Copyright © Gaurav Saini. All rights reserved
Last Updated: 1st May, 2004
Email: gs_1481@yahoo.com

CONTENTS

Sr. No.	Topic
1.	**Java SDK**
1.1	Introduction
1.2	Installing and Configuring J2SDK
2.	**Apache-Jakarta-Tomcat web server**
2.1	Introduction
2.2	Installing and Configuring Tomcat
3.	**Servlet**
3.1	Introduction
3.2	Life Cycle of a Servlet
3.3	Servlet Implementation
3.4	Basic Servlet Template
3.5	Sample Servlets
3.6	Cookies and Session Tracking
3.6.1	Introduction to Cookies
3.6.2	Advantages of Cookies
3.6.3	Servlet Cookie API
3.6.4	Servlet and Cookies Example
3.6.5	Need for Session Tracking
3.6.6	Session Tracking API
3.6.7	Looking Up the HttpSession Object Associated with the Current Request
3.6.8	Looking Up Information Associated with a Session
3.6.9	Associating Information with a Session
3.6.10	Session Example
3.6.11	Terminating Sessions
3.6.12	Encoding URL's sent to a client
4.	**Java Beans**
4.1	Introduction
4.2	Java Bean Specification (features)
4.3	Naming Conventions
4.4	Sample Code
5.	**Java Server Pages**
5.1	Introduction
5.2	Advantages over Servlet
5.3	Types of elements with JSP
5.4	Examples

Copyright © Gaurav Saini. All rights reserved
Last Updated: 1st May, 2004
Email: gs_1481@yahoo.com

6.	App Server, Web Server: What's the difference?
6.1	The Web server
6.2	The application server
6.3	An example
6.3.1	Scenario 1: Web server without an application server
6.3.2	Scenario 2: Web server with an application server

INTORDUCTION TO SERVLETS AND JSP

1. Java SDK

1.1 Introduction

The Java SDK consists of all the API's needed for writing a full-fledged Java application. But here we are concerned with the API's used to write Servlet classes

1.2 Installing and Configuring J2SDK

Step 1: Get the Java 2 SDK from the Sun's site and install it in a suitable directory.

Step 2: Set the environment variable JAVA_HOME to the path of the directory into which you have installed SDK.

Step 3: Set the environment variable PATH to the path %PATH%;%JAVA_HOME%\bin

2. Apache-Jakarta-Tomcat web server

2.1 Introduction

This is a web-server that provides the containers for running Servlet and JSP.

2.2 Installing and Configuring Tomcat

Step 1: Get .zip file of tomcat binary from the Apache's site and extract all the files in a suitable directory.

Step 2: Set the environment variable CATALINA_HOME to the path of the directory into which you have installed tomcat.

Copyright © Gaurav Saini. All rights reserved
Last Updated: 1st May, 2004
Email: gs_1481@yahoo.com

3. Servlet

3.1 Introduction

Servlet is a container, which acts as a bridge between user and the web server. The user sends the request to the servlet through the browser and then the servlet takes appropriate actions and sends the response back to the browser. The response is generally a HTML page.

The jobs performed by the servlet can be summarized as:

a. **Read any data sent by the user.**
b. **Look up any other information about the request that is embedded in the HTTP request.**
c. **Generate the results.**
d. **Format the results inside a document.**
e. **Set the appropriate HTTP response parameters.**
f. **Send the document back to the client.**

Note:

Always remember that a servlet is loaded on the web server **only once,** the subsequent requests for the servlet are served by spawning threads from the original instance.

3.2 Life Cycle of a Servlet

The servlet life cycle comprises of three stages:

a. **public void init (ServletConfig sc)**
b. **public void service (ServletRequest req, ServletResponse res)**
c. **public void destroy ()**

3.3 Servlet Implementation

i. The servlet is a instance of a class that implements the **javax.servlet.Servlet** interface.

ii. The standard implementations of javax.servlet.Servlet are used i.e. **javax.servlet.GenericServlet** or **javax.servlet.http.HttpServlet.**

iii. GenericServlet implements Servlet interface.

iv. HttpServlet extends GenericServlet.

v. GenericServlet
   a. public void init (ServletConfig sc)

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

b. public void service (ServletRequest req, ServletResponse res) throws ServletException, IOException
c. public void destroy ()

vi. HttpServlet

a. doGet, if the servlet supports HTTP GET requests

protected void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException

b. doPost, for HTTP POST requests

protected void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException

c. init and destroy, to manage resources that are held for the life of the servlet

3.4 Basic Servlet Template

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTemplate extends HttpServlet
{
	public void doGet(HttpServletRequest req, HttpServletResponse res)
	throws ServletException, IOException
	{
		// Use "req" to read incoming HTTP headers
		// (e.g. cookies) and HTML form data (e.g. data the user
		// entered and submitted).

		// Use "res" to specify the HTTP response status
		// code and headers (e.g. the content type, cookies).

		PrintWriter out = res.getWriter();
		// Use "out" to send content to browser
	}
}
```

3.5 Sample Servlets

Activity 1: Running a simple servlet that displays Hello World!!!

Step 1: Write a HelloWorld class

Copyright © Gaurav Saini. All rights reserved
Last Updated: 1st May, 2004
Email: gs_1481@yahoo.com

HelloWorld.java

```
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><head><title>First Servlet</title></head>" );
		out.println("<body>");

		out.println("Hello World");

		out.println("</body></html>");

	}
}
```

Step 2: Before compiling the .java file, remember to copy the **servlet.jar** file to the **%JAVA_HOME%\lib** directory.

Step 3: Set the CLASSPATH to **%CLASSPATH%;%JAVA_HOME%\lib\servlet.jar**

Step 4: Compile the .java file. Copy the generated .class file to your web project classes folder in i.e. **%TOMCAT_HOME%\webapps\gaurav\WEB-INF\classes**

Step 5: Now create a **web.xml** file and place it in **%TOMCAT_HOME%\webapps\gaurav\WEB-INF** folder

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
	<servlet>
```

