UNIT 1 SERVLET PROGRAMMING

Structure **Page Nos.**

1.0 INTRODUCTION

We have already learnt about core Java and how to compile and learn the Java programs. In this unit we shall cover the basics of Java Servlet and different interfaces of Servlet. Java Servlets are the small, platform-independent Java programs that runs in a web server or application server and provides server-side processing such as accessing a database and e-commerce transactions. Servlets are widely used for web processing. Servlets are designed to handle HTTP requests (get, post, etc.) and are the standard Java replacement for a variety of other methods, including CGI scripts, Active Server Pages (ASPs). Servlets always run inside a Servlet Container. A Servlet Container is nothing but a web Server, which handles user requests and generates response. Servlet Container is different from Web Server because it is only meant for Servlet and not for other files (like .html etc). In this unit, we shall also see how the Servlet Container is responsible for maintaining lifecycle of the Servlet. Servlets classes should always implement the *javax.servlet.Servlet* interface. This interface contains five methods, which must be implemented. We shall learn how these methods can be used in Servlet programming and how Servlet and JDBC combination has proved a simple elegant way to connect to database.

1.1 OBJECTIVES

After going through this unit, you should be able to know:

- how to install the Servlet Engine / Web Server;
- basics of Servlet and how it is better than other server extensions;
- how Servlet engine maintains the Servlet Life Cycle;
- where do we use HttpServletRequest Interface and some of its basic methods;
- where do we use HttpServletResponse Interface and some of its basic methods;
- what is session tracking;
- different ways to achieve Session Tracking like HttpSession & persistent cookies, and
- different ways to achieve InterServlet communication.

1.2 HOW TO INSTALL SERVLET ENGINE/WEB SERVER

A Servlet is a Java class and therefore needs to be executed in a Java VM by a service we call a Servlet engine. This Servlet engine is mostly contained in Servlet Engine or may be added as a module. Some Web servers, such as Sun's Java Web Server (JWS), W3C’s Jigsaw and Gefion Software’s LiteWebServer (LWS) are implemented in Java and have a built-in Servlet engine. Other Web servers, such as Netscape’s Enterprise Server, Microsoft's Internet Information Server (IIS) and the Apache Group’s Apache, require a Servlet engine add-on module. Examples of servlet engine add-ons are Gefion Software's WAICoolRunner, IBM’s WebSphere, Live Software’s JRun and New Atlanta’s ServletExec.

We will be using Tomcat4.0 for our Servlets. You’ll need to have JDK 1.3 installed on your system in order for Tomcat 4.0 to work. If you don't already have it, you can get it from java.sun.com.

Obtaining Tomcat 4.0

Tomcat 4.0 is an open source and free Servlet Container and JSP Engine. It is developed by Apache Software Foundation’s Jakarta Project and is available for download at http://jakarta.apache.org/tomcat, or more specifically at http://jakarta.apache.org/site/binindex.html.

Choose the latest Tomcat 4.0 version. Once you have downloaded Tomcat 4.0, proceed to the next step.Installing Tomcat 4.0Unzip the file to a suitable directory. In Windows, you can unzip to C:\ which will create a directory like C:\jakarta-tomcat-4.0-b5 containing Tomcat files.Now you'll have to create two environment variables, CATALINA_HOME and JAVA_HOME. Most probably you'll have JAVA_HOME already created if you have installed Java Development Kit on your system. If not then you should create it. The values of these variables will be something like.

```
CATALINA_HOME : C:\jakarta-tomcat-4.0-b5
JAVA_HOME : C:\jdk1.3
```

To create these environment variables in Windows 2000, go to Start -> Settings -> Control Panel -> System -> Advanced -> Environment Variables -> System variables -> New. Enter the name and value for CATALINA_HOME and also for JAVA_HOME if not already there.Under Windows 95/98, you can set these variables by editing C:\autoexec.bat file. Just add the following lines and reboot your system :SET CATALINA_HOME=C:\jakarta-tomcat-4.0-b5SET JAVA_HOME=C:\jdk1.3Now copy C:\jakarta-tomcat-4.0-b5\common\lib\servlet.jar file and replace all other servlet.jar files present on your computer in the %CLASSPATH% with this file. To be on the safe side, you should rename your old servlet.jar file/s to servlet.jar_ so that you can use them again if you get any error or if any need arises. You can search for servlet.jar files on your system by going to Start -> Search -> For Files and Folders -> Search for Files and Folders named and typing servlet.jar in this field. Then every servlet.jar file you find should be replaced by C:\jakarta-tomcat-4.0-b5\common\lib\servlet.jar file.The idea is that there should by ONLY ONE VERSION OF SERVLET.JAR FILE on your system, and that one should be C:\jakarta-tomcat-4.0-b5\common\lib\servlet.jar or you might get errors when trying to run JSP pages with Tomcat.

1.3 YOUR FIRST JAVA SERVLET

Servlets are basically developed for server side applications and designed to handle http requests. The servlet-programming interface (Java Servlet API) is a standard part of the J2EE platform and has the following advantages over other common server extension mechanisms:

- They are faster than other server extensions, like, CGI scripts because they use a different process model.
- They use a standard API that is supported by many Web servers.
- Since Servlets are written in Java, Servlets are portable between servers and operating systems. They have all of the advantages of the Java language, including ease of development.

They can access the large set of APIs available for the Java platform.
Now, after installing the Tomcat Server, let us create our first Java Servlet. Create a new text file and save it as 'HelloWorld.java' in the

'C:\jakarta-tomcat-4.0-b5\webapps\saurabh\WEB-INF\classes\com\stardeveloper\servlets' folder. 'C:\jakarta-tomcat-4.0-b5\webapps\' is the folder where web applications that we create should be kept in order for Tomcat to find them. '\saurabh' is the name of our sample web application and we have created it earlier in Installing Tomcat. '\WEB-INF\classes' is the folder to keep Servlet classes. '\com\stardeveloper\servlets' is the package path to the Servlet class that we will create. Now type the following text into the 'HelloWorld.java' file we created earlier:

```
// Your First Servlet program
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
          res.setContentType("text/html");
         PrintWriter out = res.getWriter();
         out.println("<HTML>");
         out.println("<HEAD><TITLE>Hello, Welcome to the  World of Servlet
         Programming</TITLE></HEAD>");
         out.println("<BODY>");
         out.println("<BIG>Hello World</BIG>");
         out.println("</BODY></HTML>");
}
}
```

Our HelloWorld program is extremely simple and displays Hello World in the Browser. We override method doGet() of HttpServlet class.In the doGet() method we set the content type to 'text/html' (telling the client browser that it is an HTML document). Then get hold of PrintWriter object and using it print our own HTML content to the client. Once done we close() it.

Compiling and Running the Servlet

Now, we will compile this Servlet using the following command at the DOS prompt:

```
C:\jakarta-tomcat-4.0-b5\webapps\star\WEB-
INF\classes\com\stardeveloper\servlets>javac HelloWorld.java
```

If all goes well a 'HelloWorld.class' file will be created in that folder. Now, open your browser and point to the following address:

http://localhost:8080/star/servlet/com.stardeveloper.servlets.HelloWorld

You should see response from the Servlet showing "Helloworld"
I have assumed here that you are running Tomcat at port 8080 (which is the default for Tomcat).

Just like an applet, a servlet does not have a main() method. Instead, certain methods of a servlet are invoked by the server in the process of handling requests. Each time the server dispatches a request to a servlet, it invokes the servlet's service() method. A generic servlet should override its service() method to handle requests as appropriate for the servlet. The service() method accepts two parameters: a request object and a response object. The request object tells the servlet about the request, while the response object is used to return a response. *Figure 1* shows how a generic servlet handles requests.

