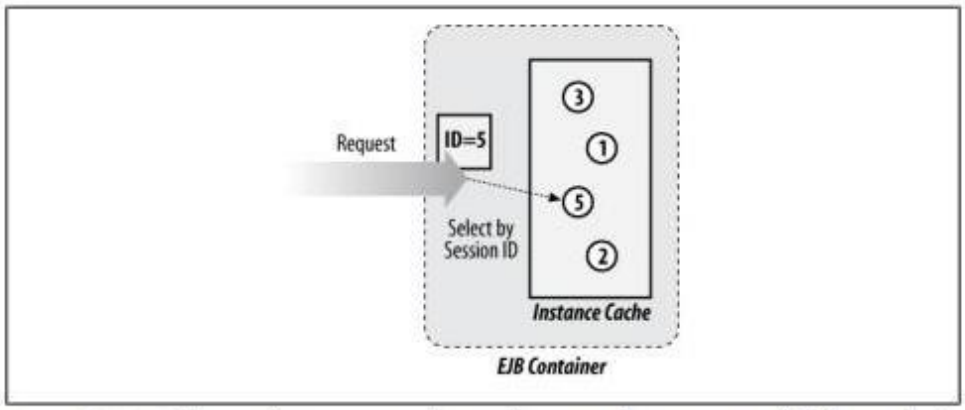


Figure 2-3. Stateful session bean creating and using the correct client instance, which lives inside the EJB Container, to carry out the invocation

Singleton beans Sometimes we don't need any more than one backing instance for our business objects. All requests upon a singleton are destined for the same bean instance, The Container doesn't have much work to do in choosing the target (Figure 2-4). The singleton session bean may be marked to eagerly load when an application is deployed; therefore, it may be leveraged to fire

application lifecycle events. This draws a relationship where deploying a singleton bean implicitlyleads to the invocation of its lifecycle callbacks. We'll put this to good use when we discuss singleton beans.

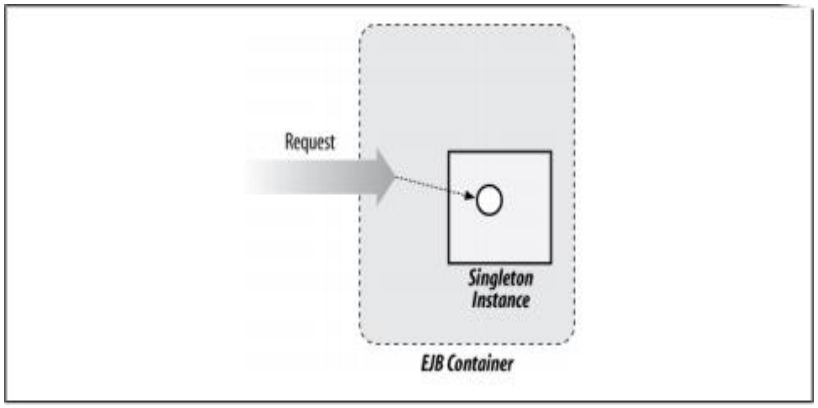


Figure 2-4. Conceptual diagram of a singleton session bean with only one backing bean instance

9.c. List out the differences between stateless and stateful session bean.

Stateless:

1) Stateless session bean maintains across method and transaction

2) The EJB server transparently reuses instances of the Bean to service different clients at the per-method level (access to the session bean is serialized and is 1 client per session bean per method.

3) Used mainly to provide a pool of beans to handle frequent but brief requests. The EJB server transparently reuses instances of the bean to service different clients.

4) Do not retain client information from one method invocation to the next. So many require the client to maintain client side which can mean more complex client code.

5) Client passes needed information as parameters to the business methods.

6) Performance can be improved due to fewer connections across the network.

Stateful:

1) A stateful session bean holds the client session's state.

2) A stateful session bean is an extension of the client that creates it.

3) Its fields contain a conversational state on behalf of the session object's client. This state describes the conversation represented by a specific client/session object pair.

4) Its lifetime is controlled by the client.

