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

13MCA42

Fourth Semester MCA Degree Examination, June/July 2017
Advanced Java Programming

Time: 3 hrs. Max. Marks:100

Note: *Answer any FIVE full questions.*

1 a. List out differences between GET method and POST method with respect to servelets. **(03 Marks)**

b. Explain the complete structure of servlet life cycle. **(07 Marks)**

c. Narrate the major range of http status codes along with their purpose. **(05 Marks)**

d. Write a java servlet program to demonstrate auto web page refresh by displaying current date and time. **(05 Marks)**

2 a. What are the need, benefits and advantages of JSP? **(06 Marks)**

b. With an example, describe the various tags available in JSP. **(08 Marks)**

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

3 a. Explain the following page directive attributes along with an example program:
(i) Import
(ii) errorPage and isErrorPage
(iii) ContentType
(iv) buffer and autoflush **(10 Marks)**

b. Write a JSP program to read data from a HTML form (Gender data from radio buttons and colours data from check boxes) and display. **(10 Marks)**

4 a. What is a package? With an example, explain usage of sub packages. **(06 Marks)**

b. Differentiate an interface with an abstract class. **(06 Marks)**

c. How are jar files created and used? Explain its advantages. **(08 Marks)**

5 a. Discuss built-in annotations with an example program. **(09 Marks)**

b. What is a manifest file? Mention its importance **(04 Marks)**

c. Write a Java JSP program to create Java bean from a HTML form data and display it, in a JSP page. **(07 Marks)**

6 a. With an example program, describe the steps involved in connecting a Java program to a database. Perform insert, delete and select operations. **(10 Marks)**

b. What are prepared statement object and callable statement object? When do we use them? Explain with an example code. **(10 Marks)**

7 a. List the differences between stateless session bean and stateful session bean. **(06 Marks)**

b. Explain any four annotations used in EJB along with their meaning. **(04 Marks)**

c. Demonstrate a program to implement an Entity bean. **(10 Marks)**

8 Write a short notes on:

a. Single thread model interface.

b. Architecture of JSP.

c. getName ( ) and getValue ( ).

d. @Documented and @Target annotations. **(20 Marks)**

* * * * *

1. **a. List out differences between GET method and POST method with respect to servlets.(03 Marks)**

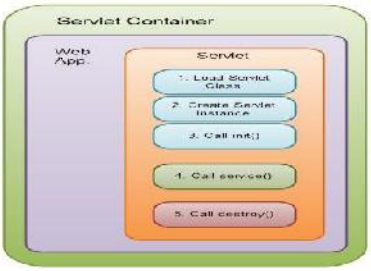
Difference Type	GET (doGet())	POST (doPost())
HTTP Request	The request contains only the request line and HTTP header.	Along with request line and header it also contains HTTP body.
URL Pattern	Query string or form data is simply appended to the URL as name-value pairs.	Form name-value pairs are sent in the body of the request, not in the URL itself.
Parameter passing	The form elements are passed to the server by appending at the end of the URL.	The form elements are passed in the body of the HTTP request.
Size	The parameter data is limited (the limit depends on the container normally 4kb)	Can send huge amount of data to the server.
Idempotency	GET is Idempotent(can be applied multiple times without changing the result)	POST is not idempotent(warns if applied multiple times without changing the result)
Usage	Generally used to fetch some information from the host.	Generally used to process the sent data.

b. Explain the complete structure of servlet life cycle. (07 Marks)

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...
}
```

c. Narrate the major range of Http status codes along with their purpose. (05 Marks)

The status line consists of the

- HTTP version (HTTP/1.1 in the example above),
- a status code (an integer; 200 in the above example),
- a very short message corresponding to the status code (OK in the example).

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.

d. Write a java servlet program to demonstrate auto web page refresh by displaying current date and time. (05 Marks)

