Module 4

Background; The Life Cycle of a Servlet; Using Tomcat for Servlet Development; A simple Servlet; The Servlet API; The Javax.servlet Package; Reading Servlet Parameter; The Javax.servlet.http package; Handling HTTP Requests and Responses; Using Cookies; Session Tracking. Java Server Pages (JSP): JSP, JSP Tags, Tomcat, Request String, User Sessions, Cookies, Session Objects

For Servlets

http://tutorials.jenkov.com/java-servlets/index.html Java Servlets overview

http://tutorials.jenkov.com/java-web-apps/directory-layout.html Slide 9

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session Slide 31, 37 HTTP Session

Servlets : Introduction

13. Define the following: i) Servlet ii) MIME

Servlets are programs that execute on the server side of a connection.

Applets dynamically extend the functionality of a web browser, servlets dynamically extend the functionality of a web server.

Servlets need the understanding of how web browsers and servers cooperate to provide content to a user.

Consider a request for a web page. A user enters a **Uniform Resource Locator (URL)** into a browser. The browser generates an HTTP request to the appropriate web server.

The web server maps this request to a specific file. That file is returned in an HTTP response manner to the browser.

The HTTP header in the response indicates the type of the content.

Servlets : Introduction

Multipurpose Internet Mail Extensions (MIME) are used for this purpose.

Ex: The Hypertext Markup Language (HTML) source code of a web page has a MIME type of text/html.

The dynamic web pages generate contents to reflect the latest information.

Servlets offer several advantages in comparison with CGI (Common Gateway Interface).

First, performance is significantly better. Servlets execute within the address space of a web server. It is not necessary to create a separate process to handle each client request.

Second, servlets are platform-independent because they are written in Java.

Third, the Java security manager on the server enforces a set of restrictions to protect the resources on a server machine.

Servlets : Introduction

Finally, the full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms.

14. Explain the life cycle of a servlet.

The Life Cycle of a Servlet

Three methods are central to the life cycle of a servlet. **init( )**, **service( )**, and **destroy( )**.

They are implemented by every servlet and are invoked at specific times by the server.

Consider a typical user scenario to understand when these methods are called.
First, assume that a user enters a Uniform Resource Locator (URL) to a web browser.
The browser then generates an HTTP request for this URL.
This request is then sent to the appropriate server.

Second, this HTTP request is received by the web server.
The server maps this request to a particular servlet.
The servlet is dynamically retrieved and loaded into the address space of the server.

Servlets : Introduction

Third, the server invokes the **init( )** method of the servlet.
This method is invoked only when the servlet is first loaded into memory.
It is possible to pass initialization parameters to the servlet so it may configure itself.

Fourth, the server invokes the **service( )** method of the servlet.
This method is called to process the HTTP request.
It is possible for the servlet to read data that has been provided in the HTTP request.
It may also formulate an HTTP response for the client.

The servlet remains in the server's address space and is available to process any other HTTP requests received from clients. **service( )** method is called for each HTTP request.

Reading Servlet Parameters

```
        // Get print writer.
        PrintWriter pw = response.getWriter();
        // Get enumeration of parameter names.
        Enumeration e = request.getParameterNames();
        // Display parameter names and values.
        while(e.hasMoreElements()) {
            String pname = (String)e.nextElement();
            pw.print(pname + " = ");
            String pvalue = request.getParameter(pname);
            pw.println(pvalue);
        }
        pw.close();
    }
}
```

Servlets : Background

Finally, the server may decide to unload the servlet from its memory. The algorithms by which this determination is made are specific to each server. The server calls the **destroy( )** method to relinquish any resources such as file handlers that are allocated for the servlet. Important data may be saved to a persistent storage. The memory allocated for the servlet and its objects will be garbage collected.

Using Tomcat for Servlet Development

A servlet development environment is to create and execute servlets and Tomcat is considered for the same.

Tomcat is an open-source product developed and maintained by Jakarta Project of the Apache Software Foundation.

It contains the class libraries, documentation, and run-time support that is needed to create and test servlets.

Servlets : Background

First step will be to create the .java file which contains the servlet code.

This file has to be compiled as shown below.

Ex: javac HelloServlet.java -classpath "C:\Program Files\Apache Software Foundation\Tomcat 5.5\common\lib\servlet-api.jar"

Once the servlet is compiled, Tomcat must be able to find it, which will be done, by copying the file into a directory under Tomcat's **webapps** directory and entering its name into a **web.xml** file.

First, copy the servlet class file into the following directory:

C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\servlets-examples\WEB-INF\classes

Next, add the servlet name and mapping to the **web.xml** file in the following directory:

C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\servlets-examples\WEB-INF

Servlets : Background

For instance, assuming the first example, called **HelloServlet**, add the following lines in the section that defines the servlets:

```
<servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
</servlet>
```

Next, add the following lines to the section that defines the servlet mappings.

```
<servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/servlet/HelloServlet</url-pattern>
</servlet-mapping>
```

Same general procedure must be followed for all of the examples.