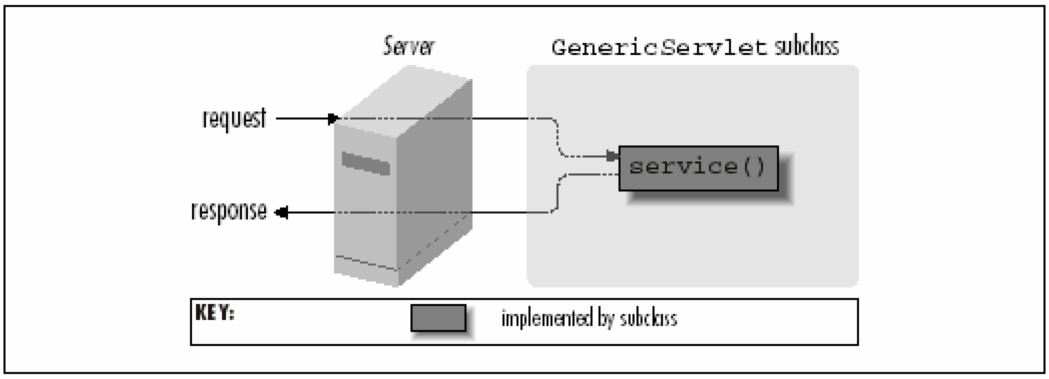

Figure 1: A generic Servlet handling a request

In contrast, an HTTP servlet usually does not override the service() method. Instead, it overrides doGet() to handle GET requests and doPost() to handle POST requests. An HTTP servlet can override either or both of these methods, depending on the type of requests it needs to handle. The service() method of HttpServlet handles the setup and dispatching to all the doXXX() methods, which is why it usually should not be overridden. *Figure 2* shows how an HTTP servlet handles GET and POST requests.

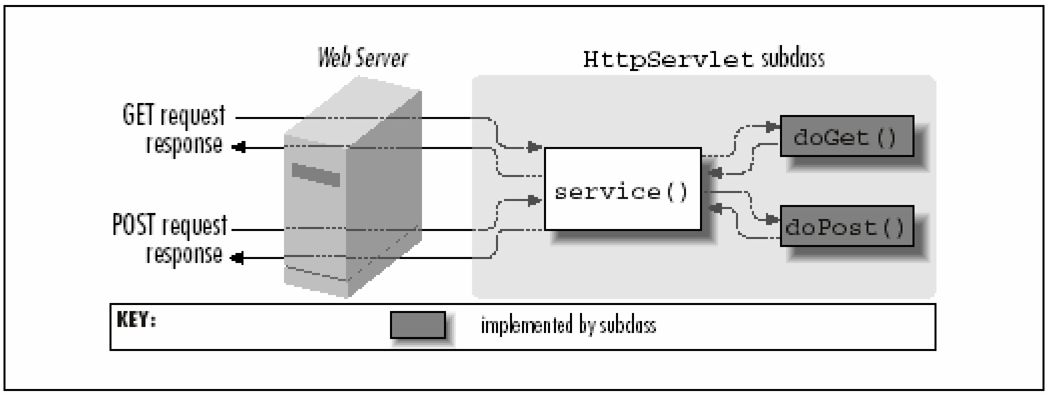

Figure 2: An Http Servlet handling get and post Requests

An HTTP servlet can override the doPut() and doDelete() methods to handle PUT and DELETE requests, respectively. However, HTTP servlets generally don't touch doHead(), doTrace(), or doOptions(). For these, the default implementations are almost always sufficient.

1.4 SERVLET LIFE CYCLE

After learning the basics of Servlet now, we shall study the life cycle of a Servlet. Servlets are normal Java classes, which are created when needed and destroyed when not needed. A Java Servlet has a lifecycle that defines how the Servlet is loaded and initialized, how it receives and responds to requests, and how it is taken out of service. In code, the Servlet lifecycle is defined by the javax.servlet.Servlet interface. Since Servlets run within a Servlet Container, creation and destruction of Servlets is the duty of Servlet Container. Implementing the init() and destroy() methods of Servlet interface allows you to be told by the Servlet Container that when it has created an instance of your Servlet and when it has destroyed that instance. An important point to remember is that your Servlet is not created and destroyed for every request it receives, rather it is created and kept in memory where requests are forwarded to it and your Servlet then generates response. We can understand Servlet Life Cycle with the help of *Figure 3*:

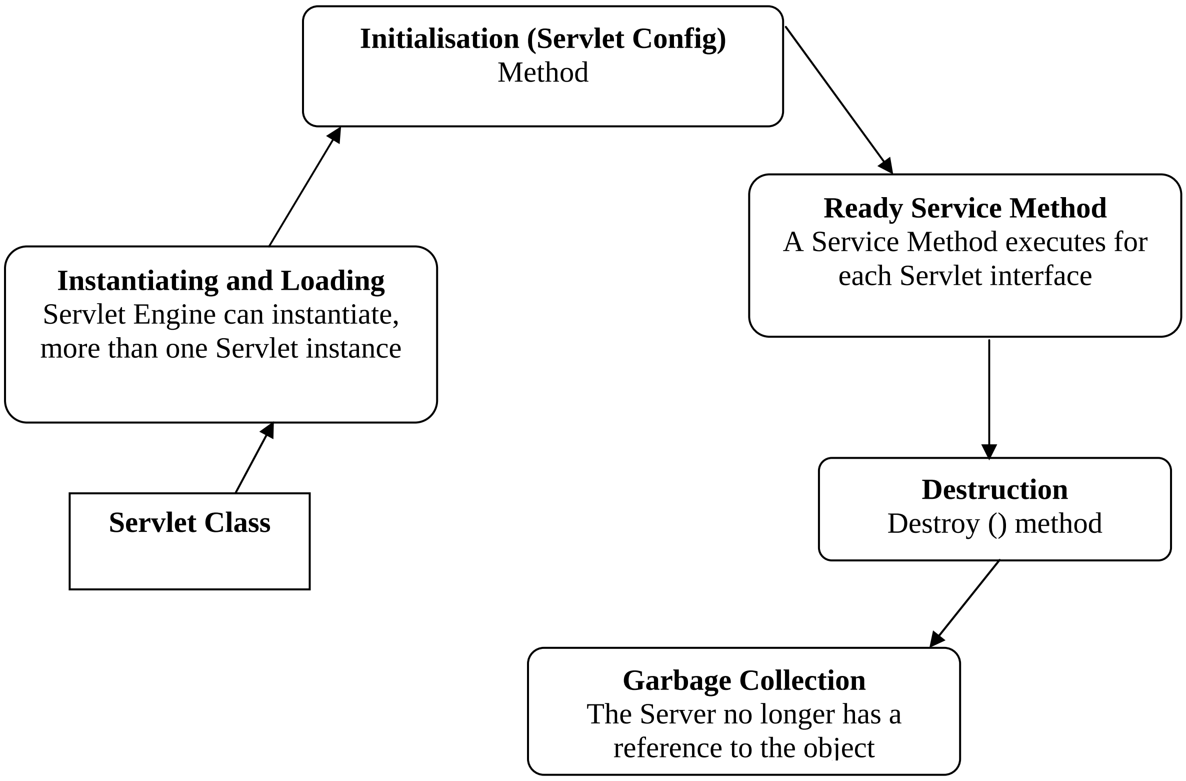
Servlet Class
Instantiating and Loading
Servlet Engine can instantiate, more than one Servlet instance
Initialisation (Servlet Config) Method
Ready Service Method
A Service Method executes for each Servlet interface
Destruction
Destroy () method
Garbage Collection
The Server no longer has a reference to the object

Figure 3: The Servlet Life Cycle

There are three principal stages in the life of a Java servlet, namely:

1) **Servlet Initialisation**: In this first stage, the servlet's constructor is called together with the servlet method init( )this is called automatically once during the servlet's execution life cycle and can be used to place any one-off initialisation such as opening a connection to a database.