```
package j2ee.prg2;
import java.io.*;
import java.util.Date;
import javax.servlet.ServletException;
```

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.*;
/**
 * Servlet implementation class program2
 */
@WebServlet("/program2")
public class program2 extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public program2() {
        super();
        // TODO Auto-generated constructor stub
    }

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

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

	private void performTask(HttpServletRequest request,HttpServletResponse
response)throws ServletException,IOException
	{
		// Setting the HTTP Content-Type response header to text/html
		response.setContentType("text/html");
		//Adds a response header to refresh the webpage in every one second.
		response.addHeader("Refresh","1");
		//	Returns a PrintWriter object that can send character text to the client.
		PrintWriter out=response.getWriter();
```

```
            //writing the output in the html format
            out.println("Text servlet says hi at "+new Date());
        }

}
```

2. a. What are the need, benefits and advantages of JSP? (06 Marks)

Need:

• **It is hard to write and maintain the HTML**. Using print statements to generate HTML? Hardly convenient: you have to use parentheses and semicolons, have to insert backslashes in front of embedded double quotes, and have to use string concatenation to put the content together. Besides, it simply does not look like HTML, so it is harder to visualize. Compare this servlet style with Listing 10.1 where you hardly even notice that the page is not an ordinary HTML document.

• **You cannot use standard HTML tools.** All those great Web-site development tools you have are of little use when you are writing Java code.

• **The HTML is inaccessible to non-Java developers**. If the HTML is embedded within Java code, a Web development expert who does not know the Java programming language will have trouble reviewing and changing the HTML.

Benefits:

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.

Advantages:

Following table lists out the other advantages of using JSP over other technologies −

vs. Active Server Pages (ASP)

The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

vs. Pure Servlets

It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

vs. Server-Side Includes (SSI)

SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
vs. JavaScript
JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
vs. Static HTML
Regular HTML, of course, cannot contain dynamic information.

b. With an example, describe the various tags available in JSP. (08 Marks)
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" />
```

Attributes:
Following is the list of attributes associated with page directive:

Attribute	Purpose
buffer	Specifies a buffering model for the output stream.
autoFlush	Controls the behavior of the servlet output buffer.
contentType	Defines the character encoding scheme.
errorPage	Defines the URL of another JSP that reports on Java unchecked runtime exceptions.

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

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!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>Applet Index</title>
</head>
<body>
<jsp:plugin type="applet" code="applet11.class" codebase=" " width="400" height="400">
<jsp:fallback><p>unable to load applet</p>p>
</jsp:fallback>
</jsp:plugin>
</body>
</html>
```

Applet11.class

```
package com;
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class applet11  extends Applet
 {
publicvoid paint(Graphics g)
 {
        setBackground(Color.pink);
        setForeground(Color.black);
        g.drawString("welcome jsp-Applet",100,100);
         }
}
```

3. a. Explain the following page directive attributes along with an example program(10 Marks)

i) import

The import attribute of the page directive lets us specify the packages that should be imported by the servlet into which the JSP page gets translated.

By default, the servlet imports java.lang.*, javax.servlet.*, javax.servlet.jsp.*, javax.servlet.http.*, and possibly some number of server-specific entries. Never write JSP code that relies on any server-specific classes being imported automatically; doing so makes your code nonportable. Use of the import attribute takes one of the following form

<% @ page import =”package.classname” %>
Ex: <% @ page import=”java.util.*” %>
ii) errorPage and isErrorPage
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.
//index.jsp
<html>
<body>

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

The isErrorPage attribute is used to declare that the current page is the error page.
<html>
<body>
<%@ page isErrorPage="true" %>
Sorry an exception occured!
The exception is: <%= exception %>
</body>
</html>

iii) Content Type
The contentType attribute sets the Content-Type response header, indicating the MIME type of the document being sent to the client.

```
<%@ page contentType="MIME-Type" %>
<%@ page contentType="MIME-Type; charset=Character-Set" %>
```