Servlets : A Simple Servlet

The basic steps are the following:

1. Create and compile the servlet source code. Then, copy the servlet class file to the proper directory, and add the servlet name and mappings to the proper **web.xml** file.
2. Start Tomcat.
3. Start a web browser and request the servlet.

To begin, a file named **HelloServlet.java** must be created that contains the following program:

```
import java.io.*;
import javax.servlet.*;
public class HelloServlet extends GenericServlet   {
 public void service(ServletRequest request, ServletResponse response)
                                                  throws ServletException, IOException   {
     response.setContentType("text/html");
     PrintWriter pw = response.getWriter();
     pw.println("<B>Hello!");
     pw.close();
  }   }
```

Servlets : A Simple Servlet

Program imports the **javax.servlet** package.

This package contains the classes and interfaces required to build servlets.

Next, the program defines **HelloServlet** as a subclass of **GenericServlet**.

The **GenericServlet** class provides functionality that simplifies the creation of a servlet.
For example, it provides versions of **init( )** and **destroy( )**, which may be used as is.
Only the **service( )** method has to be implemented.

Inside **HelloServlet**, the **service( )** method (which is inherited from **GenericServlet**) is overridden.

This method handles requests from a client.

The first argument is a **ServletRequest** object. This enables the servlet to read data that is provided via the client request.

Servlets : A Simple Servlet

The second argument is a **ServletResponse** object. This enables the servlet to formulate a response for the client.

The call to **setContentType( )** establishes the MIME type of the HTTP response. In this program, the MIME type is text/html. This indicates that the browser should interpret the content as HTML source code.

Next, the **getWriter( )** method obtains a **PrintWriter** instance. Anything written to this stream is sent to the client as part of the HTTP response.

Then **println( )** is used to write some simple HTML source code as the HTTP response.

Compile this source code and place the **HelloServlet.class** file in the proper Tomcat directory. Also, add **HelloServlet** to the **web.xml** file, as described earlier.

Servlets : A Simple Servlet

Start Tomcat

Tomcat must be running before a servlet is executed.

Start a Web Browser and Request the Servlet

Start a web browser and enter the URL as below:

http://localhost:8080/servlets-examples/HelloServlet

Alternatively, URL can be entered as below:

http://127.0.0.1:8080/servlets-examples/HelloServlet

This can be done because 127.0.0.1 is defined as the IP address of the local machine.

Output of the servlet will be seen in the browser display area, which will contain the string **Hello!** in bold type.

The Servlet API

Two packages contain the classes and interfaces that are required to build servlets.

These are **javax.servlet** and **javax.servlet.http**. They constitute the Servlet API.
(These packages are not part of the Java core packages. Instead, they are extensions provided by Tomcat.)

The javax.servlet Package

This package contains a number of interfaces and classes that establishes the framework in which servlets operate.

The following table summarizes the core interfaces that are provided in this package.

Interface	Description
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.

The Servlet API

The most significant of these is **Servlet**. All servlets must implement this interface or extend a class that implements **Servlet** interface.

The **ServletRequest** and **ServletResponse** interfaces are also important.

The following table summarizes the core classes that are provided in the **javax.servlet** package:

Class	Description
GenericServlet	Implements the **Servlet** and **ServletConfig** interfaces.
ServletInputStream	Provides an input stream for reading requests from a client.
ServletOutputStream	Provides an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable.

The Servlet interface

All servlets must implement the **Servlet** interface.

It declares the **init( )**, **service( )**, and **destroy( )** methods that are called by the server during the life cycle of a servlet.

A method is also provided that allows a servlet to obtain any initialization parameters. The methods defined by **Servlet** are shown in the table below. (next slide)

The **getServletConfig( )** method is called by the servlet to obtain initialization parameters.

A servlet developer overrides the **getServletInfo( )** method to provide a string with useful information (for example, author, version, date, copyright). This method is also invoked by the server.

The Servlet interface

Method	Description
void destroy( )	Called when the servlet is unloaded.
ServletConfig getServletConfig( )	Returns a **ServletConfig** object that contains any initialization parameters.
String getServletInfo( )	Returns a string describing the servlet.
void init(ServletConfig *sc*) throws ServletException	Called when the servlet is initialized. Initialization parameters for the servlet can be obtained from *sc*. An **UnavailableException** should be thrown if the servlet cannot be initialized.
void service(ServletRequest *req*, ServletResponse *res*) throws ServletException, IOException	Called to process a request from a client. The request from the client can be read from *req*. The response to the client can be written to *res*. An exception is generated if a servlet or IO problem occurs.

The ServletConfig Interface

The **ServletConfig** interface allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are summarized here:

Method	Description
ServletContext getServletContext( )	Returns the context for this servlet.
String getInitParameter(String *param*)	Returns the value of the initialization parameter named *param*.
Enumeration getInitParameterNames( )	Returns an enumeration of all initialization parameter names.
String getServletName( )	Returns the name of the invoking servlet.