So you have created your Servlet class in above-mentioned example, by extending the HttpServlet class and have placed it in /WEB-INF/classes/ directory of your application. Now, when will Servlet Container create an instance of your Servlet? There are a few situations:

- When you have specifically told the Servlet Container to preload your Servlet when the Servlet Container starts by setting the load-on-startup tag to a non-zero value e.g.

```
<web-app>
 <servlet>
   <servlet-name>test</servlet-name>
   <servlet-class>com.stardeveloper.servlets.TestServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
  </servlet>
</web-app>
```

So, when the Servlet Container starts it will preload your Servlet in the memory.

- If your Servlet is not already loaded, its instance will be created as soon as a request is received for it by the Servlet Container.

- During the loading of the Servlet into the memory, Servlet Container will call your Servlet's *init()* method.

2) **Servlet Execution:** Once your Servlet is initialised and its *init()* method called, any request that the Servlet Container receives will be forwarded to your Servlet's service() method. HttpServlet class breakes this *service()* method into more useful *doGet(), doPost(), doDelete(), doOptions(), doPut()* and *doTrace()* methods depending on the type of HTTP request it receives. So in order to generate response you should override the doGet() or doPost() method as per your requirement.

   At this moment all the requests will be forwarded to the appropriate doGet() or doPost() or whatever method as required. No new instance will be created for your Servlet.

   When a Servlet request is made to the Servlet engine, the Servlet engine receives all the request parameters (such as the IP address of client), user information and user data and constructs a Servlet request object, which encapsulates all this information.
   Another object, a Servlet response object is also created that contains all the information the Servlet needs to return output to the requesting client.

3) **Servlet Destruction:** When your application is stopped or Servlet Container shuts down, your Servlet's destroy() method will be called to clean up any resources allocated during initialisation and to shutdown gracefully. Hence, this acts as a good place to deallocate resources such as an open file or open database connection.

Point to remember: init() and destroy() methods will be called only once during the life time of the Servlet while service() and its broken down methods (doGet(), doPost() etc) will be called as many times as requests are received for them by the Servlet Container.

Let us understand all the phases of Servlet Life Cycle with the help of an example:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
public class TestServlet extends HttpServlet {
        private String name = "saurabh Shukla";
        public void init() throws ServletException
         {
                System.out.println("Calling  the method TestServlet : init");
        }
        public void destroy()
         {
                System.out.println("Calling  the method TestServlet : destroy");
        }

        public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws IOException, ServletException
          {
                res.setContentType("text/html");
                PrintWriter out = res.getWriter();
                // print content
                out.println("<html><head>");
                out.println("<title>TestServlet ( by ");
                out.println( name );
                out.println(" )</title>");
                out.println("</head>");
                out.println("<body>");
                out.println("<p>TestServlet ( by ");
                out.println( name );
                out.println(" ) :</p>");
                out.println("</body></html>");
                out.close();
        }
        public void doPost(HttpServletRequest req, HttpServletResponse res)
                throws IOException, ServletException {
                doGet(req, res);
        }
}
```

Our TestServlet is extremely simple to demonstrate the life cycle of a Servlet, which is displaying the name of the author to the user. We override four methods of HttpServlet class. init() is called by the Servlet Container only once during the initialization of the Servlet. Similarly destroy() is called once when removing the Servlet instance. We can thus put initialisation code in init() method. doGet() is called when a GET client request is received and doPost() is called when POST client request is received. In the doGet() method we set the content type to ‘text/html’ (telling the client browser that it is an HTML document). Then get hold of PrintWriter object and using it print our own HTML content to the client. Once done we close() it. Now, we will compile this Servlet in the same manner as we have compiled earlier:

```
C:\jakarta-tomcat-4.0-b5\webapps\star\WEB-INF\classes\
com\stardeveloper\servlets>javac TestServlet.java
```

If all goes well a 'TestServlet.class' file will be created in that folder. Now, open your browser and point to the following address:

http://localhost:8080/star/servlet/com.stardeveloper.servlets.TestServlet

You should see response from the Servlet showing "TestServlet".

Check Your Progress 1

1) State True or False:

   a) Servlet is not a Java Class. T ☐ F ☐

   b)Tomcat 4.0 is an open source and free Servlet Container and JSP Engine. T ☐ F ☐

   c) *init()* and *destroy()* methods will be called only once during the life time of the Servlet. T ☐ F ☐

2) What are the advantages of servlets over other common server extension mechanisms?

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

3) Write a Servlet program to display "Welcome to Fifth semester of MCA"

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

4) Explain different methods of service() method (of Servlet) to implement the request and response.

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

5) Draw, to represent the different phases of Servlet Life Cycle.

1.5 HTTPSERVLETREQUEST INTERFACE

There are two important interfaces included in the servlet API HttpServletRequest and HttpServletResponse

The interface HttpServletRequest encapsulates the functionality for a request object that is passed to an HTTP Servlet. It provides access to an input stream and so allows the servlet to read data from the client. The interface also provides methods for parsing the incoming HTTP FORM data and storing the individual data values - in particular *getParameterNames( )* returns the names of all the FORM's control/value pairs (request parameters). These control/value pairs are usually stored in an Enumeration object - such objects are often used for working with an ordered collection of objects.

Every call to doGet or doPost for an HTTPServlet receives an object that implements interface HTTPServletRequest. The web server that executes the servlet creates an HTTPRequest object and passes this to servlet's service method, hence this object contains the request from the client. A variety of methods are provided to enable the servlet to process the client's request. Some of these methods are listed below:

a) getCookies

public Cookie[] getCookies();

It returns an array containing all the cookies present in this request. Cookies can be used to uniquely identify clients to servlet. If there are no cookies in the request, then an empty array is returned.

b) getQueryString

public String getQueryString();

It returns query string present in the request URL if any. A query string is defined as any information following a ? character in the URL. If there is no query string, this method returns null.

c) getSession

public HttpSession getSession();
public HttpSession getSession(boolean create);

Returns the current valid session associated with this request. If this method is called with no arguments, a session will be created for the request if there is not already a session associated with the request. If this method is called with a Boolean argument, then the session will be created only if the argument is true.
To ensure the session is properly maintained, the servlet developer must call this method before the response is committed.
If the create flag is set to false and no session is associated with this request, then this method will return null.

d) getHeader

public String getHeader(String name);

Returns the value of the requested header. The match between the given name and the request header is case-insensitive. If the header requested does not exist, this method returns null.

e) getParameter(String Name)

public String getParameter(String name)

Returns the value associated with a parameter sent to the servlet as a part of a GET or POST request. The name argument represents the parameter name.

1.6 HTTP SERVLET RESPONSE INTERFACE

The interface HttpServletResponse encapsulates the functionality for a response object that is returned to the client from an HTTP Servlet. It provides access to an output stream and so allows the servlet to send data to the client. A key method is getWriter ( ) which obtains a reference to a PrintWriter object for it is this PrintWriter object that is used to send the text of the HTML document to the client.

Every call to doGet or doPost for an HTTPServlet receives an object that implements interface HTTPServletResponse. The web server that executes the servlet creates an HTTPRequest object and passes this to servlet’s service method, hence this object contains the response to the client. A variety of methods are provided to formulate the response to client. Some of these methods are listed below:

a) addCookie

public void addCookie(Cookie cookie);

It is used to add the specified cookie to the header of response. This method can be called multiple times to set more than one cookie. This method must be called before the response is committed so that the appropriate headers can be set. The cookies maximum age and whether the client allows Cookies to be saved determine whether or not Cookies will be stored on the client.

b) sendError

public void sendError(int statusCode) throws IOException;
public void sendError(int statusCode, String message) throws IOException;

It sends an error response to the client using the specified status code. If a message is provided to this method, it is emitted as the response body, otherwise the server should return a standard message body for the error code given. This is a convenience method that immediately commits the response. No further output should be made by the servlet after calling this method.

c) getWriter

public Printwriter getWriter()

It obtains a character-based output stream that enables text data to be sent to the client.

d) getOutputStream()

public ServletOutputStream getOutputStream()

It obtains a byte-based output stream that enables binary data to sent to the client.

e) sendRedirect

public void sendRedirect(String location) throws IOException;

It sends a temporary redirect response to the client (SC_MOVED_TEMPORARILY) using the specified location. The given location must be an absolute URL. Relative URLs are not permitted and throw an IllegalArgumentException.

This method must be called before the response is committed. This is a convenience method that immediately commits the response. No further output is be made by the servlet after calling this method.

1.7 SESSION TRACKING

Many web sites today provide custom web pages and / or functionality on a client-by-client basis. For example, some Web sites allow you to customize their home page to suit your needs. An excellent example of this the *Yahoo!* Web site. If you go to the site http://my.yahoo.com/

You can customize how the Yahoo! Site appears to you in future when you revisit the website. HTTP is a stateless protocol: it provides no way for a server to recognise that a sequence of requests is all from the same client. Privacy advocates may consider this a feature, but it causes problems because many web applications aren’t stateless. The shopping cart application is another classic example—a client can put items in his virtual cart, accumulating them until he checks out several page requests later.

Obviously the server must distinguish between clients so the company can determine the proper items and charge the proper amount for each client.

Another purpose of customizing on a client-by-client basis is marketing. Companies often track the pages you visit throughout a site so they display advertisements that are targeted to user's browsing needs.

To help the server distinguish between clients, each client must identify itself to the server. There are a number of popular techniques for distinguishing between clients. In this unit, we introduce one of the techniques called as Session Tracking.

Session tracking is wonderfully elegant. Every user of a site is associated with a javax.servlet.http.HttpSession object that servlets can use to store or retrieve information about that user. You can save any set of arbitrary Java objects in a session object. For example, a user's session object provides a convenient location for a servlet to store the user's shopping cart contents.

A servlet uses its request object's getSession() method to retrieve the current HttpSession object:

public HttpSession HttpServletRequest.getSession(boolean create)

This method returns the current session associated with the user making the request. If the user has no current valid session, this method creates one if create is true or returns null if create is false. To ensure the session is properly maintained, this method must be called at least once before any output is written to the response.
You can add data to an HttpSession object with the putValue() method:
public void HttpSession.putValue(String name, Object value)

This method binds the specified object value under the specified name. Any existing binding with the same name is replaced. To retrieve an object from a session, use

getValue():
public Object HttpSession.getValue(String name)

This methods returns the object bound under the specified name or null if there is no binding. You can also get the names of all of the objects bound to a session with

getValueNames():
public String[] HttpSession.getValueNames()

This method returns an array that contains the names of all objects bound to this session or an empty (zero length) array if there are no bindings. Finally, you can remove an object from a session with removeValue():
public void HttpSession.removeValue(String name)

This method removes the object bound to the specified name or does nothing if there is no binding. Each of these methods can throw a java.lang.IllegalStateException if the session being accessed is invalid.

A Hit Count Using Session Tracking

Let us understand session tracking with a simple servlet to count the number of times a client has accessed it, as shown in example below. The servlet also displays all the bindings for the current session, just because it can.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SessionTracker extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
{
res.setContentType("text/html");
PrintWriter out = res.getWriter();
// Get the current session object, create one if necessary
HttpSession session = req.getSession(true);
// Increment the hit count for this page. The value is saved
// in this client's session under the name "tracker.count".
Integer count = (Integer)session.getValue("tracker.count");
if (count == null)
count = new Integer(1);
else
count = new Integer(count.intValue() + 1);
session.putValue("tracker.count", count);
out.println("<HTML><HEAD><TITLE>SessionTracker</TITLE></HEAD>");
out.println("<BODY><H1>Session Tracking Demo</H1>");
// Display the hit count for this page
out.println("You've visited this page " + count +((count.intValue() == 1) ? " time." : "
times."));
out.println("<P>");
out.println("<H2>Here is your session data:</H2>");
String[] names = session.getValueNames();
for (int i = 0; i < names.length; i++) {
out.println(names[i] + ": " + session.getValue(names[i]) + "<BR>");
}
out.println("</BODY></HTML>");
}}
```

The output will appear as:

```
Session Tracking Demo
You've visited this page 12 times
Here is your session Data
movie.level : beginner
movie.zip : 50677
tracker.count : 12
```

This servlet first gets the HttpSession object associated with the current client. By passing true to *getSession()*, it asks for a session to be created if necessary. The servlet then gets the Integer object bound to the name “tracker.count”. If there is no such object, the servlet starts a new count. Otherwise, it replaces the Integer with a new Integer whose value has been incremented by one. Finally, the servlet displays the current count and all the current name/value pairs in the session.

Session Tracking using persistent Cookies

Another technique to perform session tracking involves **persistent cookies**. A cookie is a bit of information sent by a web server to a browser is stored it on a client machine that can later be read back from that browser. Persistent cookies offer an elegant, efficient, easy way to implement session tracking. Cookies provide as automatic introduction for each request as you could hope for. For each request, a cookie can automatically provide a client’s session ID or perhaps a list of the client’s preferences. In addition, the ability to customize cookies gives them extra power and versatility. When a browser receives a cookie, it saves the cookie and thereafter sends the cookie back to the server each time it accesses a page on that server, subject to certain rules. Because a cookie's value can uniquely identify a client, cookies are often used for session tracking.

Note: Cookies were first introduced in Netscape Navigator. Although they were not part of the official HTTP specification, cookies quickly became a de facto standard supported in all the popular browsers including Netscape 0.94 Beta and up and Microsoft Internet Explorer 2 and up.

Problem: The biggest problem with cookies is that all the browsers don't always accept cookies. Sometimes this is because the browser doesn't support cookies. Version 2.0 of the Servlet API provides the javax.servlet.http.Cookie class for working with cookies. The HTTP header details for the cookies are handled by the Servlet API. You create a cookie with the Cookie() constructor:

```
public Cookie(String name, String value)
```

This creates a new cookie with an initial name and value..
A servlet can send a cookie to the client by passing a Cookie object to the addCookie() method of HttpServletResponse:

```
public void HttpServletResponse.addCookie(Cookie cookie)
```

This method adds the specified cookie to the response. Additional cookies can be added with subsequent calls to addCookie(). Because cookies are sent using HTTP headers, they should be added to the response before you send any content. Browsers are only required to accept 20 cookies per site, 300 total per user, and they can limit each cookie's size to 4096 bytes.

The code to set a cookie looks like this:

```
Cookie cookie = new Cookie("ID", "123");
res.addCookie(cookie);
```

A servlet retrieves cookies by calling the getCookies() method of HttpServlet-Request:

```
public Cookie[] HttpServletRequest.getCookies()
```

This method returns an array of Cookie objects that contains all the cookies sent by the browser as part of the request or null if no cookies were sent. The code to fetch cookies looks like this:

```
Cookie[] cookies = req.getCookies();
if (cookies != null) {
for (int i = 0; i < cookies.length; i++) {
String name = cookies[i].getName();
String value = cookies[i].getValue();
}
}
```

You can set a number of attributes for a cookie in addition to its name and value. The following methods are used to set these attributes. As you will see, there is a corresponding get method for each set method. The get methods are rarely used, however, because when a cookie is sent to the server, it contains only its name, value, and version.