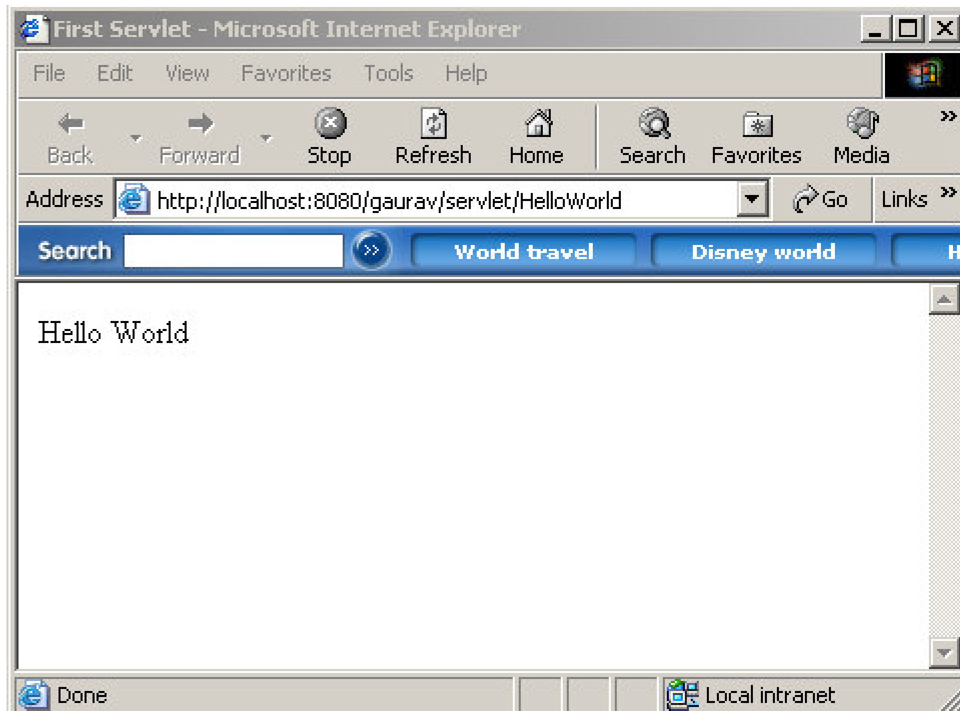
Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
        <servlet-name>Servlet1</servlet-name>
        <servlet-class>HelloWorld</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Servlet1</servlet-name>
        <url-pattern>/servlet/HelloWorld</url-pattern>
    </servlet-mapping>
</web-app>
```

Step 6: Run Tomcat

Step 7: Open the browser window and type:
http://localhost:8080/gaurav/servlet/HelloWorld



Activity 2: A servlet using Initialization Parameters

Step 1: Write a InitParamDemo class

InitParamDemo.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class InitParamDemo extends HttpServlet
```

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
{
        private String name;
        private int age;
        public void init(ServletConfig sc) throws ServletException
        {
                super.init(sc);
                name=sc.getInitParameter("name");
                age=Integer.parseInt(sc.getInitParameter("age"));
        }
        public void doGet(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException
        {
                res.setContentType("text/html");
                PrintWriter out = res.getWriter();
                out.println("<HTML>\n" +
                "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello</H1>\n" +
                "Hi!!!! " + name + ". Your age is: " + age +
                "</BODY></HTML>");
        }
}
```

Step 2: Compile the .java file. Now store the .class file in **%TOMCAT_HOME%\webapps\gaurav\WEB-INF\classes**

Step 3: Make Changes to web.xml as follows:

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
        <servlet>
                <servlet-name>Servlet1</servlet-name>
                <servlet-class>HelloWorld</servlet-class>
        </servlet>

        <servlet>
                <servlet-name>Servlet2</servlet-name>
                <servlet-class>InitParamDemo</servlet-class>
                <init-param>
                        <param-name>name</param-name>
                        <param-value>Gaurav Saini</param-value>
```

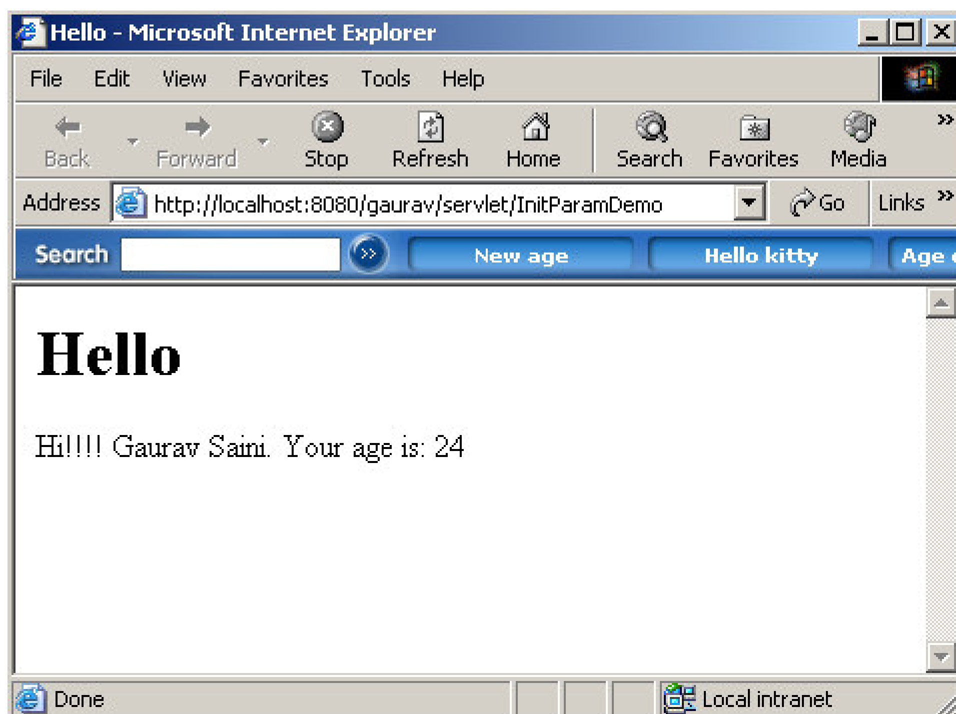
Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
            </init-param>
            <init-param>
                  <param-name>age</param-name>
                  <param-value>24</param-value>
            </init-param>
      </servlet>

