

# Exploring the Java Servlet Technology

With the increase in the usage of the Internet as the information medium, organizations are creating dynamic Web applications. A dynamic Web application consists of dynamic web pages that enable a user to retrieve data from a database, submit data, or fetch some information based on users input. These features can be incorporated by implementing server-side programs. The server-side programs can be written using various technologies, such as Servlet. Servlet is a Java technology that extends all the features of Java. This enables you to develop powerful, dynamic Web applications.

This chapter focuses on the Servlet APIs, lifecycle of a servlet, and servlet request processing mechanism. In addition, it explains creation and deployment of a servlet.

## Objectives

In this chapter, you will learn to:

- ☞ Introduce servlets
- ☞ Implement servlets





# Introduction to Servlets

With the increase in number of users accessing the Internet, the Web has evolved greatly. In addition to being a repository of information and a mode of communication, it is now increasingly used as a medium of entertainment, business, and socializing. This has enforced the need of dynamic Web applications. A dynamic Web application responds to the actions performed by a user dynamically.

Consider the Google website that enables you to search documents, links, images, and videos, on any topic. If you want to search only images, you can select the **Images** link on the main page of the website, which redirects you to the page that allows you to search and display only images.

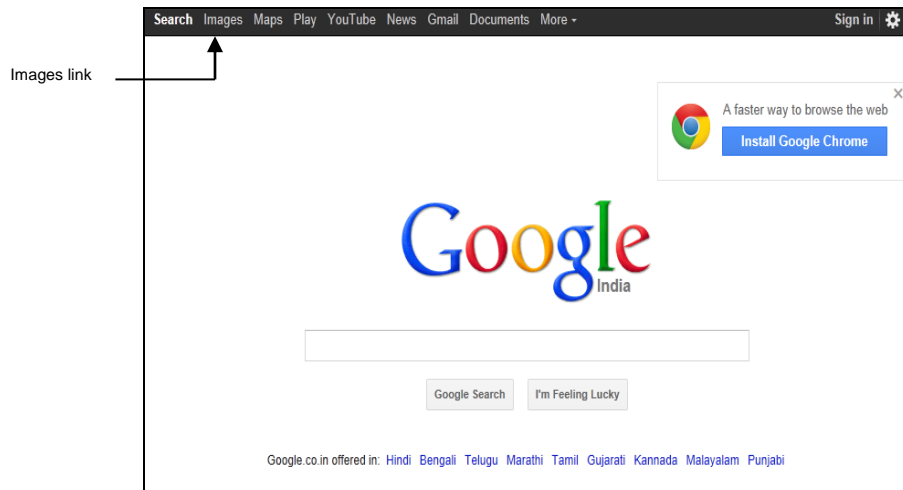
The following steps display graphical depiction of the preceding example:

1. Type **http://www.google.co.in/** in the address bar of Internet Explorer. The following page is displayed.



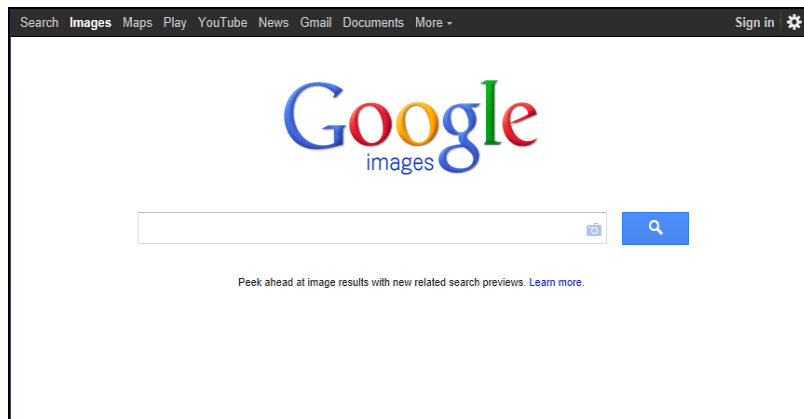
*The Main Page of the Google Website*

2. Click the **Images** link on the black bar of the page, as shown in the following figure.




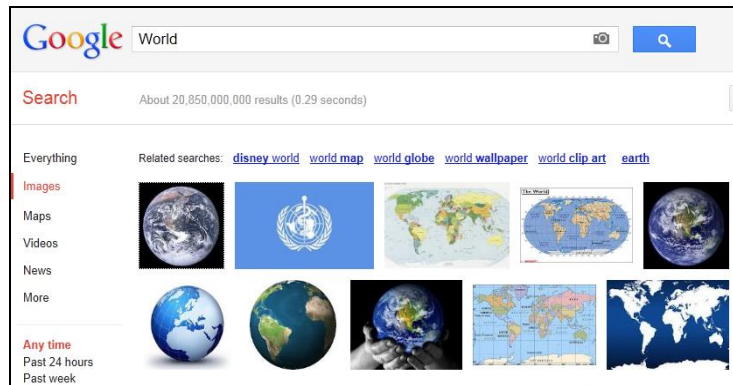
*The Home Page of the Google Website*

The user is redirected to the search images page, as shown in the following figure.



*The Search Images Page*

3. Type the word, World, in the search textbox and click the  button. It displays all the images related to the word **World**, as shown in the following figure.



*The Sample Result*

The preceding example displays the dynamic functionality of the Google website, where the search results are displayed based on the search type and the search text entered by the user. To develop such dynamic Web applications in Java, Servlet technology can be used.