The ServletContext Interface

The ServletContext interface enables servlets to obtain information about their environment. Several of its methods are summarized in the table.

Method	Description
Object getAttribute(String *attr*)	Returns the value of the server attribute named *attr*.
String getMimeType(String *file*)	Returns the MIME type of *file*.
String getRealPath(String *vpath*)	Returns the real path that corresponds to the virtual path *vpath*.
String getServerInfo( )	Returns information about the server.
void log(String *s*)	Writes *s* to the servlet log.
void log(String *s*, Throwable *e*)	Writes *s* and the stack trace for *e* to the servlet log.
void setAttribute(String *attr*, Object *val*)	Sets the attribute specified by *attr* to the value passed in *val*.

TABLE 31-2 Various Methods Defined by **ServletContext**

The ServletRequest Interface

The ServletRequest interface enables a servlet to obtain information about a client request. Several of its methods are summarized in the table.

Method	Description
Object getAttribute(String *attr*)	Returns the value of the attribute named *attr*.
String getCharacterEncoding( )	Returns the character encoding of the request.
int getContentLength( )	Returns the size of the request. The value –1 is returned if the size is unavailable.
String getContentType( )	Returns the type of the request. A **null** value is returned if the type cannot be determined.
ServletInputStream getInputStream( ) throws IOException	Returns a **ServletInputStream** that can be used to read binary data from the request. An **IllegalStateException** is thrown if **getReader( )** has already been invoked for this request.
String getParameter(String *pname*)	Returns the value of the parameter named *pname*.
Enumeration getParameterNames( )	Returns an enumeration of the parameter names for this request.
String[ ] getParameterValues(String *name*)	Returns an array containing values associated with the parameter specified by *name*.

The ServletRequest Interface

String getProtocol( )	Returns a description of the protocol.
BufferedReader getReader( ) throws IOException	Returns a buffered reader that can be used to read text from the request. An **IllegalStateException** is thrown if **getInputStream( )** has already been invoked for this request.
String getRemoteAddr( )	Returns the string equivalent of the client IP address.
String getRemoteHost( )	Returns the string equivalent of the client host name.
String getScheme( )	Returns the transmission scheme of the URL used for the request (for example, "http", "ftp").
String getServerName( )	Returns the name of the server.
int getServerPort( )	Returns the port number.

TABLE 31-3 Various Methods Defined by **ServletRequest**

The ServletResponse Interface

The ServletResponse interface enables a servlet to formulate a response for a client. Several of its methods are summarized in the table.

Method	Description
String getCharacterEncoding( )	Returns the character encoding for the response.
ServletOutputStream getOutputStream( ) throws IOException	Returns a **ServletOutputStream** that can be used to write binary data to the response. An **IllegalStateException** is thrown if **getWriter( )** has already been invoked for this request.
PrintWriter getWriter( ) throws IOException	Returns a **PrintWriter** that can be used to write character data to the response. An **IllegalStateException** is thrown if **getOutputStream( )** has already been invoked for this request.
void setContentLength(int *size*)	Sets the content length for the response to *size*.
void setContentType(String *type*)	Sets the content type for the response to *type*.

TABLE 31-4 Various Methods Defined by **ServletResponse**

The GenericServlet Class

The GenericServlet class provides implementations of the basic life cycle methods for a servlet. **GenericServlet** implements the **Servlet** and **ServletConfig** interfaces.

In addition to these, a method to append a string to the server log file is available.

The signatures of these methods are :

void log(String s)

void log(String s, Throwable e)

Here, s is the string to be appended to the log, and e is an exception that may occur.

The ServletInputStream Class

The **ServletInputStream** class extends **InputStream**.

It is implemented by the servlet container and provides an input stream that a servlet developer can use to read the data from a client request.

The ServletInputStream Class

It defines the default constructor. In addition, a method is provided to read bytes from the stream.

int readLine(byte[ ] buffer, int offset, int size) throws IOException

Here, buffer is the array into which size bytes are placed starting at offset. The method returns the actual number of bytes read or –1 if an end-of-stream condition is encountered.

The ServletOutputStream Class

The **ServletOutputStream** class extends **OutputStream**.

It is implemented by the servlet container and provides an output stream that a servlet developer can use to write data to a client response.

A default constructor is defined. It also defines the **print( )** and **println( )** methods, which output data to the stream.

The ServletException Classes

javax.servlet defines two exceptions.

The first is **ServletException**, which indicates that a servlet problem has occurred.

The second is **UnavailableException**, which extends **ServletException**, which indicates that a servlet is unavailable.

Reading Servlet Parameters

The **ServletRequest** interface includes methods that allows to read the names and values of parameters that are included in a client request.

Ex: A web page is defined in PostParameters.htm.
A servlet is defined in PostParametersServlet.java.