Ex:

```
<%@ page contentType="application/vnd.ms-excel" %>
```

iv) buffer and autoflush
The buffer attribute specifies the size of the buffer used by the out variable, which is of type JspWriter. Use of this attribute takes one of two forms:

```
<%@ page buffer="sizekb" %>
<%@ page buffer="none" %>
```

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="true" %> <%-- Default --%>
<%@ page autoFlush="false" %>
```

b. Write a JSP program to read data from a HTML form (Gender data from radio buttons and colours data from checkboxes) and display.
(10 Marks)

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>
<html>
<body>
<form name="frm" method="get" action="radioInput.jsp">
<table width="100%" border="0" cellspacing="0" cellpadding="0">
 <tr>
  <td width="22%"> </td>
  <td width="78%"> </td>
  </tr>
 <tr>
  <td>Male<input type="radio" name="gender" value="Male"></td>
  <td>Female <input type="radio" name="gender" value="Female"></td>
 </tr>

 <tr>
  <td>RED <input type="checkbox" name="colour" value="red"></td>
  <td>GREEN <input type="checkbox" name="colour" value="green"></td>
  <td>BLUE <input type="checkbox" name="colour" value="blue"></td>

 </tr>
 <tr>
<tr>
  <td> </td>
  <td><input type="submit" name="submit" value="Submit"></td>
  </tr>
 <tr>
  <td> </td>
  <td> </td>
  </tr>
</table>
</form>
</body>
</html>

<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>
<%
String radioInJSP=request.getParameter("gender");
%>

%>
<html>
```

```
<body>
Value of radio button box in JSP : <%=radioInJSP%>
<%
String colours[]= request.getParameterValues("colour");
if(colours != null)
{
%>
<h4>I likes colour/s mostly</h4>
<ul>
<%
for(int i=0; i<colours.length; i++)
{
%>
<li><%=colours[i]%></li>
<%
}
%>

</body>
</html>
```

4. a. What is a package? With an example, explain usage of sub packages (06 Marks)

A java package is a group of similar types of classes, interfaces and sub-packages.
Package in java can be categorized in two form, built-in package and user-defined package.
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
Package inside the package is called the subpackage. It should be created to categorize the package further.
Let's take an example, Sun Microsystem has defined a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

```
package importpackage.subpackage;

public class HelloWorld {
  public void show(){
  System.out.println("This is the function of the class
HelloWorld!!");
  }
}
```

Now import the package "subpackage" in the class file "CallPackage" shown as:

```
import importpackage.subpackage.*;
class CallPackage{
  public static void main(String[] args){
  HelloWorld h2=new HelloWorld();
  h2.show();
  }
}
```

b. Differentiate an interface with an abstract class. (06 Marks)

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.

6) Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

c. How are jar files created and used? Explain its advantages. (08 Marks)

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.
The jar tool provides many switches, some of them are as follows:
-c creates new archive file
-v generates verbose output. It displays the included or extracted resource on the standard output.
-m includes manifest information from the given mf file.
-f specifies the archive file name
-x extracts files from the archive file

5. a. Discuss built-in annotations with an example program. (09 Marks)

Java Annotations
Java Annotation is a tag that represents the metadata i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.
Annotations in java are used to provide additional information, so it is an alternative option for XML and java marker interfaces.
First, we will learn some built-in annotations then we will move on creating and using custom annotations.
Built-In Java Annotations
There are several built-in annotations in java. Some annotations are applied to java code and some to other annotations.
Built-In Java Annotations used in java code
@Override
@SuppressWarnings
@Deprecated
Built-In Java Annotations used in other annotations
@Target
@Retention
@Inherited
@Documented
@Override
@Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.
Sometimes, we does the silly mistake such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurity that method is overridden.

```
class Animal{
void eatSomething(){System.out.println("eating something");}
}
```

```
  class Dog extends Animal{
@Override
void eatsomething(){System.out.println("eating foods");}//should be eatSometh
ing
}
  class TestAnnotation1{
public static void main(String args[]){
Animal a=new Dog();
a.eatSomething();
}}
```

Output:

Comple Time Error

@SuppressWarnings

@SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

```
import java.util.*;
class TestAnnotation2{
@SuppressWarnings("unchecked")
public static void main(String args[]){
ArrayList list=new ArrayList();
list.add("sonoo");
list.add("vimal");
list.add("ratan");
 for(Object obj:list)
System.out.println(obj);
 }
}
```

Now no warning at compile time.