Here some of the methods of cookies are listed below which are used for session tracking:

public void Cookie.setMaxAge(int expiry)

Specifies the maximum age of the cookie in seconds before it expires. A negative value indicates the default, that the cookie should expire when the browser exits. A zero value tells the browser to delete the cookie immediately.

public int getMaxAge();

This method returns the maximum specified age of the cookie. If no maximum age was specified, this method returns -1.

public void Cookie.setVersion(int v)

Sets the version of a cookie. Servlets can send and receive cookies formatted to match either Netscape persistent cookies (Version 0) or the newer
public String getDomain();

Returns the domain of this cookie, or null if not defined.

public void Cookie.setDomain(String pattern)

This method sets the domain attribute of the cookie. This attribute defines which hosts the cookie should be presented to by the client. A domain begins with a dot (.foo.com) and means that hosts in that DNS zone (www.foo.com but not a.b.foo.com) should see the cookie. By default, cookies are only returned to the host which saved them.

public void Cookie.setPath(String uri)

It indicates to the user agent that this cookie should only be sent via secure channels (such as HTTPS). This should only be set when the cookie's originating server used a secure protocol to set the cookie's value.

public void Cookie.setValue(String newValue)

Assigns a new value to a cookie.

public String getValue()

This method returns the value of the cookie.

Let us understand how we use persistent cookies for the session tracking with the help of shopping cart example

```
// Session tracking using persistent cookies
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ShoppingCartViewerCookie extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res)throws
ServletException, IOException
{
res.setContentType("text/html");
PrintWriter out = res.getWriter();
// Get the current session ID by searching the received cookies.
String sessionid = null;
Cookie[] cookies = req.getCookies();
if (cookies != null) {
for (int i = 0; i < cookies.length; i++) {
if (cookies[i].getName().equals("sessionid")) {
sessionid = cookies[i].getValue();
break;
}
}
}
// If the session ID wasn't sent, generate one.
// Then be sure to send it to the client with the response.
if (sessionid == null) {
sessionid = generateSessionId();
Cookie c = new Cookie("sessionid", sessionid);
res.addCookie(c);
}
out.println("<HEAD><TITLE>Current Shopping Cart
Items</TITLE></HEAD>");
out.println("<BODY>");
// Cart items are associated with the session ID
String[] items = getItemsFromCart(sessionid);
// Print the current cart items.
out.println("You currently have the following items in your cart:<BR>");
if (items == null) {
out.println("<B>None</B>");
}
else {
out.println("<UL>");
for (int i = 0; i < items.length; i++) {
out.println("<LI>" + items[i]);
}
out.println("</UL>");
}
// Ask if they want to add more items or check out.
out.println("<FORM ACTION=\"/servlet/ShoppingCart\" METHOD=POST>");
out.println("Would you like to<BR>");
out.println("<INPUT TYPE=submit VALUE=\" Add More Products/Items \">");
out.println("<INPUT TYPE=submit VALUE=\" Check Out \">");
out.println("</FORM>");
// Offer a help page.
out.println("For help, click <A HREF=\"/servlet/Help" +
"?topic=ShoppingCartViewerCookie\">here</A>");
out.println("</BODY></HTML>");
}
private static String generateSessionId() {
String uid = new java.rmi.server.UID().toString(); // guaranteed unique
return java.net.URLEncoder.encode(uid); // encode any special chars
}
private static String[] getItemsFromCart(String sessionid) {
// Not implemented
}
}
```

This servlet first tries to fetch the client's session ID by iterating through the cookies it received as part of the request. If no cookie contains a session ID, the servlet generates a new one using generateSessionId() and adds a cookie containing the new session ID to the response.

1.8 DATABASE CONNECTIVITY WITH SERVLETS

Now we shall study how we can connect servlet to database. This can be done with the help of JDBC (Java Database Connectivity). Servlets, with their enduring life cycle, and JDBC, a well-defined database-independent database connectivity API, are an elegant and efficient combination and solution for webmasters who require to connect their web sites to back-end databases.

The advantage of servlets over CGI and many other technologies is that JDBC is database-independent. A servlet written to access a Sybase database can, with a two-line modification or a change in a properties file, begin accessing an Oracle database. One common place for servlets, especially servlets that access a database, is in what's called the middle tier. A middle tier is something that helps connect one endpoint to another (a servlet or applet to a database, for example) and along the way adds a little something of its own. The middle tier is used between a client and our ultimate data source (commonly referred to as middleware) to include the business logic.
Let us understand the database connectivity of servlet with table with the help of an example. The following example shows a very simple servlet that uses the MS-Access JDBC driver to perform a simple query, printing names and phone numbers for all employees listed in a database table. We assume that the database contains a table named CUSTOMER, with at least two fields, NAME and ADDRESS.

```
/* Example to demonstrate how JDBC is used with Servlet to connect to a customer
table and to display its records*/
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DBPhoneLookup extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
Connection con = null;
Statement stmt = null;
ResultSet rs = null;
res.setContentType("text/html");
PrintWriter out = res.getWriter();
try {
// Load (and therefore register) the Oracle Driver
Class.forName(sun.jdbc.odbc.JdbcOdbcDriver);
// Get a Connection to the database
Connection con = DriverManager.getConnection (jdbc:odbc:Access);
// Create a Statement object
stmt = con.createStatement();
// Execute an SQL query, get a ResultSet
rs = stmt.executeQuery("SELECT NAME, ADDRESS FROM CUSTOMER");
// Display the result set as a list
out.println("<HTML><HEAD><TITLE>Phonebook</TITLE></HEAD>");
out.println("<BODY>");
out.println("<UL>");
while(rs.next()) {
out.println("<LI>" + rs.getString("name") + " " + rs.getString("address"));
}
out.println("</UL>");
out.println("</BODY></HTML>");
}
catch(ClassNotFoundException e) {
out.println("Couldn't load database driver: " + e.getMessage());
}
catch(SQLException e) {
out.println("SQLException caught: " + e.getMessage());
}
finally {
// Always close the database connection.
try {
if (con != null) con.close();
}
catch (SQLException e1) { }
}
}
}
```

In the above example a simple servlet program is written to connect to the database, and which executes a query that retrieves the names and phone numbers of everyone in the employees table, and display the list to the user.

1.9 INTER-SERVLET COMMUNICATION

Now, we shall study why we need InterServlet communication. Servlets which are running together in the same server have several ways to communicate with each other. There are three major reasons to use interservlet communication:

a) Direct servlet manipulation / handling

A servlet can gain access to the other currently loaded servlets and perform some task on each. The servlet could, for example, periodically ask every servlet to write its state to disk to protect against server crashes.

Direct servlet manipulation / handling involves one servlet accessing the loaded servlets on its server and optionally performing some task on one or more of them. A servlet obtains information about other servlets through the ServletContext object.

Use getServlet() to get a particular servlet:
public Servlet ServletContext.getServlet(String name) throws ServletException
This method returns the servlet of the given name, or null if the servlet is not found. The specified name can be the servlet's registered name (such as "file") or its class name (such as "com.sun.server.webserver.FileServlet"). The server maintains one servlet instance per name, so getServlet("file") returns a different servlet instance than getServlet("com.sun.server.webserver .FileServlet").
You can also get all of the servlets using getServlets():
public Enumeration ServletContext.getServlets()

This method returns an Enumeration of the servlet objects loaded in the current ServletContext. Generally there's one servlet context per server, but for security or convenience, a server may decide to partition its servlets into separate contexts. The enumeration always includes the calling servlet itself.