5) Cannot be shared between clients.

8.

10.a. Write a short note on

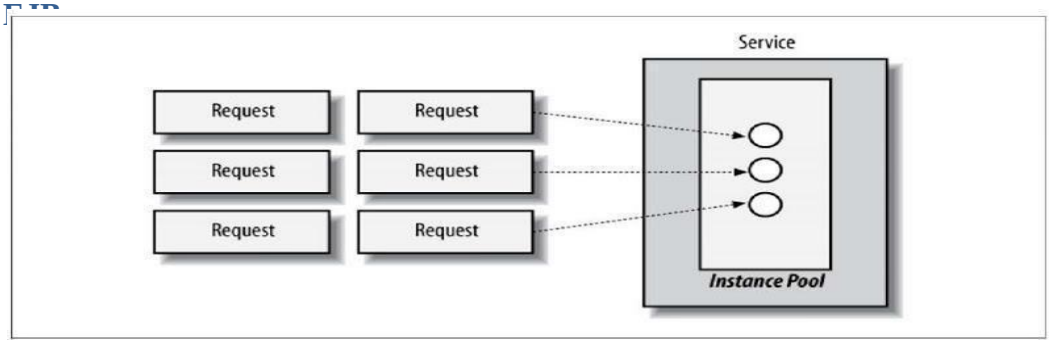
1) Dependency Injection

2)Instance Pooling
3) Transaction
4) Security

1. Instance Pooling/Caching

Because of the strict concurrency rules enforced by the Container, an intentional bottleneck is often introduced where a service instance may not be available for processing until some other request has completed.

If the service was restricted to a singular instance, all subsequent requests would have to queueup until their turn was reached



addresses this problem through a technique called instance pooling, in which each odule is allocated some number of instances with which to serve incoming requests Many vendors provide configuration options to allocate pool sizes appropriate to the work beingperformed, providing the compromise needed to achieve optimal throughput.

Transactions

2.

Transactions provide a means for the developer to easily delegate the creation and control oftransactions to the container.

When a bean calls createTimer(), the operation is performed in the scope of the current transaction. If the transaction rolls back, the timer is undone and it's not created

The timeout callback method on beans should have a transaction attribute of RequiresNew.This ensures that the work performed by the callback method is in the scope of container- initiated transactions.

3.Security

Most enterprise applications are designed to serve a large number of clients, and users arenot necessarily equal in terms of their access rights.

An administrator might require hooks into the configuration of the system, whereas unknownguests may be allowed a read-only view of data.
If we group users into categories with defined roles, we can then allow or restrict access to therole itself, as illustrated in Figure 15-1.

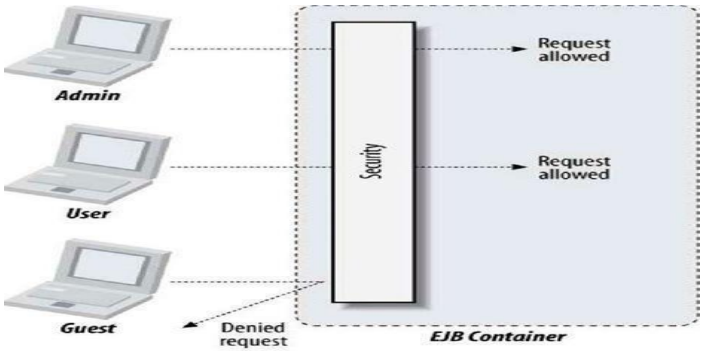


Figure 15-1. EJB security permitting access based upon the caller's role

This allows the application developer to explicitly allow or deny access at a fine- grained level based upon the caller's identity

10.b. With a neat diagram explain the life cycle of Entity Bean.

The life cycle of an entity bean is controlled by the EJB container, not by your application. However, you may find it helpful to learn about the life cycle when deciding in which method your entity bean will connect to a database.

Fig shows the stages that an entity bean passes through during its lifetime. After the EJB container creates the instance, it calls the setEntityContext method of the entity bean class. The setEntityContext method passes the entity context to the bean.