*Servlet* is a Java program that runs on the server side. It inherits all the features of the Java programming language. In addition, it consists of the following features:

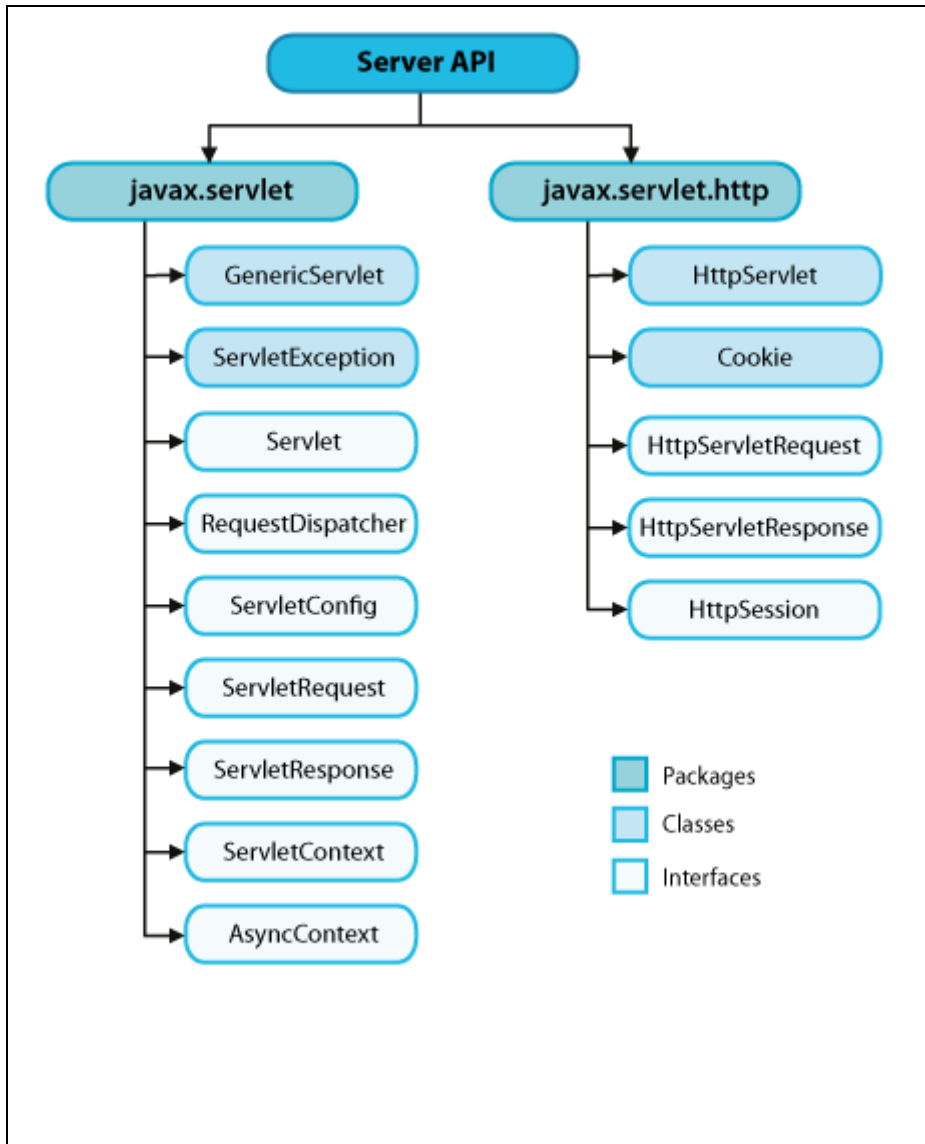
- **Efficient:** Servlets provide faster response to the client requests as it is loaded only once in the Web server memory.
- **Asynchronous support:** Servlet supports asynchronous programming that enables it to serve multiple client requests and generate responses simultaneously. This helps the user to work continuously on the application without being affected by the responses received from the server. In addition, it supports AJAX to implement asynchronous programming that allows a user to interact with the rest of the Web page while the application is processing the changed or updated parts of the Web page.

## The Servlet API

To create and manage servlets, various interfaces, classes, and methods are defined in the Servlet API. These classes and interfaces of the Servlet API are available in the following packages:

- `javax.servlet`
- `javax.servlet.http`

The following figure displays the class hierarchy of the Servlet API.



*The Servlet API Hierarchy*

## The javax.servlet Package

The `javax.servlet` package is the integral package of the Servlet API. This package consists of a `Servlet` interface that must be implemented by all servlets.

The following table lists some of the interfaces included in the `javax.servlet` package.

<i><b>Interfaces</b></i>	<i><b>Description</b></i>
<i>Servlet</i>	<i>It provides the methods that all servlets must implement.</i>
<i>ServletConfig</i>	<i>It provides information to the servlets during the servlet initialization.</i>
<i>ServletContext</i>	<i>It provides methods, which is used by the servlets to communicate with the Web container.</i>
<i>ServletRequest</i>	<i>It provides the client request information to the servlet.</i>
<i>ServletResponse</i>	<i>It helps the servlet in sending the response to the client.</i>
<i>RequestDispatcher</i>	<i>It receives a request from a client and sends it to any other resource, such as a servlet, an HTML page, or a JSP page.</i>
<i>AsyncContext</i>	<i>It provides the methods to handle multiple client requests asynchronously.</i>

*The Interfaces of the javax.servlet Package*



**Note**

*Servlets run inside a component container in the Web server. This component container is called the Web container, which provides the run-time environment for the execution of servlets.*

## The `javax.servlet.Servlet` Interface

The `Servlet` interface provides the methods that must be implemented by all the servlets. To implement these methods, you must create a servlet class that either extends a `GenericServlet` or `HttpServlet` class.

The following table describes the various methods of the `Servlet` interface.

<b>Methods</b>	<b>Description</b>
<code>void init(ServletConfig config)</code> <code>throws ServletException</code>	<i>This method is called by the Web container after creating a servlet instance. This indicates that the servlet is now ready to service the client requests.</i>
<code>ServletConfig</code> <code>getServletConfig()</code>	<i>This method returns a <code>ServletConfig</code> object that contains the configuration information, such as initialization parameters that are passed to a servlet during its initialization.</i>
<code>String getServletInfo()</code>	<i>This method returns a string that contains the information about the servlet, such as author, version, and copyright.</i>
<code>void service(ServletRequest request, ServletResponse response)</code> <code>throws ServletException, IOException</code>	<i>This method is called by the Web container to enable a servlet to generate response for a client request.</i>
<code>void destroy()</code>	<i>This method is called by the Web container just before the servlet instance is taken out of service.</i>

*The Methods of the Servlet Interface*

### The `javax.servlet.ServletConfig` Interface

The `javax.servlet.ServletConfig` interface is implemented by a Web container during the initialization of a servlet to pass the configuration information to a servlet. A servlet is initialized by passing an object of the `ServletConfig` interface to its `init()` method. It is similar to a constructor in a Java class. Therefore, it can be used to pass the initialization parameters to the servlets. The initialization parameters are passed as name-value pairs.