If you remove the @SuppressWarnings("unchecked") annotation, it will show warning at compile time because we are using non-generic collection.

@Deprecated

@Deprecated annoation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

```
class A{
void m(){
System.out.println("hello m");}
 @Deprecated
void n(){System.out.println("hello n");}
}
  class TestAnnotation3{
public static void main(String args[]){
  A a=new A();
a.n();
}}
```

At Compile Time:
Note: Test.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
At Runtime:
hello n
@Target
@Target tag is used to specify at which type, the annotation is used.
The java.lang.annotation.ElementType enum declares many constants to specify the type of element where annotation is to be applied such as TYPE, METHOD, FIELD etc.

b. What is a manifest file? Mention its importance. (04 Marks)

The manifest is a special file that can contain information about the files packaged in a JAR file. By tailoring this "meta" information that the manifest contains, you enable the JAR file to serve a variety of purposes.

pplications Bundled as JAR Files: If an application is bundled in a JAR file, the Java Virtual Machine needs to be told what the entry point to the application is. An entry point is any class with a public static void main(String[] args) method. This information is provided in the Main-Class header, which has the general form:

```
Main-Class: classname
```

The value classname is to be replaced with the application's entry point.

Download Extensions: Download extensions are JAR files that are referenced by the manifest files of other JAR files. In a typical situation, an applet will be bundled in a JAR file whose manifest references a JAR file (or several JAR files) that will serve as an extension for the purposes of that applet. Extensions may reference each other in the same way. Download extensions are specified in the Class-Path header field in the manifest file of an applet, application, or another extension. A Class-Path header might look like this, for example:

```
Class-Path: servlet.jar infobus.jar acme/beans.jar
```

With this header, the classes in the files servlet.jar, infobus.jar, and acme/beans.jar will serve as extensions for purposes of the applet or application. The URLs in the Class-Path header are given relative to the URL of the JAR file of the applet or application.

Package Sealing: A package within a JAR file can be optionally sealed, which means that all classes defined in that package must be archived in the same JAR file. A package might be sealed to ensure version consistency among the classes in your software or as a security measure. To seal a package, a Name header needs to be added for the package, followed by a Sealed header, similar to this:

```
Name: myCompany/myPackage/
Sealed: true
```

The Name header's value is the package's relative pathname. Note that it ends with a '/' to distinguish it from a filename. Any headers following a Name header, without any intervening blank lines, apply to the file or package specified in the Name header. In the above example, because the Sealed header occurs after the Name: myCompany/myPackage header, with no blank lines between, the Sealed header will be interpreted as applying (only) to the package myCompany/myPackage.

Package Versioning: The Package Versioning specification defines several manifest headers to hold versioning information. One set of such headers can be assigned to each package. The versioning headers should appear directly beneath the Name header for the package. This example shows all the versioning headers:

Name: java/util/
Specification-Title: "Java Utility Classes"
Specification-Version: "1.2"
Specification-Vendor: "Sun Microsystems, Inc.".
Implementation-Title: "java.util"
Implementation-Version: "build57"
Implementation-Vendor: "Sun Microsystems, Inc."

c. Write a java JSP program to create Java bean from a HTML form data and display it in a JSP page. (07 Marks)

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. With an example program describe the steps involved in connecting a Java program to a database. Perform insert, delete and select operations. (10 Marks)

Seven Basic Steps in Using JDBC

1. Load the Driver
2. Define the Connection UR
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, to register 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

jdbc : subprotocol : source

- each driver has its own subprotocol

each subprotocol has its own syntax for the source

jdbc : odbc : DataSource

Ex: jdbc : odbc : Employee

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

Ex: jdbc:msql://foo.nowhere.com:4333/accounting

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 access to 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 access the database.