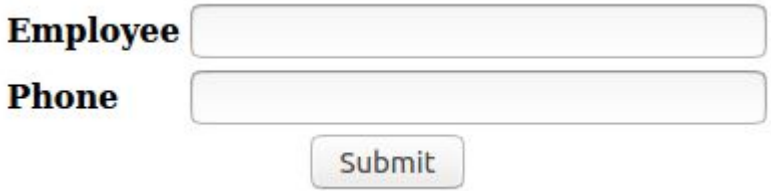
The HTML source code for PostParameters.htm defines a table that contains two labels and two text fields.

Reading Servlet Parameters

One of the labels is Employee and the other is Phone. There is also a submit button.

Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP POST request.

```
<html>
<body>
<center>
<form name="Form1"
    method="post"
    action="http://localhost:8080/servlets-examples/
    servlet/PostParametersServlet">
```



Reading Servlet Parameters

```
<table>
<tr>
    <td><B>Employee</td>
    <td><input type=textbox name="e" size="25" value=""></td>
</tr>
<tr>
    <td><B>Phone</td>
    <td><input type=textbox name="p" size="25" value=""></td>
</tr>
</table>
<input type=submit value="Submit">
</body>
</html>
```

Reading Servlet Parameters

The source code for **PostParametersServlet.java** is as follows.

The **service( )** method is overridden to process client requests.

The **getParameterNames( )** method returns an enumeration of the parameter names. These are processed in a loop.