For example, you want that when a Web application is initialized, it should establish a connection with an SQL database named, `BusinessDetails`. In addition, this database is used by most of the pages within the Web application. Therefore, you can specify the connection URL as an initialization parameter of the servlet. When the servlet is initialized, it can use the URL value to obtain a database connection and the same can be shared by the Web pages within the Web application.



The following table lists some of the methods of the `ServletConfig` interface.

<b>Methods</b>	<b>Description</b>
<code>String getInitParameter(String param)</code>	<i>It returns a <code>String</code> object containing the value of the initialization parameters. If the parameter does not exist, it returns <code>null</code>.</i>
<code>Enumeration&lt;String&gt; getInitParameterNames()</code>	<i>It returns the names of all the initialization parameters as an enumeration of <code>String</code> objects. If no initialization parameters have been defined, an empty enumeration is returned.</i>
<code>ServletContext getServletContext()</code>	<i>It returns the <code>ServletContext</code> object for the servlet in which the caller is executing.</i>
<code>String getServletName()</code>	<i>It returns a <code>String</code> object containing the name of the servlet instance.</i>

*The Methods of the ServletConfig Interface*

## The `javax.servlet.ServletContext` Interface

The `ServletContext` interface provides information to all the servlets of a Web application about the environment in which they are running, such as the initialization parameters of the application, the path information of all the files stored in the Web application, and attributes or data that is common to all the servlets.

Each Web application consists of only one `ServletContext` object, also known as a servlet context or Web context. In addition to the information, the `ServletContext` object provides methods that can be used to communicate with the Web container. These methods provide services such as, logging messages to the application server log file or determining the MIME type of a file. To provide the context information, the Web container creates an object of the `ServletContext` interface. This object represents a context within which a servlet executes.

The following table describes the various methods of the `ServletContext` interface.

<b>Methods</b>	<b>Description</b>
<code>public void setAttribute(String, Object)</code>	<i>Binds the object with a name and stores the name-value pair as an attribute of the <code>ServletContext</code> object. If an attribute already exists, this method replaces the existing attribute.</i>
<code>public Object getAttribute(String attrname)</code>	<i>Returns the object stored in the <code>ServletContext</code> object with the name passed as a parameter.</i>
<code>public Enumeration getAttributeNames()</code>	<i>Returns an enumeration of the <code>String</code> object that contains names of all the context attributes.</i>
<code>public String getInitParameter(String pname)</code>	<i>Returns the value of the initialization parameter with the name passed as a parameter.</i>
<code>public Enumeration getInitParameterNames()</code>	<i>Returns an enumeration of the <code>String</code> object that contains names of all the initialization parameters.</i>

*The Methods of the ServletContext Interface*

## The ServletRequest Interface

The `ServletRequest` interface enables a servlet to handle the client requests by providing the client information to the servlet. For example, a servlet may need to validate the user Id and password entered by a user in the login page of a website. For this purpose, the servlet uses an object of the `ServletRequest` interface to retrieve the user Id and password entered by the user.

The following table describes the various methods of the `ServletRequest` interface.

<b>Methods</b>	<b>Description</b>
<code>public String getParameter(String paramName)</code>	Returns a <code>String</code> object that specifies the value of a particular request parameter.
<code>public String[] getParameterValues(String paramName)</code>	Returns an array of <code>String</code> objects that contains all the values of the request parameter.
<code>public Enumeration getParameterNames()</code>	Returns an <code>Enumeration</code> object containing all the parameter names as <code>String</code> objects that a servlet request contains.
<code>public String getRemoteHost()</code>	Returns a <code>String</code> object that specifies the full-qualified name of the computer from which the request is sent.
<code>public String getRemoteAddr()</code>	Returns a <code>String</code> object that specifies the IP address of the computer from which the request is sent.

*The Methods of the ServletRequest Interface*

## The ServletResponse Interface

The `ServletResponse` interface contains various methods that enable a servlet to respond to the client requests. A servlet can send the response as either character or binary data. The `PrintWriter` stream can be used to send the character data as a servlet response and the `ServletOutputStream` stream can be used to send the binary data as a servlet response.

The following table describes the various methods of the `ServletResponse` interface.

<i>Methods</i>	<i>Description</i>
<code>public ServletOutputStream getOutputStream() throws IOException</code>	Returns an object of the <code>ServletOutputStream</code> class that represents an output stream to send the binary data as response.
<code>public PrintWriter getWriter() throws IOException</code>	Returns an object of the <code>PrintWriter</code> class that the servlet uses to send character data as response.
<code>public void setContentType(String type)</code>	Sets the MIME type for a servlet response. Some of the MIME types are <code>text/plain</code> , <code>image/jpeg</code> , and <code>text/html</code> .

*The Methods of the ServletResponse Interface*

## The javax.servlet.http Package

The `javax.servlet.http` package provides classes and interfaces that enable you to create a servlet that communicates using the HTTP protocol. The following table lists some of the interfaces that belong to the `javax.servlet.http` package.

<i>Interfaces</i>	<i>Description</i>
<code>HttpServletRequest</code>	It extends the <code>ServletRequest</code> interface to represent the request information being sent to the HTTP servlets by an HTTP client.
<code>HttpServletResponse</code>	It extends the <code>ServletResponse</code> interface and provides methods to handle response, status codes, and response headers of the servlets that communicate using HTTP.
<code>HttpSession</code>	It provides a mechanism that helps in identifying a client across multiple requests.