Let us take an example to understand how we can view the currently loaded servlets.

```
//Example Checking out the currently loaded servlets
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Loaded extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
res.setContentType("text/plain");
PrintWriter out = res.getWriter();
ServletContext context = getServletContext();
Enumeration names = context.getServletNames();
while (names.hasMoreElements()) {
String name = (String)names.nextElement();
Servlet servlet = context.getServlet(name);
out.println("Servlet name: " + name);
out.println("Servlet class: " + servlet.getClass().getName());
out.println("Servlet info: " + servlet.getServletInfo());
out.println();
}
}
}
```

In the above example, it retrieves its ServletContext to access the other servlets loaded in the server. Then it calls the context's getServletNames() method. This returns an enumeration of String objects that the servlet iterates over in a while loop. For each name, it retrieves the corresponding servlet object with a call to the context's getServlet() method. Then it prints three items of information about the servlet: its name, its class name, and its getServletInfo() text.

b) Servlet reuse

Another use for interservlet communication is to allow one servlet to reuse the abilities (the public methods) of another servlet. The major challenge with servlet reuse is for the "user" servlet to obtain the proper instance of "usee" servlet when the usee servlet has not yet been loaded into the server. For example a servlet named as ChatServlet was written as a server for chat applets, but it could be reused (unchanged) by another servlet that needed to support an HTML-based chat interface. Servlet can be done with the user servlet to ask the server to load the usee servlet, then call getServlet() to get a reference to it. Unfortunately, the Servlet API distinctly lacks any methods whereby a servlet can control the servlet life cycle, for itself or for other servlets. This is considered a security risk and is officially "left for future consideration." Fortunately, there's a backdoor we can use today. A servlet can open an HTTP connection to the server in which it's running, ask for the unloaded servlet, and effectively force the server to load the servlet to handle the request. Then a call to getServlet() gets the proper instance.

c) Servlet collaboration

Sometimes servlets have to cooperate, usually by sharing some information. We call this type of communication as servlet collaboration. Collaborating servlets can pass the shared information directly from one servlet to another through method invocations. This approach requires each servlet to know the other servlets with which it is collaborating. The most common situation involves two or more servlets sharing state information. For example, a set of servlets managing an online store could share the store's product inventory count. Session tracking can be considered as a special case of servlet collaboration.

Colloboration using system property list:

One simple way for servlets to share information is by using Java's system-wide Properties list, found in the java.lang.System class. This Properties list holds the standard system properties, such as java.version and path. separator, but it can also hold application-specific properties. Servlets can use the properties list to hold the information they need to share. A servlet can add (or change) a property by calling:
System.getProperties().put("key", "value");
That servlet, or another servlet running in the same JVM, can later get the value of the property by calling:
String value = System.getProperty("key");
The property can be removed by calling:
System.getProperties().remove("key");

The Properties class is intended to be String based, meaning that each key and value is supposed to be a String.

Collaboration through a shared object :

Another way for servlets to share information is through a shared object. A shared object can hold the pool of shared information and make it available to each servlet as needed. In a sense, the system Properties list is a special case example of a shared object. By generalising the technique into sharing any sort of object, however, a servlet is able to use whatever shared object best solves its particular problem.

Often the shared object incorporates a fair amount of business logic or rules for manipulating the object's data. This business logic protects the shared object's actual data by making it available only through well-defined methods.

There's one thing to watch out for when collaborating through a shared object is the garbage collector. It can reclaim the shared object if at any time the object isn't referenced by a loaded servlet. To keep the garbage collector at bay, every servlet using a shared object should save a reference to the object.

Check Your Progress 2

1) What are the main functions of HTTPServletRequest Interface? Explain the methods which are used to obtain cookies and querystring from the request object.

   ………………..………..……………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

2) What are the main functions of HTTPServletResponse Interface? Explain the methods which are used to add cookies to response and send an error response.

   ………………..………..……………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

3) Explain the various purposes for which we use Session tracking. Also, Explain in brief the two ways to handle Session Tracking in Servlets.

   ………………..………..……………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

4) Assume there is a table named Product in MS-access with fields (Product_id, Prod_name, Price, Qty). Write a code for Servlet which will display all the fields of product table in Tabular manner.

   ………………..………..……………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

5) What are the two ways used for Servlet collaboration

   ………………..………..……………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

6) How do I call a servlet with parameters in the URL?

   ………………..………..……………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………………

7) How do I deserialize an httpsession?

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

8) How do I restrict access to servlets and JSPs?

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

9) What is the difference between JSP and servlets ?

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

10) Difference between GET and POST .

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

11) Can we use the constructor, instead of init(), to initialize servlet?

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

12) What is servlet context ?

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

13) What are two different types of servlets ? Explain the differences between these two.

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

14) What is the difference between ServletContext and ServletConfig?

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

15) What are the differences between a session and a cookie?

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

16) How will you delete a cookie?

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

17) What is the difference between Context init parameter and Servlet init parameter?

………………..………..………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

18) What are the different types of ServletEngines?

………………..………..………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

19) What is Servlet chaining?

………………..………..………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

1.10 SUMMARY

Java servlets are the small, platform-independent Java programs that run in a web server or application server and provide server-side processing such as accessing a database and e-commerce transactions. Servlets are widely used for web processing. Servlets dynamically extend the functionality of a web server. A servlet engine can only execute servlet which is contained in the web-servers like, JWS or JIGSAW.

Servlets are basically developed for the server side applications and designed to handle http requests. They are better than other common server extensions like, CGI as they are faster, have all the advantages of Java language and supported by many of the browsers.

A Java Servlet has a lifecycle that defines how the servlet is loaded and initialised, how it receives and responds to requests, and how it is taken out of service. Servlets run within a Servlet Container, creation and destruction of servlets is the duty of Servlet Container. There are three principal stages in the life of a Java Servlet, namely: Servlet Initialisation, Servlet Execution and Servlet Destruction. In this first stage, the servlet's constructor is called together with the servlet method init( ) - this is called automatically once during the servlet's execution life cycle. Once your servlet is initialised, any request that the Servlet Container receives will be forwarded to your Servlet's service() method. HttpServlet class breaks this service() method into more useful doGet(), doPost(), doDelete(), doOptions(), doPut() and doTrace() methods depending on the type of HTTP request it receives. When the application is stopped or Servlet Container shuts down, your Servlet's destroy() method will be called to clean up any resources allocated during initialisation and to shutdown gracefully.

There are two important interfaces included in the servlet API. They are HttpServletRequest and HttpServletResponse. HttpServletRequest encapsulates the functionality for a request object that is passed to an HTTP Servlet. It provides access to an input stream and so allows the servlet to read data from the client and it has methods like, getCookies(), getQueryString() & getSession etc. HttpServletResponse encapsulates the functionality for a response object that is returned to the client from an HTTP Servlet. It provides access to an output stream and so allows the servlet to send data to the client and it has methods like, addCookie(), sendError() and getWriter() etc.

Session tracking is another important feature of servlet. Every user of a site is associated with a javax.servlet.http.HttpSession object that servlets can use to store or retrieve information about that user. A servlet uses its request object's getSession()

method to retrieve the current HttpSession object and can add data to an HttpSession object with the putValue() method. Another technique to perform session tracking involves persistent cookies. A cookie is a bit of information sent by a web server to a browser and stores it on a client machine that can later be read back from that browser. For each request, a cookie can automatically provide a client's session ID or perhaps a list of the client's preferences.

Servlet along JDBC API can be used to connect to the different databases like, Sybase, Oracle etc. A servlet written to access a Sybase database can, with a two-line modification or a change in a properties file, begin accessing an Oracle database. It again uses the objects and methods of java.sql.* package.