      <servlet-mapping>
            <servlet-name>Servlet2</servlet-name>
            <url-pattern>/servlet/InitParamDemo</url-pattern>
      </servlet-mapping>

      <servlet-mapping>
            <servlet-name>Servlet1</servlet-name>
            <url-pattern>/servlet/HelloWorld</url-pattern>
      </servlet-mapping>
</web-app>
```

Step 4: Run tomcat

Step 5: Type the following URL in the browser window:
http://localhost:8080/gaurav/servlet/InitParamDemo



Activity 3: Retrieving form data from servlets

Step 1: Write a SubmitForm class

Copyright © Gaurav Saini. All rights reserved
Last Updated: 1st May, 2004
Email: gs_1481@yahoo.com

SubmitForm.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SubmitForm extends HttpServlet
{
	public void doGet(HttpServletRequest req, HttpServletResponse res)
	throws ServletException, IOException
	{
		res.setContentType("text/html");
		PrintWriter out = res.getWriter();
		out.println("<HTML>\n" +
		"<HEAD><TITLE>Read form data</TITLE></HEAD>\n" +
		"<BODY>\n" +
		"<H1>Hello</H1>\n" +
		"Hi!!!! " + req.getParameter("name") + ". Your age is: " +
		req.getParameter("age") + "</BODY></HTML>");
	}
	public void doPost(HttpServletRequest request, HttpServletResponse response)
	throws ServletException, IOException
	{
		doGet(request,response);
	}
}
```

Step 2: Compile the .java file. Now store the .class file in **%TOMCAT_HOME%\webapps\gaurav\WEB-INF\classes**

Step 3: Write the FormSubmit.html file as:

FormSubmit.html

```
<html>

<head>
<title>Read form data</title>
</head>

<body>

<form method="POST" action="/gaurav/servlet/SubmitForm">
  <table border="0" width="100%">
    <tr>
```

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
    <td width="30%"><b>Name</b></td>
    <td width="70%"><input type="text" name="name" size="20"></td>
   </tr>
   <tr>
    <td width="30%"><b>Age</b></td>
    <td width="70%"><input type="text" name="age" size="20"></td>
   </tr>
  </table>
  <p><input type="submit" value="Submit" name="B1"><input type="reset"
value="Reset" name="B2"></p>
</form>

</body>

</html>
```

Step 4: Make Changes to web.xml as follows:

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
	<servlet>
		<servlet-name>Servlet1</servlet-name>
		<servlet-class>HelloWorld</servlet-class>
	</servlet>

	<servlet>
		<servlet-name>Servlet2</servlet-name>
		<servlet-class>InitParamDemo</servlet-class>
		<init-param>
			<param-name>name</param-name>
			<param-value>Gaurav Saini</param-value>
		</init-param>
		<init-param>
			<param-name>age</param-name>
			<param-value>24</param-value>
		</init-param>
	   </servlet>

	<servlet>
		<servlet-name>Servlet3</servlet-name>
		<servlet-class>SubmitForm</servlet-class>
```

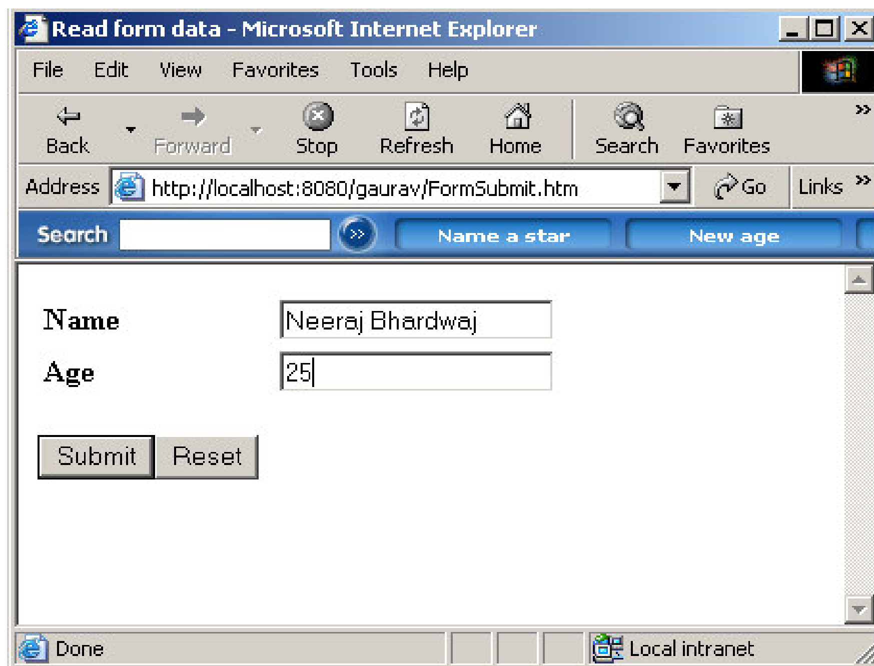
Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
    </servlet>