*The Interfaces of the javax.servlet.http Package*

The `javax.servlet.http` package provides the `HttpServlet` class to create servlets, which communicate using the HTTP protocol. `HttpServlet` is an abstract class that extends from the `GenericServlet` class and provides built-in functionality for the HTTP protocol.

For example, the `HttpServlet` class provides methods, such as `doGet()`, `doPost()`, and `doHead()` that allow a servlet to process a client request coming through a specific HTTP method.

Consider the following code snippet that creates an HTTP servlet named `MyServlet`:

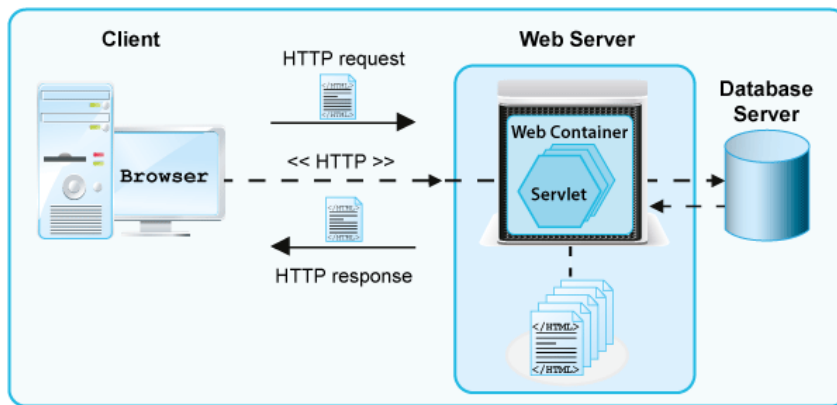
```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    {
        ...
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    {
        ...
    }
}
```

## Understanding the Web Container

The *Web container* is a component of the Web server which provides the run-time environment for executing the servlets. Whenever a client sends a request to the Web server for a particular servlet, the Web server passes the request to the Web container. The Web container checks whether an instance of the requested servlet exists. If the servlet instance exists, the Web container delegates the request to the servlet, which processes the client request and sends back the response. In case the servlet instance does not exist, the Web container locates and loads the servlet class. The Web container then creates an instance of the servlet and initializes it. The servlet instance, after initialization, starts processing the request. The Web container passes the response generated by the servlet to the client.

The following figure depicts the Web container architecture.

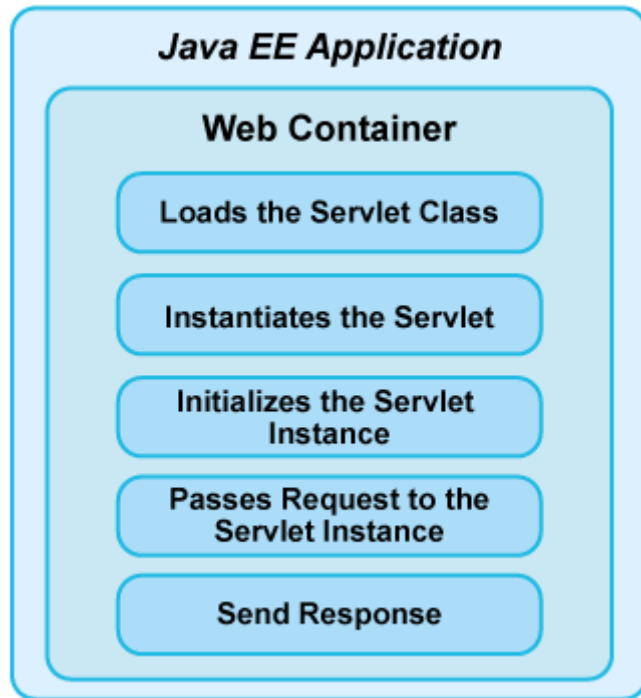


*The Web Container Architecture*

The Web container implements the Servlet APIs. It provides the following services to the Web applications:

- **Communication Support:** The Web container provides communication support so that a servlet can communicate with the Web server. Therefore, a Web developer only needs to develop the business logic of the servlets.
- **Lifecycle Management:** The Web container manages the lifecycle of the servlets. It manages the loading and instantiation of the servlets. The Web container is also responsible for invoking the servlet methods and making the instance of the servlet eligible for garbage collection.
- **Multithreading Support:** The Web container provides multithreading support for every servlet. Whenever a client requests for a servlet, the container automatically creates a thread and associates the client request with it. When the client request is processed completely, the thread is destroyed by the container.
- **Declarative Security:** The Web container defines the security constraints that specify that only the authorized users can access the deployed servlets. You need to write the complete code in your servlet to make your Web applications secure. Therefore, you can manage and change your security settings without recompiling the Java source files.
- **JSP Support:** The Web container compiles and translates the JSP pages into the servlet class files, which are then used to process the requests of a client.

The following figure shows the tasks performed by the Web container when the client request is received for the first time.



*The Tasks Performed by the Web Container*

## Life Cycle of a Servlet

The lifecycle of a servlet is managed by the Web container. To manage the servlet lifecycle, it uses the methods of the `javax.servlet.Servlet` interface. They are:

- `init()`
- `service()`
- `destroy()`

### The `init()` Method

The `init()` method is called during the initialization phase of the servlet life cycle. The Web container first maps the requested URL to the corresponding servlet available in the Web container and then instantiates the servlet. The Web container then creates an object of the `ServletConfig` interface, which contains the startup configuration information, such as initialization parameters of the servlet. The Web container then calls the `init()` method of the servlet and passes the `ServletConfig` object to it.

The `init()` method throws the `ServletException` exception, if the Web container cannot initialize the servlet resources.

