Client Server Application Programming

Week 9: Java Servlets - How Servlets Work, Using Servlets, Writing a Simple Servlet, Running Servlets, Single Thread Model etc

By Elubu Joseph - MSc.IS

Lecturer

Department of Information Technology

Kumi University

Email: josebulinda@gmail.com

jose@kumiuniversity.ac.ug

Summary of previous lecture

1. Overview of Application protocol
2. Application Protocol Specifications,
3. Application Protocol Implementation,
4. HyperText Transfer Protocol and Java,
5. Common Gateway Interface (CGI)

Agenda

1. Java Servlets Overview
2. How Servlets Work,
3. Using Servlets,
4. Writing a Simple Servlet and Running it,
5. Single Thread Model,
6. Servlet Request and http Servlet Request and
7. Servlet Response and http Response

Java Servlets Overview

Java Servlets Overview

Java servlets are server-side applications that execute similar to CGI scripts, but without a separate process for each request (which consumes both CPU and memory resources).

Servlets are multithreaded, and thus can share resources across servlet instances.

The performance of servlets increase due to the use of a compiled language compared to the use of scripting interpreter. This makes servlets a good choice for developers. Of course, the most obvious benefit to servlet developers is that servlets are written in Java

Java Servlets Overview

Java servlets form part of a broad complement of technologies known as the Java 2 Enterprise Edition (J2EE).

These technologies are suited from medium to large-scale application development, and include topics such as Jakarta Enterprise Edition, Java Server Pages (JSPs), and much more.

Note!. Such technologies merit book-length coverage in their own right, and are beyond the scope of an introductory network programming lession such as this.

How Servlets Work

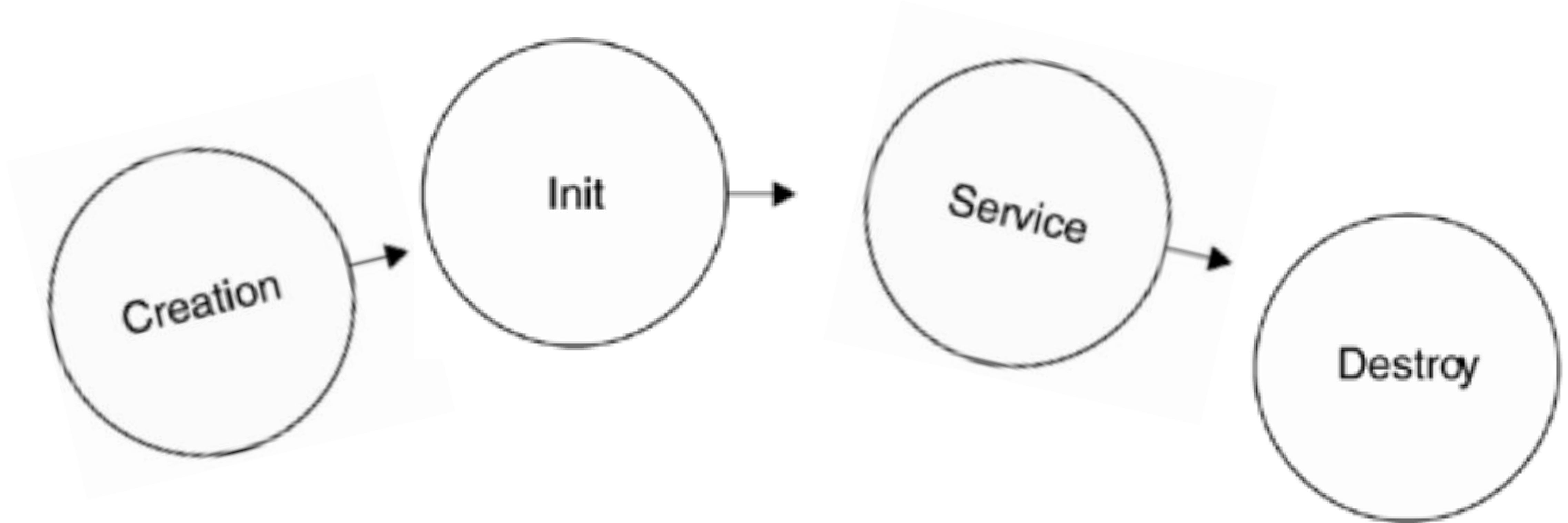
How Servlets Work

When a client sends a request to a servlet-enabled server that invokes a servlet, the server first checks to see if the servlet is loaded. If it is not, the servlet class is loaded and a new instance is created.

The servlet is then initialized by a call to its **init()** method, which can contain startup code similar to the **init()** method of an applet. Once the servlet is ready, a call to its **service()** method is made.