    <servlet-mapping>
        <servlet-name>Servlet3</servlet-name>
        <url-pattern>/servlet/SubmitForm</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>Servlet2</servlet-name>
        <url-pattern>/servlet/InitParamDemo</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>Servlet1</servlet-name>
        <url-pattern>/servlet/HelloWorld</url-pattern>
    </servlet-mapping>
</web-app>
```

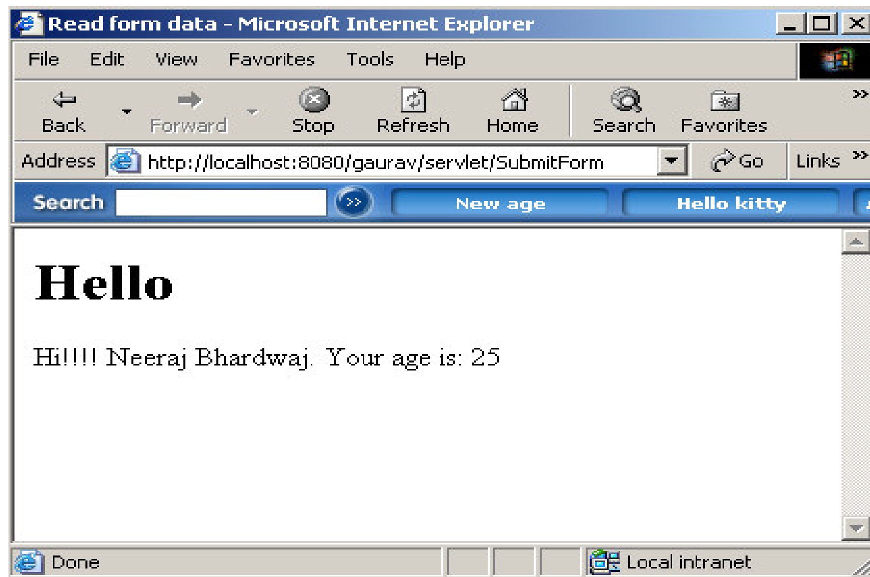
Step 5: Run tomcat

Step 6: Type the following URL in the browser window:
http://localhost:8080/gaurav/FormSubmit.htm



Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com



3.6 Cookies and Session Tracking

3.6.1 Introduction to Cookies

Cookies are small bits of information that a web server sends to the client and the client sends the same information unchanged back, when later visiting the same domain.

3.6.2 Advantages of Cookies

i. Identifying a user during an E-Com session.
ii. Avoiding User name and Password
iii. Customizing a site

Warning: Although cookies are not a serious security threat, but its recommended that not to use cookies when dealing with highly sensitive data e.g. Credit Card No.

3.6.3 Servlet Cookie API

i. Cookie() constructor

Cookie object-name = new Cookie (name, value)

where, name/value are String type and should not contain white space or any of the following characters [ ] ( ) = , " / ? @ : ;

ii. Attributes

Before adding a cookie to response headers you can use various **setXXX()** methods to set the attributes of that cookie. Later you can retrieve the cookie attributes from the request headers using **getXXX()** methods.

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

Method	Purpose
public String **getDomain** ( ) public void **setDomain** (String domain)	get or set the domain to which the cookie applies (default applies to the domain that sent the cookie)
public int **getMaxAge** ( ) public void **setMaxAge** (int age)	get or set how much time (in seconds) should elapse before the cookie expires.
public String **getName** ( ) public void **setName** (String cookieName)	get or the name of the cookie
public String **getValue** ( ) public void **setValue** (String cookieValue)	get or set the value of the cookie
public String **getPath** ( ) public void **setPath** (String value)	get or set the path to which the cookie applies (default applies to the URL in and below the directory containing the page that sent the cookie)
public boolean **getSecure** () public void **setSecure** (boolean setSecureFlag)	should the cookie be sent over SSL connection or not (default is false)

iii. Placing cookies in response headers

The cookie is inserted into a Set-Cookie HTTP response header by means of the addCookie method of HttpServletResponse.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
{
        .........
        Cookie userCookie = new ("userid", "user1234");
        userCookie.setMaxAge(365 * 24 * 60 *60); //for one year
        response.addCookie (userCookie);
        .........
}
```

iv. Reading cookies from client

To read the cookies that come back *from* the client, you call getCookies on the HttpServletRequest.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
```

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
        {
                ……….
                Cookie[] cookies = request.getCookies();
                Cookie cookie;
                for(int i=0; i<cookies.length; i++)
                {
                        cookie = cookies[i];
                        //now use the methods cookie.getName() or
                        //cookie.getValue()
                }
                ………
        }
```

3.6.4 Servlet and Cookies Example

Activity 4: Setting and Reading cookies

SetCookies.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SetCookies extends HttpServlet
{
        public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
        {
                for(int i=0; i<3; i++)
                {
                        //session cookie
                        Cookie cookie = new Cookie("Session-Cookie " + i,"Cookie-
                        Value-S" + i);
                        response.addCookie(cookie);
                        //persistent cookie
                        cookie = new Cookie("Persistent-Cookie " + i, "Cookie-Value-P"
                        + i);
                        cookie.setMaxAge(3600);//for 1 hour
                        response.addCookie(cookie);
                }
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                String title = "Setting Cookies";
                out.println ("<HTML><HEAD><TITLE>" + title +
                "</TITLE></HEAD>" +
                "<BODY>\n" +
```

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "There are six cookies associated with this page.\n" +
                "To see them, visit the\n" +
                "<A HREF=\"ShowCookies\">\n" +
                "<CODE>ShowCookies</CODE> servlet</A>.\n" +
                "<P>\n" +
                "Three of the cookies are associated only with the\n" +
                "current session, while three are persistent.\n" +
                "Quit the browser, restart, and return to the\n" +
                "<CODE>ShowCookies</CODE> servlet to verify that\n" +
                "the three long-lived ones persist across sessions.\n" +
                "</BODY></HTML>");
        }
}
```

ShowCookies.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ShowCookies extends HttpServlet
{
        public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
        {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                String title = "Active Cookies";
                out.println("<HTML><HEAD><TITLE>" + title +
                "</TITLE></HEAD>" +
                "<BODY>\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
                "<TR>\n" +
                " <TH>Cookie Name\n" +
                " <TH>Cookie Value");