The parameter name and value are output to the client. The parameter value is obtained via the **getParameter( )** method.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
public class PostParametersServlet extends GenericServlet {
public void service(ServletRequest request, ServletResponse response)
                                        throws ServletException, IOException {
```

The javax.servlet.http Package

The javax.servlet.http package contains a number of interfaces and classes that are commonly used by servlet developers.

This functionality makes it easy to build servlets that work with HTTP requests and responses.

The following table summarizes the core interfaces that are provided in this package:

Interface	Description
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session.

The javax.servlet.http Package

The following table summarizes the core classes that are provided in this package.

The most important of these is HttpServlet.

Servlet developers can extend this class in order to process HTTP requests.

Class	Description
Cookie	Allows state information to be stored on a client machine.
HttpServlet	Provides methods to handle HTTP requests and responses.
HttpSessionEvent	Encapsulates a session-changed event.
HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a session value, or that a session attribute changed.

15. List the methods that are available in HTTPServletRequest and HTTPServletResponse interface.

The HttpServletRequest Interface

The HttpServletRequest interface enables a servlet to obtain information about a client request. Several of its methods are shown in the table.

Method	Description
String getAuthType( )	Returns authentication scheme.
Cookie[ ] getCookies( )	Returns an array of the cookies in this request.
long getDateHeader(String *field*)	Returns the value of the date header field named *field*.
String getHeader(String *field*)	Returns the value of the header field named *field*.
Enumeration getHeaderNames( )	Returns an enumeration of the header names.
int getIntHeader(String *field*)	Returns the **int** equivalent of the header field named *field*.
String getMethod( )	Returns the HTTP method for this request.
String getPathInfo( )	Returns any path information that is located after the servlet path and before a query string of the URL.
String getPathTranslated( )	Returns any path information that is located after the servlet path and before a query string of the URL after translating it to a real path.
String getQueryString( )	Returns any query string in the URL.
String getRemoteUser( )	Returns the name of the user who issued this request.
String getRequestedSessionId( )	Returns the ID of the session.
String getRequestURI( )	Returns the URI.

The HttpServletRequest Interface

String getServletPath( )	Returns that part of the URL that identifies the servlet.
HttpSession getSession( )	Returns the session for this request. If a session does not exist, one is created and then returned.
HttpSession getSession(boolean *new*)	If *new* is **true** and no session exists, creates and returns a session for this request. Otherwise, returns the existing session for this request.
boolean isRequestedSessionIdFromCookie( )	Returns **true** if a cookie contains the session ID. Otherwise, returns **false**.
boolean isRequestedSessionIdFromURL( )	Returns **true** if the URL contains the session ID. Otherwise, returns **false**.
boolean isRequestedSessionIdValid( )	Returns **true** if the requested session ID is valid in the current session context.

TABLE 31-5 Various Methods Defined by **HttpServletRequest**

The HttpServletResponse Interface

The HttpServletResponse interface enables a servlet to formulate an HTTP response to a client.

Several constants are defined in this interface. These correspond to the different status codes that can be assigned to an HTTP response.

Ex: SC_OK indicates that the HTTP request succeeded, and
SC_NOT_FOUND indicates that the requested resource is not available.

Several methods of this interface are summarized in the table.

The HttpServletResponse Interface

Method	Description
void addCookie(Cookie *cookie*)	Adds *cookie* to the HTTP response.
boolean containsHeader(String *field*)	Returns **true** if the HTTP response header contains a field named *field*.
String encodeURL(String *url*)	Determines if the session ID must be encoded in the URL identified as *url*. If so, returns the modified version of *url*. Otherwise, returns *url*. All URLs generated by a servlet should be processed by this method.
String encodeRedirectURL(String *url*)	Determines if the session ID must be encoded in the URL identified as *url*. If so, returns the modified version of *url*. Otherwise, returns *url*. All URLs passed to **sendRedirect( )** should be processed by this method.

TABLE 31-6 Various Methods Defined by **HttpServletResponse**

The HttpServletResponse Interface

Method	Description
void sendError(int *c*) throws IOException	Sends the error code *c* to the client.
void sendError(int *c*, String *s*) throws IOException	Sends the error code *c* and message *s* to the client.
void sendRedirect(String *url*) throws IOException	Redirects the client to *url*.
void setDateHeader(String *field*, long *msec*)	Adds *field* to the header with date value equal to *msec* (milliseconds since midnight, January 1, 1970, GMT).
void setHeader(String *field*, String *value*)	Adds *field* to the header with value equal to *value*.
void setIntHeader(String *field*, int *value*)	Adds *field* to the header with value equal to *value*.
void setStatus(int *code*)	Sets the status code for this response to *code*.

TABLE 31-6 Various Methods Defined by **HttpServletResponse** *(continued)*

The HttpSession Interface

The HttpSession interface enables a servlet to read and write the state information that is associated with an HTTP session.

Several of its methods are summarized in the table. All of these methods throw an **IllegalStateException** if the session has already been invalidated.

In client-server protocols, like HTTP, sessions consist of three phases:

1. The client establishes a connection (Ex: TCP).
2. The client sends its request, and waits for the answer.
3. The server processes the request, sending back its answer, providing a status code and appropriate data.

As of HTTP/1.1, the connection is no longer closed after completing the third phase, and the client is now granted a further request: this means the second and third phases can now be performed any number of times.

The HttpSession Interface

Method	Description
Object getAttribute(String *attr*)	Returns the value associated with the name passed in *attr*. Returns **null** if *attr* is not found.
Enumeration getAttributeNames( )	Returns an enumeration of the attribute names associated with the session.
long getCreationTime( )	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when this session was created.
String getId( )	Returns the session ID.
long getLastAccessedTime( )	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the client last made a request for this session.
void invalidate( )	Invalidates this session and removes it from the context.
boolean isNew( )	Returns **true** if the server created the session and it has not yet been accessed by the client.
void removeAttribute(String *attr*)	Removes the attribute specified by *attr* from the session.
void setAttribute(String *attr*, Object *val*)	Associates the value passed in *val* with the attribute name passed in *attr*.

TABLE 31-7 The Methods Defined by **HttpSession**

The HttpSessionBindingListener Interface

The HttpSessionBindingListener interface is implemented by objects that need to be notified when they are bound to or unbound from an HTTP session.

The methods that are invoked when an object is bound or unbound are

void valueBound(HttpSessionBindingEvent e)

void valueUnbound(HttpSessionBindingEvent e)

Here, e is the event object that describes the binding.

The Cookie Class

The Cookie class encapsulates a cookie. A cookie is stored in a client machine and contains state information. Cookies are valuable for tracking user activities.

Ex: Assuming that a user visits an online store. A cookie can save the user's name, address, and other information. The user does not need to enter this data each time he or she visits the store.

The Cookie Class

A servlet can write a cookie to a client's machine via the **addCookie( )** method of the **HttpServletResponse** interface.

The data for that cookie is then included in the header of the HTTP response that is sent to the browser.

The names and values of cookies are stored on the client's machine. Some of the information that is saved for each cookie includes the following:

- The name of the cookie
- The value of the cookie
- The expiration date of the cookie
- The domain and path of the cookie

The Cookie Class

The **expiration date** determines when this cookie is deleted from the user's machine. If an expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends. Otherwise, the cookie is saved in a file on the user's machine.

The **domain** and **path** of the cookie determine when it is included in the header of an HTTP request.

If the user enters a URL whose domain and path match these values, the cookie is then supplied to the Web server. Otherwise, it is not.

There is one constructor for Cookie.

Cookie(String name, String value)

Here, the name and value of the cookie are supplied as arguments to the constructor. The methods of the Cookie class are summarized in the table.

The Cookie Class

Method	Description
Object clone( )	Returns a copy of this object.
String getComment( )	Returns the comment.
String getDomain( )	Returns the domain.
int getMaxAge( )	Returns the maximum age (in seconds).
String getName( )	Returns the name.
String getPath( )	Returns the path.
boolean getSecure( )	Returns **true** if the cookie is secure. Otherwise, returns **false**.
String getValue( )	Returns the value.
int getVersion( )	Returns the version.

The Cookie Class

void setComment(String *c*)	Sets the comment to *c*.
void setDomain(String *d*)	Sets the domain to *d*.
void setMaxAge(int *secs*)	Sets the maximum age of the cookie to *secs*. This is the number of seconds after which the cookie is deleted.
void setPath(String *p*)	Sets the path to *p*.
void setSecure(boolean *secure*)	Sets the security flag to *secure*.
void setValue(String *v*)	Sets the value to *v*.
void setVersion(int *v*)	Sets the version to *v*.

TABLE 31-8 The Methods Defined by **Cookie**

The HttpServlet Class

The **HttpServlet** class extends **GenericServlet**. It is commonly used when developing servlets that receive and process HTTP requests. The methods of the HttpServlet class are summarized in the table.

Method	Description
void doDelete(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP DELETE request.
void doGet(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP GET request.
void doHead(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP HEAD request.
void doOptions(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP OPTIONS request.
void doPost(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP POST request.

The HttpSessionEvent Class

HttpSessionEvent encapsulates session events. It extends **EventObject** and is generated when a change occurs to the session.

It defines this constructor:

HttpSessionEvent(HttpSession session)

Here, session is the source of the event.

HttpSessionEvent defines one method, getSession( ):

HttpSession getSession( )

It returns the session in which the event occurred.

The HttpSessionBindingEvent Class

The HttpSessionBindingEvent class extends HttpSessionEvent. It is generated when a listener is bound to or unbound from a value in an HttpSession object.

It is also generated when an attribute is bound or unbound. Here are its constructors:

HttpSessionBindingEvent(HttpSession session, String name)

HttpSessionBindingEvent(HttpSession session, String name, Object val)

Here, session is the source of the event, and name is the name associated with the object that is being bound or unbound. If an attribute is being bound or unbound, its value is passed in val.

The getName( ) method obtains the name that is being bound or unbound.

String getName( )

The getSession( ) method, obtains the session to which the listener is being bound or unbound:

HttpSession getSession( )

The HttpSessionBindingEvent Class

The getValue( ) method obtains the value of the attribute that is being bound or unbound.

Object getValue( )

Handling HTTP Requests and Responses

The HttpServlet class provides specialized methods that handle the various types of HTTP requests. A servlet developer overrides one of these methods.

These methods are **doDelete( )**, **doGet( )**, **doHead( )**, **doOptions( )**, **doPost( )**, **doPut( )**, and **doTrace( )**.

The **GET** and **POST** requests are commonly used when handling form input.

Handling HTTP GET Requests

A servlet will be developed which handles an HTTP GET request.

The servlet is invoked when a form on a web page is submitted.

The example contains two files.

A web page is defined in **ColorGet.htm**, and a servlet is defined in **ColorGetServlet.java**.

The HTML source code for **ColorGet.htm** is shown in the following listing.

Handling HTTP GET Requests