String url = jdbc : odbc : Employee;

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

• Sometimes a DBMS requires extra information besides userID & password to grant access 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 results produced by it.

Statement stmt = con.createStatement();

This statement creates a Statement object, stmt that can pass SQL statements to the DBMS using 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 database operations are expected

```
package j2ee.p9;
import java.sql.*;
import java.io.*;
public class Studentdata {
        public static void main(String[] args) {
                Connection con;
                PreparedStatement pstmt;
                Statement stmt;
                ResultSet rs;
                String uname, pword;
                Integer marks,count;
                try
                {
                        Class.forName("com.mysql.jdbc.Driver"); // type1 driver
```

```
                try{

        con=DriverManager.getConnection("jdbc:mysql://127.0.0.1/mca","root","system"); //
type1 access connection
                        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
                    do
                    {


                        System.out.println("\n1. Insert.\n2. Select.\n3. Update.\n4.
Delete.\n5. Exit.\nEnter your choice:");
                        int choice=Integer.parseInt(br.readLine());
                        switch(choice)
                        {
                            case 1: System.out.print("Enter UserName :");
                                uname=br.readLine();
                                System.out.print("Enter Password :");
                                pword=br.readLine();
                                pstmt=con.prepareStatement("insert into student
values(?,?)");
                                pstmt.setString(1,uname);
                                pstmt.setString(2,pword);
                                pstmt.execute();
                                System.out.println("\nRecord Inserted
successfully.");
                            break;
                            case 2:
                                stmt=con.createStatement();
                                rs=stmt.executeQuery("select *from student");
                                if(rs.next())
                                {
                                System.out.println("User Name\tPassword\n-----
--------------------------");
                                do
                                {
                                    uname=rs.getString(1);
                                    pword=rs.getString(2);

                                    System.out.println(uname+"\t"+pword);
                                }while(rs.next());
                                }
                                else
```

```
                                        System.out.println("Record(s) are not
available in database.");
                                break;
                                case 3:
                                        System.out.println("Enter User Name to
update :");
                                        uname=br.readLine();
                                        System.out.println("Enter new password
:");
                                        pword=br.readLine();
                                        stmt=con.createStatement();
                                        count=stmt.executeUpdate("update
student set password='"+pword+"'where username='"+uname+"'");
                                        System.out.println("\n"+count+" Record
Updated.");
                                break;
                                case 4: System.out.println("Enter User Name to
delete record:");
                                        uname=br.readLine();
                                        stmt=con.createStatement();
                                        count=stmt.executeUpdate("delete from
student where username='"+uname+"'");


                                        if(count!=0)
                                                System.out.println("\nRecord
"+uname+" has deleted.");
                                        else
                                                System.out.println("\nInvalid
USN, Try again.");
                                break;

                                case 5: con.close(); System.exit(0);
                                default: System.out.println("Invalid choice, Try
again.");
                        }//close of switch
                        }while(true);
                        }//close of nested try
                        catch(SQLException e2)
                        {
                                System.out.println(e2);
                        }
                        catch(IOException e3)
                        {
```

```
                                        System.out.println(e3);
                                }
                }//close of outer try
                catch(ClassNotFoundException e1)
                {
                        System.out.println(e1);
                }
        }
}
```

b. What are prepared statement object and callable statement object? When do we use them? (10 Marks)

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
```

Callable Statement:

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();
}
catch (Exception e){}
```

7. a. List the differences between stateless session bean and stateful session bean. (06 Marks)

Stateless:

1) Stateless session bean maintains across method and transction
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 reuests. 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 requir the client tomaintain on the 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 extesion 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.

b. Explain any four annotations used in EJB along with their meaning. (04 Marks)