After instantiation, the entity bean moves to a pool of available instances. While in the pooled stage, the instance is not associated with any particular EJB object identity. All instances in the pool are identical. The EJB container assigns an identity to an instance when moving it to the ready stage.

There are two paths from the pooled stage to the ready stage. On the first path, the client invokes the create method, causing the EJB container to call the ejbCreate and ejbPostCreate methods. On the second path, the EJB container invokes the ejbActivate method. While in the ready stage, an entity bean's business methods may be invoked.

There are also two paths from the ready stage to the pooled stage. First, a client may invoke the remove method, which causes the EJB container to call the ejbRemove method. Second, the EJB container may invoke the ejbPassivate method.

At the end of the life cycle, the EJB container removes the instance from the pool and invokes the unsetEntityContext method.

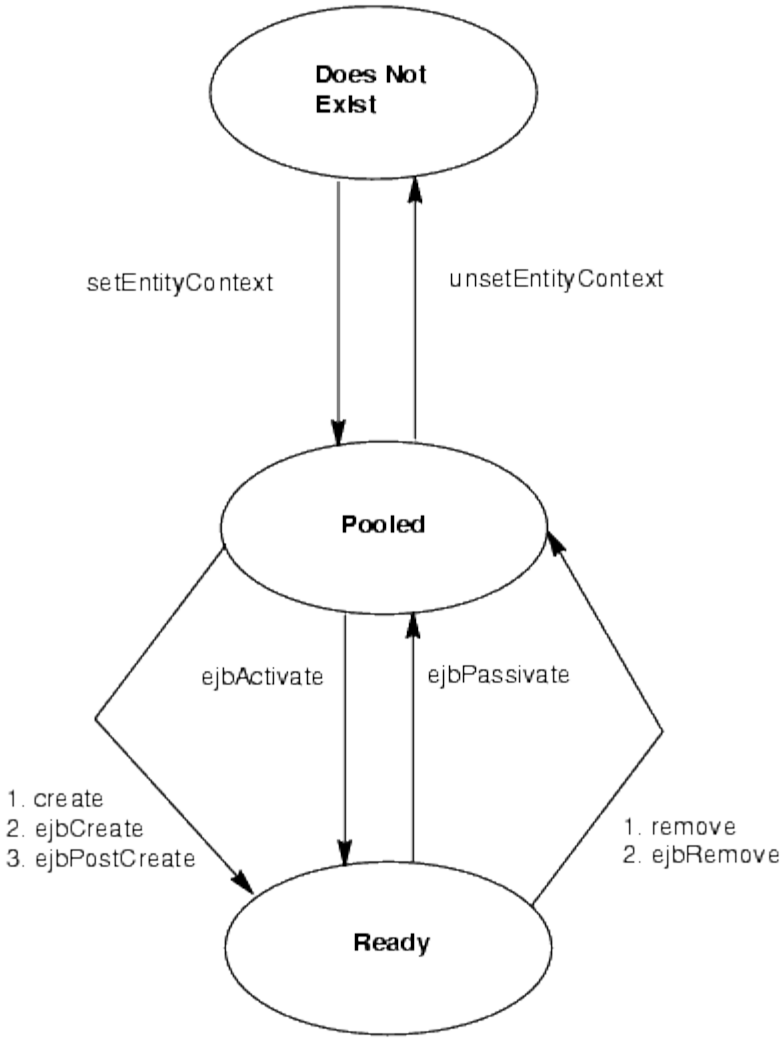


Figure 12 Life Cycle of an Entity Bean

In the pooled state, an instance is not associated with any particular EJB object identity. With bean-managed persistence, when the EJB container moves an instance from the pooled state to the ready state, it does not automatically set the primary key. Therefore, the ejbCreate and ejbActivate methods must set the primary key. If the primary key is incorrect, the ejbLoad and ejbStore methods cannot synchronize the instance variables with the database.

In the pooled state, the values of the instance variables are not needed. You can make these instance variables eligible for garbage collection by setting them to null in the ejbPasssivate method.