```
<html>
<body>
<center>
<form name="Form1"
                action="http://localhost:8080/servlets-examples/servlet/ColorGetServlet">
<B>Color:</B>
        <select name="color" size="1">
                <option value="Red">Red</option>
                <option value="Green">Green</option>
                <option value="Blue">Blue</option>
        </select>
<br><br>
        <input type=submit value="Submit">
</form>
</body>
</html>
```

Handling HTTP GET Requests

It defines a form that contains a **select element** and a **submit button**. The action parameter of the form tag specifies a URL. The URL identifies a servlet to process the HTTP GET request.

The source code for **ColorGetServlet.java** is shown in the following listing. The **doGet( )** method is overridden to process any HTTP GET requests that are sent to this servlet.

It uses the **getParameter( )** method of **HttpServletRequest** to obtain the selection that was made by the user. A response is then formulated.

16. Define the following: i) Get method ii) Post method

Handling HTTP GET Requests

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorGetServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
        String color = request.getParameter("color");
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<B>The selected color is: ");
        pw.println(color);
        pw.close();
    }
}
```

Handling HTTP GET Requests

Compile the servlet. Next, copy it to the appropriate directory, and update the web.xml. Then, perform these steps to test this example:

1. Start Tomcat, if it is not already running.
2. Display the web page in a browser.
3. Select a color.
4. Submit the web page.

After completing these steps, the browser will display the response that is dynamically generated by the servlet.

One other point: Parameters for an HTTP GET request are included as part of the URL that is sent to the web server. Assume that the user selects the red option and submits the form. The URL sent from the browser to the server is

http://localhost:8080/servlets-examples/servlet/ColorGetServlet?color=Red

The **characters to the right of the question mark are known as the query string.**

Handling HTTP POST Requests

A servlet that handles an HTTP POST request is developed here. The servlet is invoked when a form on a web page is submitted.

The example contains two files. A web page is defined in **ColorPost.htm**, and a servlet is defined in **ColorPostServlet.java**.

The HTML source code for ColorPost.htm is shown in the following listing.

```
<html>
<body>
<center>
<form name="Form1" method="post"
    action="http://localhost:8080/servlets-examples/servlet/ColorPostServlet">
    <B>Color:</B>
        <select name="color" size="1">
            <option value="Red">Red</option>
            <option value="Green">Green</option>
```

Handling HTTP POST Requests

```
        <option value="Blue">Blue</option>
        </select>
<br><br>
        <input type=submit value="Submit">
</form>
</body>
</html>
```

It is identical to ColorGet.htm except that the method parameter for the form tag explicitly specifies that the POST method should be used, and the action parameter for the form tag specifies a different servlet.