	Name	Description
	javax.ejb.Stateless	Specifies that a given EJB class is a stateless session bean. Attributes name − Used to specify name of the session bean. mappedName − Used to specify the JNDI name of the session bean. description − Used to provide description of the session bean.
	javax.ejb.Stateful	Specifies that a given EJB class is a stateful session bean. Attributes name − Used to specify name of the session bean. mappedName − Used to specify the JNDI name of the session bean. description − Used to provide description of the session bean.
	javax.ejb.Message DrivenBean	Specifies that a given EJB class is a message driven bean. Attributes name − Used to specify name of the message driven bean. messageListenerInterface− Used to specify message listener interface for the message driven bean. activationConfig − Used to specify the configuration details of the message-driven bean in an operational environment of the message driven bean. mappedName − Used to specify the JNDI name of the session bean. description − Used to provide description of the session bean.

	javax.ejb.EJB	Used to specify or inject a dependency as EJB instance into another EJB. Attributes name − Used to specify name, which will be used to locate the referenced bean in the environment. beanInterface − Used to specify the interface type of the referenced bean. beanName − Used to provide name of the referenced bean. mappedName − Used to specify the JNDI name of the referenced bean. description − Used to provide description of the referenced bean.

c. Demonstrate a program to implement an Entity bean. (10 Marks)

```
package Persist;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Student implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String usnno;
   private String name;
   private int mark;

   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }

   @Override
```

```
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not set
    if (!(object instanceof Student)) {
        return false;
    }
    Student other = (Student) object;
    if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
        return false;
    }
    return true;
}
@Override
public String toString() {
    return "Persist.Student[ id=" + id + " ]";
}
public String getUsnno() {
    return usnno;
}

public void setUsnno(String usnno) {
    this.usnno = usnno;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}

public int getMark() {
    return mark;
}

public void setMark(int mark) {
    this.mark = mark;
}
```

```
}
```
StudServlet.java
```
package WebClient;
import Persist.Student;
import Persist.StudentFacadeLocal;
import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class StudServlet extends HttpServlet
{
   @EJB
   private StudentFacadeLocal studentFacade;
     protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      Student obj=new Student();
      obj.setUsnno(request.getParameter("usnno"));
      obj.setName(request.getParameter("name"));
      obj.setMark(Integer.parseInt(request.getParameter("mark")));
      studentFacade.create(obj);

      try (PrintWriter out = response.getWriter()) {
         /* TODO output your page here. You may use following sample code. */
         out.println("<!DOCTYPE html>");
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Student Data</title>");
         out.println("</head>");
         out.println("<body>");
         out.println("<h1>Congrats : Student " + request.getParameter("name") + " Record is
created successfully </h1>");
         out.println("</body>");
         out.println("</html>");
      }
   }
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      processRequest(request, response);
```

```
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

Index.html