Servlets, which are running together in the same server, have several ways to communicate with each other. There are three reasons to use InterServlet communication. First is Direct Servlet manipulation handling in which servlet can gain access to the other currently loaded servlets and perform some task on each. Second is Servlet Reuse that allows one servlet to reuse the abilities (the public methods) of another servlet. Third is Servlet colloboration that allows servlets to cooperate, usually by sharing some information.

1.11 SOLUTIONS/ANSWERS

Check Your Progress 1

1) True/ False

   a) False b) True c) True

2) A servlet is a Java class and therefore needs to be executed in a Java VM by a service we call a Servlet engine. Servlets dynamically extend the functionality of a web server and basically developed for the server side applications. Servlets have the following advantages over other common server extension mechanisms:

   - They are faster than other server extensions like, CGI scripts because they use a different process model.
   - They use a standard API that is supported by many web servers.
   - It executes within the address space of a web server.
   - Since servlets are written in Java, servlets are portable between servers and operating systems. They have all of the advantages of the Java language, including ease of development and platform independence.
   - It does not require creation of a separate process for each client request.

3) Code to display **"Welcome to Fifth semester of MCA"**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
```

```
            out.println("<HEAD><TITLE>5th Semester MCA </TITLE></HEAD>");
            out.println("<BODY>");
            out.println("<B>Welcome to Fifth Semester of MCA</B>");
            out.println("</BODY></HTML>");
}
}
```

4) Servlets are the Java classes which are created when needed and destroyed when not needed. Since servlets run within a Servlet Container, creation and destruction of servlets is the duty of Servlet Container. There are three principal stages in the life of a Java Servlet Life Cycle, namely:

i) **Servlet Initialisation**: In this first stage, the servlet's constructor is called together with the servlet method init( ) - this is called automatically once during the servlet's execution life cycle and can be used to place any one-off initialisation such as opening a connection to a database.

ii) **Servlet Execution:** Once your servlet is initialized and its init() method called, any request that the Servlet Container receives will be forwarded to your Servlet's service() method. HttpServlet class breaks this service() method into more useful doGet(), doPost(), doDelete(), doOptions(), doPut() and doTrace() methods depending on the type of HTTP request it receives. So in order to generate response, the doGet() or doPost() method should be overridden as per the requirement.

When a servlet request is made to the Servlet engine, the Servlet engine receives all the request parameters (such as the IP address of client), user information and user data and constructs a Servlet request object, which encapsulates all this information.

iii) **Servlet Destruction:** When the application is stopped or Servlet Container shuts down, Servlet’s destroy() method will be called to clean up any resources allocated during initialisation and to shutdown gracefully. Hence, it acts as a place to deallocate resources such as an open file or open database connection.

5)

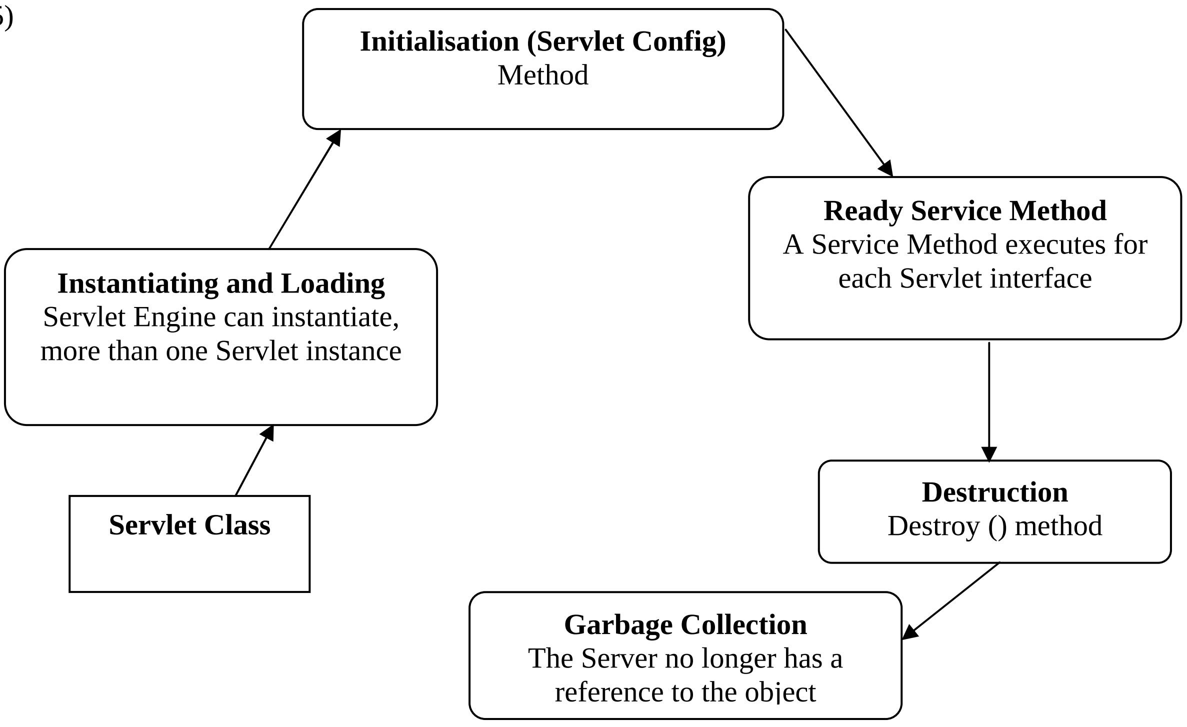
Initialisation (Servlet Config) Method
Ready Service Method A Service Method executes for each Servlet interface
Instantiating and Loading Servlet Engine can instantiate, more than one Servlet instance
Servlet Class
Destruction Destroy () method
Garbage Collection The Server no longer has a reference to the object

Figure 4: The servlet life cycle

Check Your Progress 2

1. Main functions of HttpServletRequest Interface are the following:

- The interface HttpServletRequest encapsulates the functionality for a request object that is passed to an HTTP Servlet.
- It provides access to an input stream and so allows the servlet to read data from the client.
- It provides methods for parsing the incoming HTTP FORM data and storing the individual data values - in particular getParameterNames( ) returns the names of all the FORM's control/value pairs

- It contains the request from the client

getCookies is the method which is used to obtain cookies from the request object and following is the its syntax:

```
public Cookie[] getCookies();
```

It returns an array containing all the cookies present in this request. Cookies can be used to uniquely identify clients to servlet. If there are no cookies in the request, then an empty array is returned.

GetQueryString is the method used to obtain the querystring from the request object. The syntax used is

```
public String getQueryString();
```

It returns query string present in the request URL if any. A query string is defined as any information following a ? character in the URL. If there is no query string, this Method returns null.

2) Main functions of HttpServlet Response Interface are:

- It encapsulates the functionality for a response object that is returned to the client from an HTTP Servlet.
- It provides access to an output stream and so allows the servlet to send data to the client.
- It uses getWriter( ) method to obtain a reference to a PrintWriter object.and PrintWriter object is used to send the text of the HTML document to the client.
- The web server that executes the servlet creates an HTTPRequest object and passes this to servlet's service method.
- It contains the response to the client.

addCookie is the method which is used to add cookies to the response object. Syntax is

```
public void addCookie(Cookie cookie);
```

It is used to add the specified cookie to the header of response. This method can be called multiple times to set more than one cookie. This method must be called before the response is committed so that the appropriate headers can be set.

sendError is the method used to send an error response. Syntax is :