The source code for **ColorPostServlet.java** is shown in the following listing. The **doPost( )** method is overridden to process any HTTP POST requests that are sent to this servlet. It uses the **getParameter( )** method of HttpServletRequest to obtain the selection that was made by the user. A response is then formulated.

Handling HTTP POST Requests

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorPostServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException
    {
        String color = request.getParameter("color");
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<B>The selected color is: ");
        pw.println(color);
        pw.close();
    }   }
```

Handling HTTP POST Requests

NOTE: Parameters for an HTTP POST request are not included as part of the URL that is sent to the web server. In this example, the URL sent from the browser to the server is http://localhost:8080/servlets-examples/servlet/ColorPostServlet.
The parameter names and values are sent in the body of the HTTP request.

Using Cookies

A servlet will be developed to illustrate the use of cookies.

The servlet is invoked when a form on a web page is submitted.
The example contains three files as summarized here:

File	Description
AddCookie.htm	Allows a user to specify a value for the cookie named **MyCookie**.
AddCookieServlet.java	Processes the submission of **AddCookie.htm**.
GetCookiesServlet.java	Displays cookie values.

Using Cookies

17. Define the following: i) Cookie usage ii)Session objects

The HTML source code for **AddCookie.htm** is shown in the following listing. This page contains a text field in which a value can be entered.

There is also a submit button on the page. When this button is pressed, the value in the text field is sent to **AddCookieServlet** via an HTTP POST request.

```
<html>
  <body>
    <center>
      <form name="Form1"     method="post"
        action="http://localhost:8080/servlets-examples/servlet/AddCookieServlet">
         <B>Enter a value for MyCookie:</B>
        <input type=textbox  name="data" size=25 value="">
       <input type=submit   value="Submit">
    </form>
  </body>
</html>
```

Using Cookies

The source code for **AddCookieServlet.java** is shown in the following listing.

It gets the value of the parameter named "data".
It then creates a **Cookie** object that has the name "MyCookie" which contains value of the "data" parameter.

The cookie is then added to the header of HTTP response via **addCookie( )** method. A feedback message is then written to the browser.

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

public class AddCookieServlet extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException
   {
```

Using Cookies

```
        // Get parameter from HTTP request.
         String data = request.getParameter("data");

        // Create cookie.
         Cookie cookie = new Cookie("MyCookie", data);

        // Add cookie to HTTP response.
         response.addCookie(cookie);

        // Write output to browser.
         response.setContentType("text/html");
         PrintWriter pw = response.getWriter();
         pw.println("<B>MyCookie has been set to");
         pw.println(data);

        pw.close();
}       }
```

Using Cookies

```
<html>
  <body>
    <center>
      <form name="Form1"  method="get"
          action="http://localhost:8080/cookies/GetCookiesServlet">
      <input type=submit value="Submit">
      </form>
  </body>
</html>
```

Using Cookies

The source code for **GetCookiesServlet.java** is shown in the following listing.

It invokes the **getCookies( )** method to read any cookies that are included in the HTTP GET request.

The names and values of these cookies are then written to the HTTP response.

getName( ) and **getValue( )** methods are called to obtain this information.

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

public class GetCookiesServlet extends HttpServlet {

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

Using Cookies

```
        // Get cookies from header of HTTP request.
        Cookie[] cookies = request.getCookies();

       // Display these cookies.
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<B>");
        for(int i = 0; i < cookies.length; i++)
        {
            String name = cookies[i].getName();
            String value = cookies[i].getValue();
            pw.println("name = " + name + "; value = " + value);
        }
       pw.close();
  }
}
```

Session Tracking

HTTP is a stateless protocol. Each request is independent of the previous one.

In some applications it may be necessary to save state information so that information can be collected from several interactions between a browser and a server. Sessions provide such a mechanism.

A session can be created via the **getSession( )** method of **HttpServletRequest**.
An **HttpSession** object is returned by this method.
This object can store a set of bindings that associate names with objects.

The **setAttribute( )**, **getAttribute( )**, **getAttributeNames( )**, and **removeAttribute( )** methods of HttpSession manage these bindings.

Session Tracking

It is important to note that session state is shared among all the servlets that are associated with a particular client.

The following servlet illustrates the usage of session state.

The **getSession( )** method gets the current session. A new session is created if one does not already exist.

The **getAttribute( )** method is called to obtain the object that is bound to the name “date”.

Date object encapsulates the date and time when the current page was last accessed.

A Date object encapsulating the current date and time is then created.

The **setAttribute( )** method is called to bind the name “date” to this object.