The following code snippet shows the `init()` method:

```
public void init (ServletConfig config) throws ServletException
{
    ...
}
```

Consider a scenario where you want to track the number of users accessing a Web application. To implement this, you need a variable, which will track the number of users requesting the Web application. However, this variable should not be initialized for the subsequent requests. This can be implemented using the following code snippet:

```
public class ServletLifeCycle extends HttpServlet
{
    static int count;

    public void init (ServletConfig config) throws ServletException
    {
        count=0;
    }
}
```

In the preceding code snippet, a variable named `count` is declared inside the `ServletLifeCycle` class. It is initialized with the value 0 in the `init()` method of the servlet class. As the `init()` method is called only once during the lifecycle of a servlet, hence, the `count` variable will be initialized only once.

## The `service()` Method

Once the servlet instance is initialized, it is ready to service the client requests. When the client requests for the servlet, the Web container invokes the `service()` method of the servlet to process the client requests. The Web container passes an object of the `HttpServletRequest` interface and an object of the `HttpServletResponse` interface to the `service()` method. The `HttpServletRequest` object contains information about the request made by the client. The `HttpServletResponse` object contains the information returned by the servlet to the client.

The following code snippet shows the `service()` method:

```
public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException
{
    ...
}
```



The `service()` method throws the `ServletException` exception when the processing of the client request by the servlet fails. For example, the `service()` method identifies the type of client requests, and dispatches them to the `doGet()` and `doPost()` methods for processing. For example, if the user has clicked on a link, then the `service()` method will call the `doGet()` method.

Therefore, when a client requests for a resource using the HTTP GET method, then the `doGet()` method can be overridden in the servlet class to handle the client request. Similarly, you can override the `doPost()` method in the servlet class to handle the client requests, which are sent by the client using the HTTP POST method.

## The `destroy()` Method

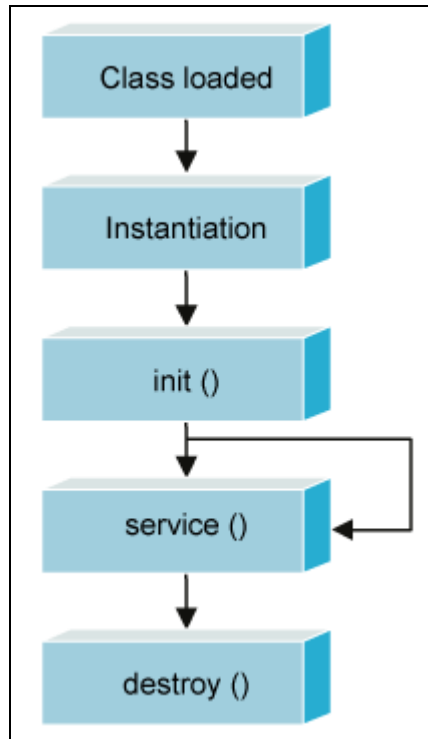
The `destroy()` method marks the end of the life cycle of a servlet. The Web container calls the `destroy()` method when the servlet instance is not required to service any subsequent client request. The Web container calls the `destroy()` method in the following situations:

- The time period specified for the servlet has elapsed. The time period of a servlet is the period for which the servlet is kept in the active state by the Web container to service the client request.
- The Web container needs to release servlet instances to reclaim memory.
- The Web container is about to shut down.
- There are no subsequent requests for a particular servlet instance.

In the `destroy()` method, you can write the code to release the resources occupied by a servlet. In addition, it is used to save any persistent information before the servlet instance is removed from the service. The following code snippet shows the `destroy()` method:

```
public void destroy()
{
}
```

These lifecycle methods are called in a specific sequence by the Web container, as shown in the following figure.



*The Servlet Lifecycle Methods Execution Sequence*

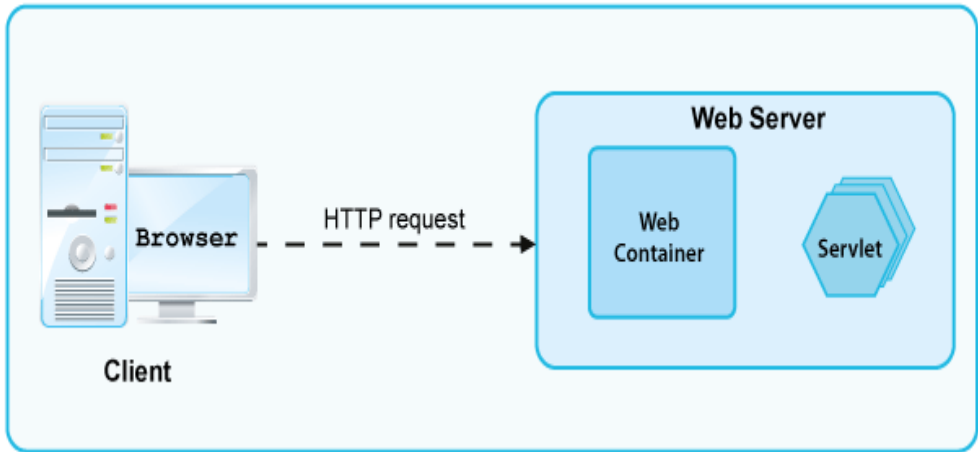
The following is the list of the sequence in which the Web container calls the life cycle methods of a servlet:

1. The Web container loads the servlet class and creates one or more instances of the servlet class.
2. The Web container invokes the `init ()` method of the servlet instance during initialization of the servlet. The `init ()` method is invoked only once in the servlet life cycle.
3. The Web container invokes the `service ()` method to allow a servlet to process a client request.
4. The `service ()` method processes the request and returns the response to the Web container.
5. The servlet then waits to receive and process subsequent requests as explained in the steps 3 and 4.
6. The Web container calls the `destroy ()` method before removing the servlet instance from the service. The `destroy ()` method is also invoked only once in a servlet life cycle.