                Cookie[] cookies = request.getCookies();
                Cookie cookie;

                for(int i=0; i<cookies.length; i++)
                {
                        cookie = cookies[i];
                        out.println("<TR>\n" +
                        " <TD>" + cookie.getName() + "\n" +
```

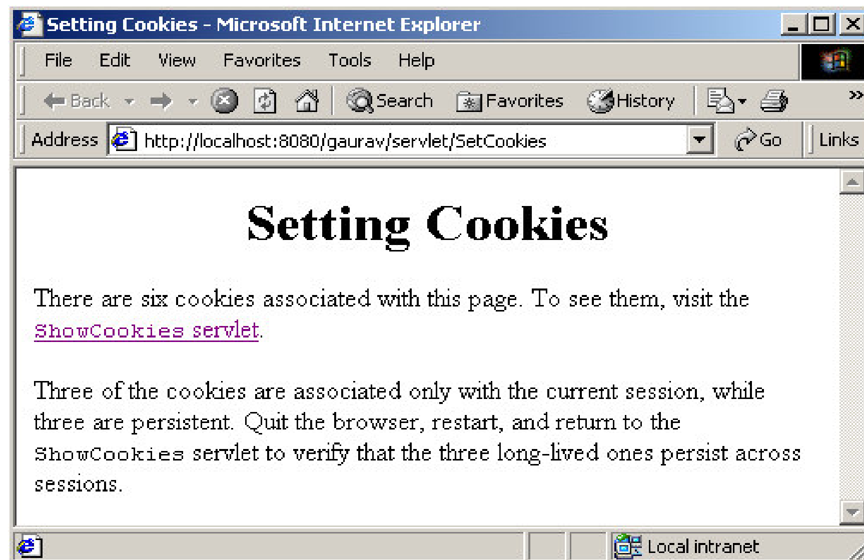
Copyright © Gaurav Saini. All rights reserved
Last Updated: 1st May, 2004
Email: gs_1481@yahoo.com

```
                        " <TD>" + cookie.getValue());
                }
                out.println("</TABLE></BODY></HTML>");
        }
}
```

Output:

i. Open the browser and type the URL as

http://localhost:8080/gaurav/servlet/SetCookies



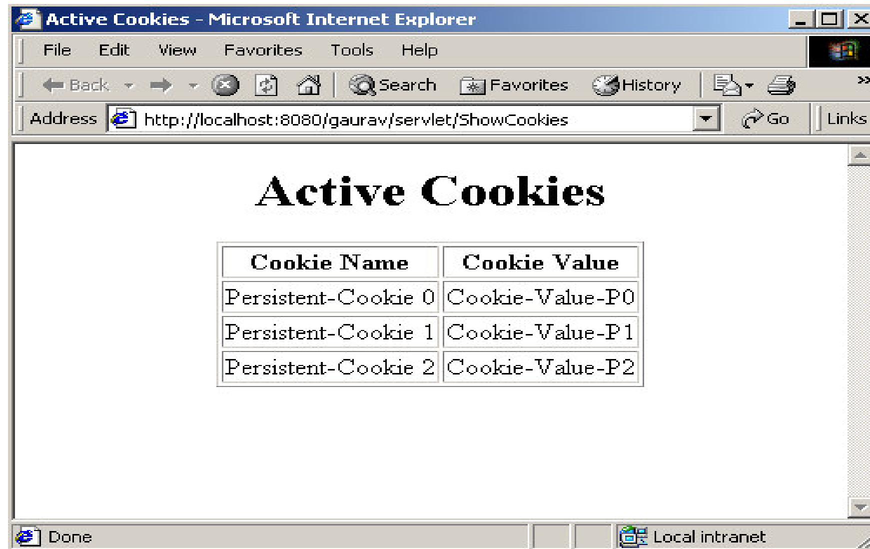
ii. Click the hyperlink to view all the cookies set by the servlet. Now the following output is obtained.

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

iii. Now close the browser. Open a new browser window and type the URL as:

http://localhost:8080/gaurav/servlet/ShowCookies



3.6.5 Need for Session Tracking

HTTP is a "stateless" protocol: each time a client retrieves a Web page, it opens a separate connection to the Web server, and the server does not automatically maintain contextual information about a client.

There are three typical solutions to this problem:

i. Cookies
ii. URL-rewriting, and
iii. Hidden form fields.

These methods have their own drawbacks and hence are not in much use. Generally we use a more enhanced methodology provided by the Servlet's **HttpSession** API.

This high-level interface is built on top of cookies or URL-rewriting.

In fact, most servers use cookies if the browser supports them, but automatically revert to URL-rewriting when cookies are unsupported or explicitly disabled. But, the servlet author doesn't need to bother with many of the details, doesn't have to explicitly manipulate cookies or information appended to the URL, and is automatically given a convenient place to store arbitrary objects that are associated with each session.

3.6.6 Session Tracking API

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

i. Look up for the session object associated with the current request.
ii. Create a new session object when necessary.
iii. Look up information associated with a session.
iv. Storing information in a session.
v. Discarding completed or abandoned sessions.

3.6.7 Looking Up the HttpSession Object Associated with the Current Request

i. You look up the **HttpSession** object by calling the **getSession** method of **HttpServletRequest**.

ii. If getSession returns null, this means that the user is not already participating in a session, so you can create a new session. The new session is created as:

HttpSession session = request.getSession(true);

3.6.8 Looking Up Information Associated with a Session

i. HttpSession objects live on the server.

ii. You use **session.getAttribute("attribute")** to look up a previously stored value.

iii. The return type is Object, so you have to do a typecast to whatever more specific type of data was associated with that attribute name in the session.

iv. The return value is null if there is no such attribute, so you need to check for null before calling methods on objects associated with sessions.

v. This is how its done:

```
HttpSession session = request.getSession(true);
ShoppingCart cart = (ShoppingCart) session.getAttribute ("shoppingCart");
if (cart == null)
{ // No cart already in session
    cart = new ShoppingCart();
    session.putValue("shoppingCart", cart);
}
//doSomethingWith(cart);
```

3.6.9 Associating Information with a Session

i. To specify information, you use setAttribute, supplying a key and a value.

ii. This is how its done:

```
HttpSession session = request.getSession(true);
```

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
        session.setAttribute ("referringPage", request.getHeader("Referer"));
        ShoppingCart cart = (ShoppingCart) session.getAttribute ("previousItems");

        if (cart == null)
        { // No cart already in session
            cart = new ShoppingCart();
            session.setAttribute("previousItems", cart);
        }

        String itemID = request.getParameter("itemID");

        if (itemID != null)
        {
            cart.addItem(Catalog.getItem(itemID));
        }
```

3.6.10 Session Example

Activity 5: Session Example

SessionExample.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class SessionExample extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String title = "Session Tracking Example";

        HttpSession session = request.getSession(true);
        String heading;

        Integer accessCount = (Integer)session.getAttribute("accessCount");

        if (accessCount == null)
        {
            accessCount = new Integer(0);
            heading = "Welcome, Newcomer";
```

Copyright © Gaurav Saini. All rights reserved

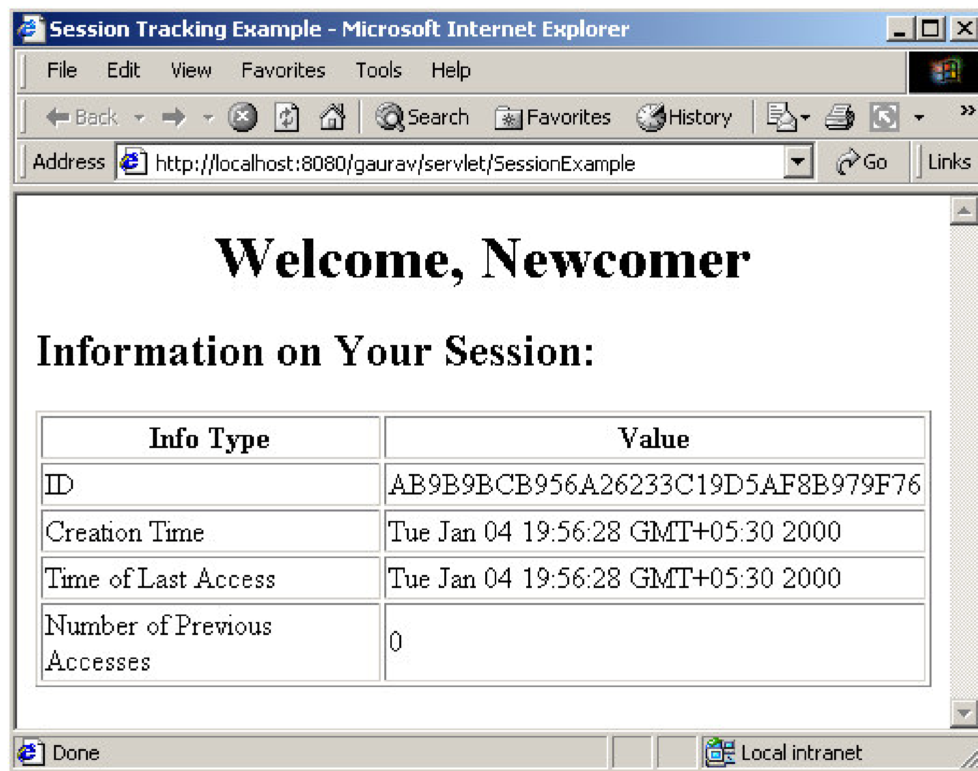
Email: gs_1481@yahoo.com

```
        }
        else
        {
                heading = "Welcome Back";
                accessCount = new Integer(accessCount.intValue() + 1);
        }

        session.setAttribute("accessCount", accessCount);

        out.println("<HTML><HEAD><TITLE>" + title + "</TITLE></HEAD>"
        + "<BODY>\n" +
        "<H1 ALIGN=\"CENTER\">" + heading + "</H1>\n" +
        "<H2>Information on Your Session:</H2>\n" +
        "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
        "<TR>\n" +
        " <TH>Info Type<TH>Value\n" +
        "<TR>\n" +
        " <TD>ID\n" +
        " <TD>" + session.getId() + "\n" +
        "<TR>\n" +
        " <TD>Creation Time\n" +
        " <TD>" +
        new Date(session.getCreationTime()) + "\n" +
        "<TR>\n" +
        " <TD>Time of Last Access\n" +
        " <TD>" +
        new Date(session.getLastAccessedTime()) + "\n" +
        "<TR>\n" +
        " <TD>Number of Previous Accesses\n" +
        " <TD>" + accessCount + "\n" +
        "</TABLE>\n" +
        "</BODY></HTML>");
    }

    // Handle GET and POST requests identically.
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        doGet(request, response);
    }
}
```

Output:

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com



3.6.11 Terminating Sessions

i. Sessions will automatically become inactive when the amount of time between client accesses exceeds the interval specified by **getMax-InactiveInterval**. When this happens, any objects bound to the HttpSession object automatically get unbound.

ii. Rather than waiting for sessions to time out, you can explicitly deactivate a session through the use of the session's **invalidate** method.

3.6.12 Encoding URL's sent to a client

If you are using URL-rewriting for session tracking and you send a URL that references your site to the client, you need to explicitly add on the session data.

There are two possible places you might use URLs that refer to your own site:

i. The first is when the URLs are embedded in the Web page that the servlet generates.

These URLs should be passed through the encodeURL method of HttpServletResponse.

Here's an example:

Copyright © Gaurav Saini. All rights reserved


```
String originalURL = someRelativeOrAbsoluteURL;
String encodedURL = response.encodeURL (originalURL);
out.println ("<A HREF=\"" + encodedURL + "\">...</A>");
```

ii. **The second place you might use a URL that refers to your own site is in a sendRedirect call (i.e., placed into the Location response header).**

Fortunately, HttpServletResponse supplies an encodeRedirectURL method to handle that case.

Here's an example:

```
String originalURL = someURL;
String encodedURL = response.encodeRedirectURL (originalURL);
response.sendRedirect (encodedURL);
```

4. Java Beans

4.1 Introduction

i. A Java bean is a reusable software component that can be manipulated visually in a builder tool.
ii. A java bean is simply a regular java class **(should have no-arg constructor)** designed according to a set of specifications.
iii. A java bean can be faceless or GUI based.

4.2 Java Bean Specification (features)

i. **Introspection** (analysis by a builder that how a bean works)

ii. **Customization** (the user can customize the appearance and behavior of a bean using a builder)

iii. **Events** (notifying a bean)

iv. **Properties** (customized through a tool and for programmatic use)

v. **Persistence** (saving and reloading states; a bean should implement **java.io.Serializable**)

4.3 Naming Conventions

i. Accessor methods (getter and setter methods; composed of the prefix **get** and **set** respectively, plus the property name with the first character of each word capitalized)

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

ii. A readable property has getter methods and a writable property has setter methods.

iii. Depending on the combination of getter-setter a property is:
   a. Read-only
   b. Write-only
   c. Read/Write

4.4 Sample Code

```
public class DemoBean
{
        private String firstName;
        private int[] marks;
        private boolean status;

        //setter methods
        public void setFirstName (String firstName)
        {
                this.firstName = firstName;
        }
        //indexed property can be set in any of the ways
        public void setMarks (int[] marks)
        {
                this.marks = marks;
        }
        public void setMarks (int index, int value)
        {
                marks[index] = value;
        }
        public void setStatus (boolean status)
        {
                this.status = status;
        }

        //getter methods
        public String getFirstName ()
        {
                return firstName;
        }
        public int[] getMarks ()
        {
                return marks;
        }
        public int getMarks (int index)
        {
```

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
            return marks[index];
        }
        //recommended way to read boolean values
        public boolean isStatus ()
        {
            return status;
        }
}
```

Note: A read-only property doesn't necessarily have to match an instance variable exactly. Instead, it can combine instance variable values, or any values, and return a computed value.

5. Java Server Pages

5.1 Introduction

i. JSP is an extension to the Java Servlet technology.

ii. JSP = Static HTML + Dynamic content (special Tags).

iii. Typically each JSP page is first translated into the Java source code for a servlet class, and then that servlet is compiled. The resulting servlet is invoked to handle all requests for that page. Typically, the **page translation step is performed the first time a given JSP page is requested**, and then only when that page's source code changes thereafter. Otherwise, the resulting servlet is simply executed, providing very quick delivery of content to the user.

iv. No compiling, no packages and no CLASSPATH setting is required for running a JSP page.

5.2 Advantages over Servlet

a. vs. Active Server Pages (ASP).

i. In JSP, the dynamic part is written in Java, not Visual Basic or other MS-specific language, so it is more powerful and easier to use.

ii. JSP is portable to other operating systems and non-Microsoft Web servers.

b. vs. Pure Servlets.

i. With JSP it is more convenient to write (and to modify!) regular HTML than to have a zillion println statements that generate the HTML.

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

ii. **JSP separates the look from the content** you can put different people on different tasks: your Web page design experts can build the HTML, leaving places for your servlet programmers to insert the dynamic content.

c. vs. Server-Side Includes (SSI).

i. JSP lets you use servlets instead of a separate program to generate that dynamic part.

ii. SSI is only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

d. vs. JavaScript.

i. JavaScript only handles situations where the dynamic information is based on the client's environment.

ii. Cookies, HTTP and form submission data is not available to JavaScript.

iii. JavaScript runs on the client, it can't access server-side resources like databases, catalogs, pricing information, and the like.

e. vs. Static HTML.

i. Regular HTML, of course, cannot contain dynamic information.

ii. JSP is so easy and convenient that it is quite feasible to augment HTML pages that only benefit marginally by the insertion of small amounts of dynamic data.

5.3 Types of elements with JSP

i. Directives
ii. Action Elements
iii. Scripting Elemets

i. Directives

Directive	**Purpose**
<%@ page …%>	Controls properties of the JSP page
<%@ include …%>	Includes the contents of a file into the JSP page at **translation time**
<%@ taglib …%>	Makes a custom tag library available within the including page

Copyright © Gaurav Saini. All rights reserved
Last Updated: 1st May, 2004
Email: gs_1481@yahoo.com

ii. Action elements

Element	Description
<jsp:useBean>	Makes a JavaBeans component available in a page.
<jsp:getProperty>	Gets a property value from a JavaBeans component and adds it to the response.
<jsp:setProperty>	Sets a JavaBeans property value.
<jsp:include>	Includes the response from a servlet or JSP page during the **request processing** phase.
<jsp:forward>	Forwards the processing of a request to a servlet or JSP page.
<jsp:param>	Adds a parameter value to a request handed off to another servlet or JSP page using <jsp:include> or <jsp:forward>.
<jsp:plugin>	Generates HTML that contains the appropriate client browser-dependent elements (OBJECT or EMBED) needed to execute an Applet with the Java Plugin software.

iii. Scripting elements

a. **Expressions** of the form <%= expression %>
b. **Scriptlets** of the form <% code %>
c. **Declarations** of the form <%! code %>

Note: Difference between include directive and jsp:include.