Session Tracking

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

public class DateServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,  HttpServletResponse response)
                        throws ServletException, IOException   {
        // Get the HttpSession object.
        HttpSession hs = request.getSession(true);
        // Get writer.
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.print("<B>");
```

```
        // Display date/time of last access.
        Date date = (Date)hs.getAttribute("date");
        if(date != null)
          pw.print("Last access: " + date + "<br>");
        // Display current date/time.
        date = new Date();
        hs.setAttribute("date", date);
        pw.println("Current date: " + date);
    }
}
```

Java Servlet Pages (JSP)

JSP is a server-side program that is similar in design and functionality to a java servlet.

JSP will be called by a client to provide a web service, the nature of which depends on the J2EE (Java 2 Enterprise Edition) application.

JSP processes the request (by using logic built into the JSP or by calling other web components built using Java servlet technology or created using other technologies), and responds by sending the results to the client.

JSP differs from a Java servlet in the way in which the JSP is written.

Java servlet is coded using the Java programming language and responses are encoded as an output String object that is passed to the println( ) method. The output String object is formatted in HTML, XML or the format required by the client.

Java Servlet Pages (JSP)

JSP is coded in HTML, XML or in the client's format that is interlaced with scripting elements, directives and actions composed of Java programming language and JSP syntax.

JSP can be used as a middle-level program between clients and web services.

JSP

JSP is simpler to create than a Java servlet, because JSP is coded in HTML rather than with the Java programming language.

JSP provides the same features found in a Java servlet because a JSP is converted to a Java servlet the first time a client requests the JSP.

There are 3 methods that are automatically called when a JSP is requested and when JSP terminates normally. These are **jspInt( )** method, the **jspDestroy( )** method and the **service( )** method. These methods can be overridden.

jspInt( ) method and **jspDestroy( )** methods are commonly overridden in a JSP to provide customized functionality, when JSP is called and terminates.

jspInt( ) method is identical to the **init( )** method in a Java servlet and in an applet.
jspInt( ) method is called when the JSP is requested and is used to initialize objects and variables that are used throughout the life of JSP.

jspDestroy( ) method is identical to the **destroy( )** method in a Java servlet.
destroy( ) is automatically called when the JSP terminates normally. This method will not be called if the JSP abruptly terminates (when server crashes).
destroy( ) method is used for cleanup where resources used during the execution of the JSP are released, such as disconnecting from a database.

service( ) method is automatically called and retrieves a connection to HTTP.

JSP Tags

A JSP program consists of a combination of HTML tags and JSP tags.

JSP tag defines Java code that is to be executed before the output of the JSP program is sent to the browser.

A JSP tag begins with **<%**, which is followed by Java code and ends with **%>**.

JSP tags are embedded into the HTML component of a JSP program and are processed by a JSP virtual engine such as Tomcat. Tomcat reads the JSP program, whenever the program is called by a browser and resolves JSP tags, then sends the HTML tags and related information to the browser.

Java code associated with JSP tags in the JSP program is executed when encountered by Tomcat, and the result of that process is sent to the browser. Browser is aware of displaying the result because the JSP tag is enclosed within an open and closed HTML tag.

JSP Tags

There are 5 types of JSP tags in a JSP program, which are as follows.

Comment tag

A comment tag opens with **<%--** and closes with **--%>** and is followed by a comment that usually describes the functionality of statements that follow the comment tag.

Declaration statement tag

This tag opens with **<%!** and is followed by a Java declaration statement that defines variables, objects and methods that are available to other components of the JSP program.

Directive tags

This tag opens with **<%@** and commands the JSP virtual engine to perform a specific task, such as importing a Java package required by objects and methods used in a declaration statement. Directive tag closes with **%>**.

JSP Tags

There are 3 commonly used directives, which are **import**, **include** and **taglib**.

import tag is used to import java packages into the JSP program.

include tag inserts a specified file into the JSP program replacing the include tag.

taglib tag specifies a file that contains a tag library.

Ex:

```
<%@ page import=" import java.sql.*"; %>
<%@ include file="keogh\books.html" %>
<%@ taglib url="myTags.tld" %>
```

First tag imports java.sql package.

Second tag includes the books.html located in the keogh directory.

Last tag loads the myTags.tld library.

JSP Tags

Expression tags

This tag opens with **<%=** and is used for an expression statement whose result replaces the expression tag when the JSP virtual engine resolves JSP pages. This tag closes with **%>**.

Scriptlet tags

This tag opens with **<%** and contains commonly used Java control statements and loops. A scriplet tag closes with **%>**.

Variables and Objects

Declaration of variables and objects are similar to Java programs, except that they must appear as a JSP tag within the JSP program before the variable or object is used in the program.

JSP Tags

A simple JSP program is shown below.

```
<html>
   <head>
      <title>Printing value of a variable</title>
    </head>
      <body>
            <%! int age=29; %>
            <P> Your age is: <%=age%> </P>
      </body>
</html>
```

Program declares an int called age and initializes the variable with value 29, and the declaration statement is placed within the JSP tag (which begins with <%!. **Pg 30**