How Servlets Work – Servlet life cycle

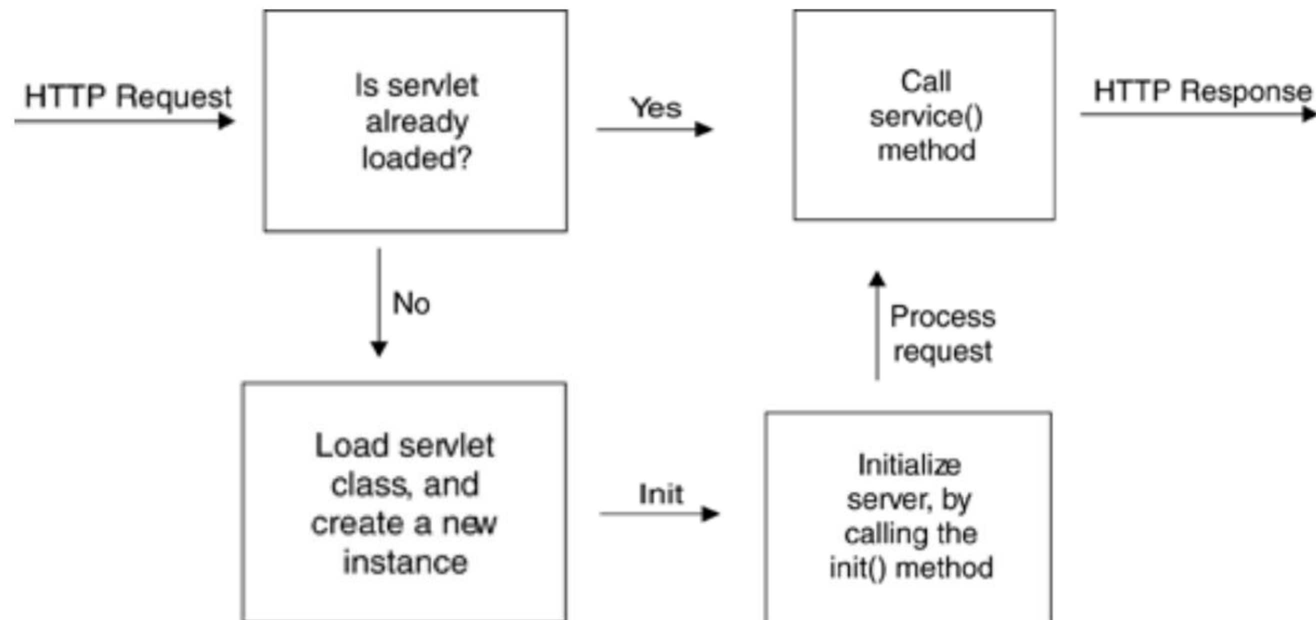
This lifecycle is relatively simple, as shown below.



How Servlets Work+

Servlet instantiation, initialization, and processing of a request

The Figure below shows the process by which a servlet is loaded, initialized, and made ready to service client requests in a servlet-enabled Web server.



As already mentioned, different servers may use servlets over different protocols, such as HTTP or even FTP, since the specification was written to be generic. By default, Sun provides an HTTP implementation only for servlets, and in this chapter we will use that implementation.

However, the information can apply to other servlet implementations as well, and you may find them of use in developing your own custom protocol servers.

Using Servlets

Using Servlets

Servlets rely on classes defined in the **javax.servlet** and **javax.servlet.http** packages. These packages are a standard extension to the Java API, and ship with the Java Servlet Development Kit (JSDK) as well as many servlet-compatible Web servers.

The starting point for a servlet is the servlet interface. This interface provides the basic structure methods for servlets, such as the initializing, service, and destruction methods.

It also provides structure methods for accessing the servlet's context and configuration, as shown later in this section. By default, servlets that implement this interface use a shared threading model that drastically improves the performance of servlets to handle large numbers of requests.

Using Servlets+

The GenericServlet provides a generic protocol-independent starting point with simple implementations of **init()** and **destroy(),** so that the developer needs only to extend service. However, if the goal is to write an HTTP servlet that is used for the Web, the HTTPServlet class should be extended; it extends from GenericServlet and provides the starting point for Web servlets.

Unless the functionality is changed, the **service()** method of HttpServlet will check the type of HTTP request sent by the browser and pass it off to special handler functions. While it is possible to write a custom **service()** method, in most cases it is easier to use the default service method.

For each of the major HTTP request types, there is a corresponding handler function that can be overridden by developers. The Table in the next slide shows the mapping between HTTP requests and servlet methods.

Using Servlets+++

Mapping of HTTP Request Types to Servlet Methods

HTTP Request	Servlet Handler Function
GET	`doGet (HttpServletRequest, HttpServletResponse)`
POST	`doPost (HttpServletRequest, HttpServletResponse)`
PUT	`doPut (HttpServletRequest, HttpServletResponse)`
DELETE	`doDelete (HttpServletRequest, HttpServletResponse)`
TRACE	`doTrace (HttpServletRequest, HttpServletResponse)`
OPTIONS	`doOptions (HttpServletRequest, HttpServletResponse)`

Developers are free to override one, several, or all of these handler functions. However, if a handler function is not overridden, it will return an **HTTP_BAD_ REQUEST error response**.

For example, suppose a form servlet supported only the POST request. Here is a sample of what such a transaction might look like.