## Processing of a Servlet Request

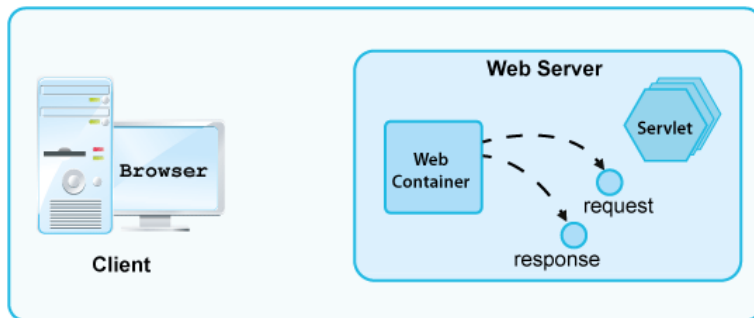
The following steps describe processing of a client request by a servlet:

1. A user sends a request to the server by using the Web browser. The server then forwards the request to the Web container, as shown in the following figure.



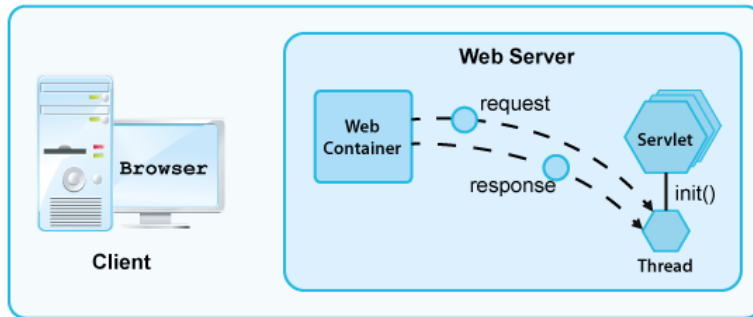
*The HTTP Request Being Sent*

2. The Web container in turn creates the `HttpServletRequest` and `HttpServletResponse` objects for request handling and response generation, as shown in the following figure.



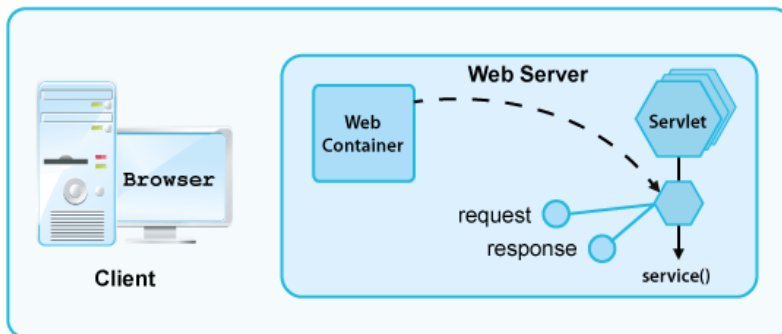
*The Creation of Request and Response Objects*

3. The Web container identifies the requested servlet, and then initializes the identified servlet thread by calling the `init()` method. In addition, it passes the `HttpServletRequest` and `HttpServletResponse` objects to the initialized servlet thread, as shown in the following figure.



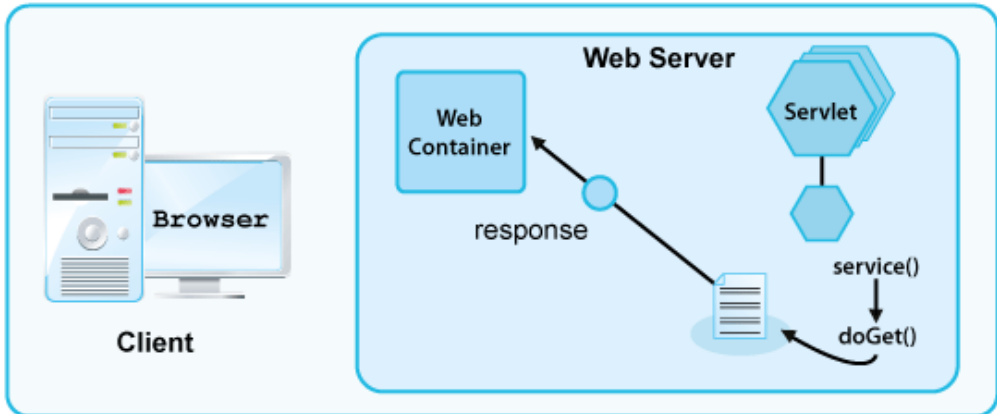
*The Creation of a Servlet Thread*

4. Now, the Web container calls the `service()` method of the servlet to enable the servlet to process the client requests, as shown in the following figure.



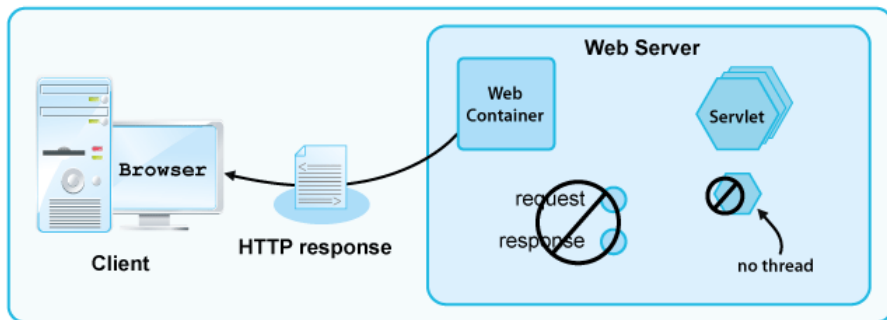
*The Invocation of the service() Method*

5. Depending upon the type of request, the `service()` method invokes the `doGet()` or `doPost()` method. For example, in the given example if the request is sent to the server by using the GET method, then the `doGet()` method is invoked to service the client, as shown in the following figure.



*The Invocation of the doGet() Method*

6. The `doGet()` method generates a response for the client and attaches it with the response object. The Web container sends the generated response to the client. Finally, the Web container deletes the servlet instance along with the request and response objects, as shown in the following figure.