```
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
         <form method="get" action="StudServlet">
        Enter USN No. :<input type ="text" name ="usnno"/><br/>
        Enter Name    :<input type ="text" name ="name"/><br/>
        Enter Mark    :<input type ="text" name ="mark"/><br/>
            <input type="submit" value="Submit"/>
    </form>
    </body>
</html>
```

8. Write a short notes on

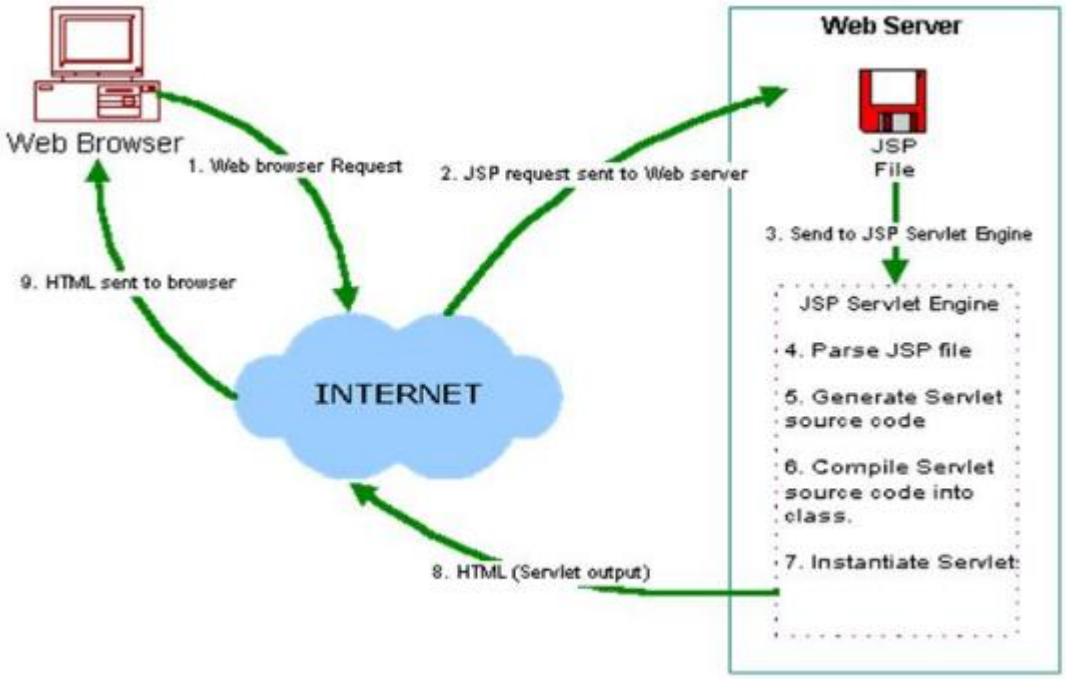
a. Single thread model interface.

Normally, the system makes a single instance of your servlet and then creates a new thread for each user request, with multiple simultaneous threads running if a new request comes in while a previous request is still executing. doGet and doPost methods must be careful to synchronize access to fields and other shared data, since multiple threads may be trying to access the data simultaneously. If you want to prevent this multithreaded access, we can have our servlet implement the SingleThreadModel interface, as below.

```
public class YourServlet extends HttpServlet implements SingleThreadModel { ... }
```

If we implement this interface, the system guarantees that there is never more than one request thread accessing a single instance of your servlet. It does so either by queuing up all the requests and passing them one at a time to a single servlet instance, or by creating a pool of multiple instances, each of which handles one request at a time.

b. Architecture of JSP.



User requesting a JSP page through internet via web browser.
The JSP request is sent to the Web Server.
Web server accepts the requested .jsp file and passes the JSP file to the JSP Servlet Engine.
If the JSP file has been called the first time then the JSP file is parsed otherwise servlet is instantiated.
The next step is to generate a servlet from the JSP file. In that servlet file, all the HTML code is converted in println statements.
Now, The servlet source code file is compiled into a class file (bytecode file).
The servlet is instantiated by calling the init and service methods of the servlet's life cycle.
Now, the generated servlet output is sent via the Internet form web server to user's web browser.
Now in last step, HTML results are displayed on the user's web browser.

c. getName() and getValue()

The getName method retrieves the name of the cookie. However, since the name is supplied to the Cookie constructor, there is no setName method; you cannot change the name once the cookie is created. On the other hand, getName is used on almost every cookie received by the server. Since the getCookies method of HttpServletRequest returns an array of Cookie objects, a common practice is to loop down the array, calling getName until you have a particular name, then to check the value with getValue.
getValue() is used to retrieve the value of the cookie.

d. @Documented and @Target annotations

The @Documented annotation is used to signal to the JavaDoc tool that your custom annotation should be visible in the JavaDoc for classes using your custom annotation. Here is a @Documented Java annotation example:

```
import java.lang.annotation.Documented;
@Documented
public @interface MyAnnotation {
}
@MyAnnotation
public class MySuperClass { ... }
```

@Target

@Target tag is used to specify at which type, the annotation is used.

The java.lang.annotation.ElementType enum declares many constants to specify the type of element where annotation is to be applied such as TYPE, METHOD, FIELD etc.