```
public void sendError(int statusCode) throws IOException;
```

public void sendError(int statusCode, String message) throws IOException;

It sends an error response to the client using the specified status code. If a message is provided to this method, it is emitted as the response body, otherwise the server should return a standard message body for the error code given.

3) Various purposes for the session tracking are:

- To know the client's preferences
- To distinguish between different clients
- To customize the website like shopping cart as per the user requirement or preference.

Session Tracking using persistent Cookies: A cookie is a bit of information sent by a web server to a browser and stores it on a client machine that can later be read back from that browser. Persistent cookies offer an elegant, efficient, easy way to implement session tracking. For each request, a cookie can automatically provide a client's session ID or perhaps a list of the client's preferences. In addition, the ability to customize cookies gives them extra power and versatility. When a browser receives a cookie, it saves the cookie and thereafter sends the cookie back to the server each time it accesses a page on that server, subject to certain rules. Because a cookie's value can uniquely identify a client, cookies are used for session tracking.

Cookie can be created with the Cookie() constructor:

public Cookie(String name, String value)

A servlet can send a cookie to the client by passing a Cookie object to the

addCookie() method of HttpServletResponse:

public void HttpServletResponse.addCookie(Cookie cookie)

4) Code for servlet to display all the fields of PRODUCT Table:

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DBPhoneLookup extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        try {
        // Load (and therefore register) the Oracle Driver
        Class.forName(sun.jdbc.odbc.JdbcOdbcDriver);
        // Get a Connection to the database
        Connection con = DriverManager.getConnection (jdbc:odbc:Access);
        // Create a Statement object
        stmt = con.createStatement();
        // Execute an SQL query, get a ResultSet
        rs = stmt.executeQuery("SELECT Product_id, Prod_name, Price, Qty FROM
        PRODUCT");
        // Display the result set as a list
```

```
          out.println("<HTML><HEAD><TITLE>Phonebook</TITLE></HEAD>");
          out.println("<BODY>");
          out.println("<Table>");
          while(rs.next()) {
          out.println("<TR>");
          out.println("<TD>" + rs.getString("Product_id") + "</TD><TD>" +
          rs.getString("Prod_name")  + "</TD><TD> "  +  rs.getString("Price")  +
          "</TD><TD> "  +  rs.getString("Qty")  + "</TD>");
          out.println("</TR>");

          }
          out.println("</Table>");
          out.println("</BODY></HTML>");
          }
          catch(ClassNotFoundException e) {
          out.println("Couldn't load database driver: " + e.getMessage());
          }
          catch(SQLException e) {
          out.println("SQLException caught: " + e.getMessage());
          }
          finally {
          // Always close the database connection.
          try {
          if (con != null) con.close();
          }
          catch (SQLException e1) { }
}}}
```

5) The two ways used for servlet collaboration are the following:

   a) Colloboration using system property list: One simple way for servlets to share information is by using Java's system-wide Properties list, found in the java.lang.System class. This Properties list holds the standard system properties, such as java.version and path. separator, but it can also hold application-specific properties. Servlets can use the properties list to hold the information they need to share. A servlet can add(or change) a property by calling:
   System.getProperties().put("key", "value");

   b) Collaboration through a shared object : Another way for servlets to share information is through a shared object. A shared object can hold the pool of shared information and make it available to each servlet as needed, the system Properties list is a special case example of a shared object. The shared object may incorporate the business logic or rules for manipulating the object's data.

6) The usual format of a servlet parameter is a name=value pair that comes after a question-mark (?) at the end of the URL. To access these parameters, call the getParameter() method on the HttpServletRequest object, then write code to test the strings. For example, if your URL parameters are "func=topic," where your URL appears as:

http://www..ignou.ac.in/myservlet?func=topic then you could parse the parameter as follows, where "req" is the HttpServletRequest object:

```
 String func = req.getParameter("func");
if (func.equalsIgnoreCase("topic"))
        {    . . . Write an appropriate code}
```

7) To deserialise an httpsession, construct a utility class that uses the current thread’s contextclassloader to load the user defined objects within the application context. Then add this utility class to the system CLASSPATH.

8) The Java Servlet API Specification v2.3 allows you to declaratively restrict access to specific servlets and JSPs using the Web Application deployment descriptor. You can also specify roles for EJBs and Web applications through the Administration Console.

9) JSP is used mainly for presentation only. A JSP can only be HttpServlet that means the only supported protocol in JSP is HTTP. But a servlet can support any protocol like, HTTP, FTP, SMTP etc.

10) In GET entire form submission can be encapsulated in one URL, like a hyperlink. Query length is limited to 255 characters, not secure, faster, quick and easy. The data is submitted as part of URL. Data will be visible to user.

    In POST data is submitted inside body of the HTTP request. The data is not visible on the URL and it is more secure.

11) Yes. But you will not get the servlet specific things from constructor. The original reason for init() was that ancient versions of Java couldn’t dynamically invoke constructors with arguments, so there was no way to give the constructor a ServletConfig. That no longer applies, but servlet containers still will only call your no-arg constructor. So you won’t have access to a ServletConfig or ServletContext.

12) The servlet context is an object that contains information about the web application and container. Using the context, a servlet can log events, obtain URL references to resources, and set and store attributes that other servlets in the context can use.

13) GenericServlet and HttpServlet. HttpServlet is used to implement HTTP protocol, whereas Generic servlet can implement any protocol.

    By extending GenericServlet we can write a servlet that supports our own custom protocol or any other protocol.

14) The ServletConfig gives the information about the servlet initialization parameters. The servlet engine implements the ServletConfig interface in order to pass configuration information to a servlet. The server passes an object that implements the ServletConfig interface to the servlet’s init() method. The ServletContext gives information about the container. The ServletContext interface provides information to servlets regarding the environment in which they are running. It also provides standard way for servlets to write events to a log file.

15) Session is stored in server but cookie stored in client. Session should work regardless of the settings on the client browser. There is no limit on the amount of data that can be stored on session. But it is limited in cookie. Session can store objects and cookies can store only strings. Cookies are faster than session.

16)
```
   Cookie c = new Cookie (“name”, null);

   c.setMaxAge(0);

response.addCookie(killCookie);
```

17) Servlet init parameters are for a single servlet only. No body outside that servlet can access that. It is declared inside the <servlet> tag inside Deployment Descriptor, whereas context init parameter is for the entire web application. Any servlet or JSP in that web application can access context init parameter. Context parameters are declared in a tag <context-param> directly inside the <web-app> tag. The methods for accessing context init parameter is getServletContext ().getInitParamter ("name") whereas method for accessing servlet init parameter is getServletConfig ().getInitParamter ("name");

18) The different types of ServletEngines available are: Standalone ServletEngine: This is a server that includes built-in support for servlets. Add-on ServletEngine: It is a plug-in to an existing server. It adds servlet support to a server that was not originally designed with servlets in mind.

19) Servlet chaining is a technique in which two or more servlets can cooperate in servicing a single request. In servlet chaining, one servlet's output is the input of the next servlet. This process continues until the last servlet is reached. Its output is then sent back to the client. We are achieving Servlet Chaining with the help of RequestDispatcher.

1.12 FURTHER READINGS/REFERENCES

- Dietel and Dietel, *Internet & World wide Web Programming,* Prentice Hall
- Potts, Stephen & Pestrikov, Alex, *Java 2 Unleashed,* Sams
- Keogh, James, *J2EE: The Complete Reference*, McGraw-Hill
- Inderjeet Singh & Bet Stearns, *Designing Enterprise Application with j2EE*, Second Edition platform, Addison Wesley
- Budi Kurniawan, *Java for the Web with Servlets, JSP, and EJB: A Developer's Guide to J2EE Solutions*, New Riders Publishing.
- Justin Couch and Daniel H. Steinberg, *Java 2 Enterprise Edition Bible,* Hungry Minds
- Marty Hall, *Core Servlets and JavaServer Pages (JSP),* Prentice Hall.

Reference websites:

- www.apl.jhu.edu
- www.java.sun.com
- www.novocode.com
- www.javaskyline.com
- www.stardeveloper.com