*The Deletion of Request and Response Objects*



***Just a minute:***

*The \_\_\_\_\_ package provides the classes and interfaces to create servlets that support HTTP protocol.*

***Answer:***

`javax.servlet.http`

# Implementing Servlets

To create a servlet, you need to perform the following tasks:

1. Create a servlet that either extends the `HttpServlet` or `GenericServlet` class.
2. Configure the servlet.
2. Compile and package the servlet.
3. Deploy the servlet on the application server.
4. Access the servlet using a browser application.

## Creating a Servlet

Consider an example, where you have been asked to create a servlet, which displays the current date to the user, each time the servlet is accessed. To implement this, you have to create a servlet that extends from the `HttpServlet` class, as shown in the following code snippet:

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;

public class CurrentDate extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<h1>" + new java.util.Date() + "</h1>");
        out.println("</BODY></HTML>");
    }
}
```

In the preceding code, the `CurrentDate` class extends the `HttpServlet` class and overrides the `doGet()` method. The `setContentType()` method associated with the `HttpServletResponse` object is used to set the content type of the response to `text/html`. The `getWriter()` method of the `HttpServletResponse` object is used to obtain an instance of the `PrintWriter` class. Finally, the `println()` method of the object of the `PrintWriter` class is used to add the HTML markup to the servlet response. After creating the servlet class, it is then configured to be accessed by the Web container.

## Configuring a Servlet

After creating the servlet, it must be associated to a URL so that for a particular request the corresponding servlet is called. To implement this, you need to configure the servlet. Following are the ways to configure a servlet:

- Using the Deployment Descriptor (DD) file
- Using annotations

### Configuring a Servlet Using the DD File

The DD file is an XML file named **web.xml**. It contains configuration information, such as the initialization parameters, URL mappings, welcome files, and security constraints of a servlet in which it resides.

The following code shows the content of a sample **web.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <servlet>
        <servlet-name>CurrentDate</servlet-name>
        <servlet-class>mypackage.CurrentDate</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CurrentDate</servlet-name>
        <url-pattern>/CurrentDate</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

Some of the commonly used elements of the **web.xml** file along with their usage are:

- **<servlet>**: It is the root tag, which specifies the configuration details of all the servlets used in a Web application. It consists of the following nested tags:
  - **<<servlet-name>**: It specifies a logical name of a servlet class.. For example, in the preceding code, the **<servlet-name>** tag specifies a logical servlet name `CurrentDate`, for the servlet class, `mypackage.CurrentDate`.
  - **<servlet-class>**: It specifies the name of the servlet class, which defines the servlet. For example in the preceding code, the **<servlet-class>** tag specifies the name of the servlet class, `mypackage.CurrentDate`.



- **<servlet-mapping>:** It consists of the following nested tags:
    - **<servlet-name>:** It contains the same name as specified in the `<servlet-name>` tag nested under the `<servlet>` tag.
    - **<url-pattern>:** It specifies the URL pattern using which a servlet can be accessed. For example in the preceding code snippet, the `<url-pattern>` tag associates the `/CurrentDate` URL pattern to the servlet named `CurrentDate`.
- In addition, you can associate a servlet to multiple URL patterns. This is accomplished using multiple `<url-pattern>` tags or by using the wildcard symbol(`*`), as shown in the following code snippet:

```
<servlet-mapping>
<servlet-name>CurrentDate</servlet-name>
<url-pattern>/CurrentDate</url-pattern>
<url-pattern>/MyServlet/*</url-pattern>
</servlet-mapping>
```

The preceding URL patterns will be resolved in the client browser as:

```
http://localhost:8080/MyAppRoot/CurrentServlet
http://localhost:8080/MyAppRoot/MyServlet
http://localhost:8080/MyAppRoot/MyServlet/one
http://localhost:8080/MyAppRoot/MyServlet/twenty
```



*MyAppRoot in the preceding URLs refers to the root of the Web application.*

- **<session-config>:** It specifies the session information for the servlet, such as the session timeout value, as shown in the following code snippet:

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

The preceding code snippet specifies that the servlet session will expire after 30 minutes.

In addition to the preceding elements, the **web.xml** file can consist of the following elements:

- **<context-param>**: It is used to specify the servlet context initialization parameters of a Web application. For example, the following code snippet provides the database connectivity information in the initialization parameter:

```
<context-param>
<param-name> JDBC Driver </param-name>
<param-value>
Class.forName("oracle.jdbc.driver.OracleDriver");</param-value>
</context-param>
```

- **<init-param>**: It specifies the initialization parameter for a servlet. Unlike the context initialization parameter that is available to all the servlets of a Web application, this parameter is available only to the servlet for which it is declared. The following code snippet shows the usage of the `init-param` element:

```
<init-param>
<param-name> title </param-name>
<param-value> This is the First Servlet </param-value>
</init-param>
```

- **<mime-mapping>**: It specifies the mapping between a file extension and a MIME type, as shown in the following code snippet:

```
<mime-mapping>
<extension>html</extension>
<mime-type> text/html </mime-type>
</mime-mapping>
```

Configuring the servlets by using the **web.xml** file involves extensive coding and is inconvenient during the development process. To simplify this, process annotations can be used.

## Configuring a Servlet by Using Annotations

*Annotation* is a metadata that can be added to the code. It is a keyword that is used to add extra explanation to various programming elements, such as classes, fields, and methods in a program. Annotations are primarily one line code that may further include elements for additional values.

Annotations can also be used to simplify the configuration process of a servlet. The `javax.servlet.annotation` package defines the `@WebServlet` annotation that you can use to configure a servlet.

Consider the following code snippet:

```
@WebServlet(name="CurrentDate",
urlPatterns={"/CurrentDate","/MyServlet/*"})
public class CurrentDate extends HttpServlet
{
    ...
}
```

The preceding code snippet uses the `@WebServlet` annotation to assign the name, `CurrentDate`, to the servlet and associate it to a set of URL patterns, such as `/CurrentDate` and `/MyServlet/*`.

Once the servlet is configured, it needs to be compiled and packaged.

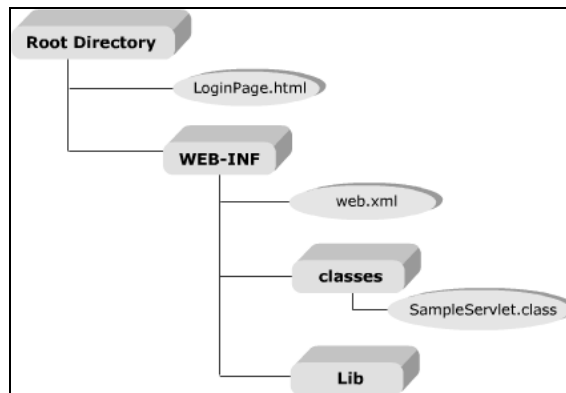
## Compiling and Packaging a Servlet

After the servlet is created and configured, it needs to be compiled to generate a servlet class file. While compiling you need to add the classpath of the servlet API using the `-cp` option of the `javac` utility provided in the JDK, if the classpath is not set. After compiling the servlet, you need to package the servlet into the Java EE application structure.

Java EE defines a standard packaging structure to package a servlet into a Java EE application. This makes the Java EE application portable across different application servers. This allows application servers to easily locate and load application files from the standard directory structure. To create the packaging structure of a Web application, you need to create the following directories:

- **The root directory:** The root directory contains static resources, such as HTML files, JSP files, and image files. For example, you can place an HTML file or JSP file in the root directory.
- **The WEB-INF directory inside the root directory:** The **WEB-INF** directory contains the application deployment descriptor file, **web.xml**, which stores various configurations of a Web application.
- **The classes directory:** The **classes** directory present in the **WEB-INF** directory contains the class files of the application. For example, you can place the **SampleServlet.class** file of the `SampleServlet` servlet in the classes directory.
- **The lib directory:** The **lib** directory present in the **WEB-INF** directory contains the Java Archive (JAR) files of libraries that are required by the application components.

The following figure shows the standard packaging structure of a Java Web application.



*The Standard Packaging Structure*

After you have placed your application specific files in the standard directory structure, you need to package the application into a Web Archive (WAR) file. A WAR file is an archived form of a Java EE Web application. To create a WAR file, type the following command at the command prompt:

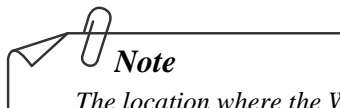
```
jar cvf <war filename>
```

This creates the .war file that you can use to deploy the application in the application server.

## Deploying a Servlet

After a Web application has been packaged, it needs to be deployed on the application server. You need to run the application server before an application can be deployed on it. Most application servers provide utilities that automatically deploy a servlet on the application server. However, you can manually deploy a servlet on the application server by creating the same packaging structure of the Web application on the application server. For example, if you are deploying your Web application on the GlassFish server, then you need to copy the WAR file of the Web application at the following location:

```
[GlassFish installation directory]/glassfish/domains/domain1/autodeploy
```



*The location where the WAR file is deployed varies from one Web server to another.*

## Accessing the Servlet from a Browser

Finally, you need to test the Java EE application that you have deployed. You can test the application by accessing it from the Web browser. The syntax of the URL that you use to access the servlet is:

```
http://<server_name>:<portnumber>/<context root>/<Servlet-name>
```

For example, to access the servlet, `SampleServlet`, with `serv_app` as the context root, you can use the following URL:

```
http://192.168.0.52:8080/serv_app/ SampleServlet
```



### Activity 2.1: Creating a Servlet



#### ***Just a minute:***

The \_\_\_\_\_ contains the static resources, such as *HTML* files, *JSP* files, and image files of a Web application.

#### ***Answer:***

*Root directory*

## Practice Questions

1. Which one of the following features of the servlets enables it to serve multiple client requests and generate responses simultaneously?
  - a. Efficient
  - b. Portable
  - c. Persistent
  - d. Asynchronous support
2. Which one of the following methods is called by the Web container after creating an instance of a servlet?
  - a. `service()`
  - b. `init()`
  - c. `getServletConfig()`
  - d. `getServletInfo()`
3. Which one of the following objects contains the initialization information of a servlet?
  - a. `ServletConfig`
  - b. `ServletContext`
  - c. `ServletRequest`
  - d. `ServletResponse`
4. Which one of the following objects allows interaction of the servlets with the Web container?
  - a. `ServletConfig`
  - b. `ServletContext`
  - c. `ServletRequest`
  - d. `ServletResponse`
5. Which one of the following directories contains the **web.xml** file of the Java Web application?
  - a. WEB-INF
  - b. classes
  - c. lib
  - d. root

## Summary

In this chapter, you learned that:

- Servlet is a Java program that runs on the server side. It inherits all the features of the Java programming language.
- The Servlet API contains various interfaces, classes, and methods that are used to create and manage servlets. These classes and interfaces are available in the following packages:
  - `javax.servlet`
  - `javax.servlet.http`
- The `Servlet` interface of the `javax.servlet` package defines the methods that the Web container calls to manage the servlet life cycle.
- The `javax.servlet.ServletConfig` interface is implemented by a Web container during the initialization of a servlet to pass the configuration information to the servlet.
- The `javax.servlet.Servlet` interface defines the life cycle methods of a servlet, such as `init()`, `service()`, and `destroy()`.
- The Web container invokes the `init()`, `service()`, and `destroy()` methods of a servlet during its life cycle.
- You can configure a servlet by using the:
  - **web.xml** file.
  - `@WebServlet` annotation.
- To create a servlet class, you need to perform the following tasks:
  - Create a servlet that either extends the `HttpServlet` or `GenericServlet` class.
  - Configure the servlet.
  - Compile the servlet.
  - Package the servlet.
  - Deploy the servlet on the application server.
  - Access the servlet using a browser application.
- To create the packaging structure of a Web application, you need to create the following directories:
  - A root directory
  - A `WEB-INF` directory inside the root directory
  - A classes directory
  - A lib directory