5.4 Examples

Activity1: Simple JSP page

FirstJsp.jsp

```
<%@ page import="java.util.Date" %>
<%!
	private static String loadTime = new Date().toString();
	private static String getLoadTime() { return loadTime; }
```

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
	private static String getCurrentTime() { return new Date().toString(); }
%>
<html>
	<head><title>JSP Example</title></head>
<body>
	This page was loaded into memory at <%= getLoadTime() %>.<br>
	The current time is <%= getCurrentTime() %><br>
	<%if (Math.random() < 0.5) { %>
		Have a <b>nice</b> day!
	<% } else { %>
		Have a <b>lousy</b> day!
	<% } %>
</body>
</html>
```

Activity 2: Combining JavaBeans and JSP

BeanDemo.jsp

```
package bean;
public class BeanDemo
{
	private String name;
	private String age;

	public void setName(String name)
	{
		this.name=name;
	}
	public void setAge(String age)
	{
		this.age=age;
	}

	public String getName()
	{
		return name;
	}
	public String getAge()
	{
		return age;
	}
}
```

Entry.jsp

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

```
<%@ page language="java" contentType="text/html" %>
<html>
<head>
	<title>Bean Demo</title>
</head>
<body>
	<form method="POST" action="/gaurav/Demo1.jsp">
	<table border="0" width="100%">
	<tr>
		<td width="30%"><b>Name</b></td>
		<td width="70%"><input type="text" name="name" size="20"></td>
	</tr>
	<tr>
		<td width="30%"><b>Age</b></td>
		<td width="70%"><input type="text" name="age" size="20"></td>
	</tr>
	</table>
	<p><input type="submit" value="Submit" name="B1">
	<input type="reset" value="Reset" name="B2"></p>
	</form>
</body>
</html>
```

Welcome.jsp

```
<%@ page language="java" contentType="text/html" %>
<html>
<head>
	<title>Bean Demo</title>
</head>
<body>
	<jsp:useBean id="obj1" class="bean.BeanDemo">
		<jsp:setProperty name="obj1" property="*"/>
	</jsp:useBean>
	Hi!!!  <jsp:getProperty name="obj1" property="name"/>
	<p>Your age is  <jsp:getProperty name="obj1" property="age"/>
</body>
</html>
```

6. App Server, Web Server: What's the difference?

A Web server serves pages for viewing in a Web browser, while an application server provides methods that client applications can call. A little more precisely, we can say that:

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

"A Web server exclusively handles HTTP requests, whereas an application server serves business logic to application programs through any number of protocols. "

6.1 The Web server

A Web server handles the HTTP protocol. When the Web server receives an HTTP request, it responds with an HTTP response, such as sending back an HTML page. To process a request, a Web server may respond with a static HTML page or image, send a redirect, or delegate the dynamic response generation to some other program such as CGI scripts, JSPs (JavaServer Pages), servlets, ASPs (Active Server Pages), server-side JavaScripts, or some other server-side technology. Whatever their purpose, such server-side programs generate a response, most often in HTML, for viewing in a Web browser.

A Web server's delegation model is fairly simple. When a request comes into the Web server, the Web server simply passes the request to the program best able to handle it. The Web server doesn't provide any functionality beyond simply providing an environment in which the server-side program can execute and pass back the generated responses. The server-side program usually provides for itself such functions as transaction processing, database connectivity, and messaging.

While a Web server may not itself support transactions or database connection pooling, it may employ various strategies for fault tolerance and scalability such as load balancing, caching, and clustering—features oftentimes erroneously assigned as features reserved only for application servers.

6.2 The application server

As for the application server, according to the definition, an application server exposes business logic to client applications through various protocols, possibly including HTTP. While a Web server mainly deals with sending HTML for display in a Web browser, an application server provides access to business logic for use by client application programs. The application program can use this logic just as it would call a method on an object (or a function in the procedural world).

Such application server clients can include GUIs (graphical user interface) running on a PC, a Web server, or even other application servers. The information traveling back and forth between an application server and its client is not restricted to simple display markup. Instead, the information is program logic. Since the logic takes the form of data and method calls and not static HTML, the client can employ the exposed business logic however it wants.

In most cases, the server exposes this business logic through a component API, such as the EJB (Enterprise JavaBean) component model found on J2EE (Java 2 Platform, Enterprise Edition) application servers. Moreover, the application server manages its own resources. Such gate-keeping duties include security, transaction

processing, resource pooling, and messaging. Like a Web server, an application server may also employ various scalability and fault-tolerance techniques.

6.3 An example

As an example, consider an online store that provides real-time pricing and availability information. Most likely, the site will provide a form with which you can choose a product. When you submit your query, the site performs a lookup and returns the results embedded within an HTML page. The site may implement this functionality in numerous ways.

6.3.1 Scenario 1: Web server without an application server

In the first scenario, a Web server alone provides the online store's functionality. The Web server takes your request, then passes it to a server-side program able to handle the request. The server-side program looks up the pricing information from a database or a flat file. Once retrieved, the server-side program uses the information to formulate the HTML response, then the Web server sends it back to your Web browser.

To summarize, a Web server simply processes HTTP requests by responding with HTML pages.

6.3.2 Scenario 2: Web server with an application server

Scenario 2 resembles Scenario 1 in that the Web server still delegates the response generation to a script. However, we can now put the business logic for the pricing lookup onto an application server. With that change, instead of the script knowing how to look up the data and formulate a response, the script can simply call the application server's lookup service. The script can then use the service's result when the script generates its HTML response.

In this scenario, the application server serves the business logic for looking up a product's pricing information. That functionality doesn't say anything about display or how the client must use the information. Instead, the client and application server send data back and forth. When a client calls the application server's lookup service, the service simply looks up the information and returns it to the client.

By separating the pricing logic from the HTML response-generating code, the pricing logic becomes far more reusable between applications. A second client, such as a cash register, could also call the same service as a clerk checks out a customer. In

contrast, in Scenario 1 the pricing lookup service is not reusable because the information is embedded within the HTML page.

Copyright © Gaurav Saini. All rights reserved

Email: gs_1481@yahoo.com

To summarize, in Scenario 2's model, the Web server handles HTTP requests by replying with an HTML page while the application server serves application logic by processing pricing and availability requests.

Note:

Recently, XML Web services have blurred the line between application servers and Web servers. By passing an XML payload to a Web server, the Web server can now process the data and respond much as application servers have in the past.

Additionally, most application servers also contain a Web server, meaning you can consider a Web server a subset of an application server. While application servers contain Web server functionality, developers rarely deploy application servers in that capacity. Instead, when needed, they often deploy standalone Web servers in tandem with application servers. Such a separation of functionality aids performance (simple Web requests won't impact application server performance), deployment configuration (dedicated Web servers, clustering, and so on), and allows for best-of-breed product selection.