Using Servlets+++

Mapping of HTTP Request Types to Servlet Methods

```
public void doGet (HttpServletRequest req, HttpServletResponse res)
 throws IOException {

      doPost(req,res);
}
```

NOTE: Since all of the handler functions take the same parameters (an instance of HttpServletRequest and HttpServletReponse), it is easy to write a handler for either GET or POST that passes control off to the other. This means that if you accidentally specify the wrong HTTP method in your HTML forms, the servlet will handle it gracefully. As seen above.

Using Servlets+++

Mapping of HTTP Request Types to Servlet Methods

Once a servlet handler function is called, it can respond to the request and its parameters, perform some processing, and then output a response. However, developers should be mindful of the fact that each servlet is capable of servicing multiple requests simultaneously, so normal thread safety issues apply.

Finally, when the Web server shuts down or terminates a servlet due to inactivity after a specified period of time, the destroy() method will be invoked. This is the servlet's last opportunity to close open files, remove temporary or unnecessary data files, commit changes to disk, or close open network/database connections.

As with applets, **servlets have a definite lifecycle** from creation to destruction. First the **init()** method is called, followed by one or more **service()** requests, and finally a **destroy()** call is made.

GET and POST

The GET and POST HTTP request types are used to retrieve data from a Web server, and both have the ability to encode parameters into the request. The only **major difference between** them is that GET has a **limit of 255** characters for its parameters, while **POST** has **no limit**.

A GET request involves using a question mark in the URL to signify that the rest of the URL includes encoded parameters. An example of a simple GET request that encodes parameters is:

```
GET /servlet/Query?name=tom&age=28 HTTP/1.0
```

A POST request, on the other hand, **does not modify the URL that is requested**. Instead, the POST request is followed by the actual parameters that are sent to the server-side script or servlet. An example of a simple POST request, which performs the same job as the previous GET request, is the following:

GET and POST +

```
POST /servlet/Query HTTP/1.0 name=tom age=28
```

According to the Java Servlet API, these two types of requests are treated completely differently. However, these differences are transparent to the developer, as the API provides generic methods that abstract the differences between them.

In many cases, it is easier to write a **single handler** than to divide the work between a GET and a POST handler.

Assuming that you would like your servlet to be easily manageable, you can refer both GET and POST requests to a single handler, as shown in this example:

GET and POST ++

```
public void doGet(HttpServletRequest req,
HttpServletResponse res) throws IOException {
    doGetPost(req,res);
}
public void doPost(HttpServletRequest req,
HttpServletResponse res) throws IOException {
    doGetPost(req,res);
}
public void doGetPost( HttpServletRequest req,
HttpServletResponse res ) throws IOException {
    //Implementation goes here
}
```

PUT and DELETE

were are used for adding and removing files from the Web server's storage space. For example, you could use PUT to send a new file across, and DELETE to remove one. Of course, their usefulness in most applications is limited—the thought of someone overwriting your page, uploading a rogue servlet, or deleting your Web site is fairly frightening.

TRACE and OPTIONS

The TRACE method is used to invoke a remote, application-layer loop-back of the request message.

It is used as a diagnostic test of the servlet request, and it is not necessary to override this method to provide an implementation. If an override is necessary for debugging purposes, however, Sun has included this capability.

TRACE and OPTIONS Requets

The OPTIONS request type in HTTP is used to probe the server in general or a single servlet about **what types of HTTP request types are valid**. For instance, when you extend the HTTPServlet and implement only the doGet method, you receive the following output from an OPTION request:

```
Allow: GET, HEAD, TRACE, OPTIONS
```

Unless you plan on adding extra methods that are beyond HTTP/1.1, you do not need to implement this method.

Running Servlets

To run a servlet, you need either a Web server with servlet support or a servlet engine that will augment an existing server.

A popular choice of Web server for Java developers is the **iPlanet Web Server**. This Web server was originally developed as the **Netscape Enterprise Server** and the **Java Web Server**. However, Sun and Netscape created an alliance, and the **iPlanet Web Server** took the best of both these servers. It includes built-in support for servlets and other dynamic server-side features.

Another common choice is the free open-source **Apache**, which runs on a variety of Unix platforms as well as Windows. **Our choice in this course.**

Running Servlets+

Apache can be easily modified to support servlets with the **Apache JServ add-on**. If you're using other servlets, products like JRun and ServletExec can add support to existing servers, such as Netscape servers, O'Reilly's Website, and Microsoft's Internet Information Server (IIS). If you're planning on doing a lot of servlet development, it might be wise to evaluate several options through links below.

1. Apache (http://www.apache.org/)
2. Apache JServ (http://java.apache.org/jserv/)
3. Apache's Jakarta-Tomcat (http://jakarta.apache.org/)
4. JRun (http://www.allaire.com/products/jrun/)
5. ServletExec (http://www.newatlanta.com/)
6. BEA Weblogic (http://www.beasys.com/)
7. Borland Enterprise Application Server (http://www.inprise.com/bes/appserver/) integrated with JBuilder

Downloading the Java Servlet Development Kit

Sun includes a basic Web server with servlet support as part of the free Java Servlet Development Kit (JSDK), which can be downloaded from the Sun Microsystems Web site. Downloading this tool will be essential if you would want to use it for this purpose; Sun Microsystems Web site at http://java.sun.com/products/servlet/archive.html.

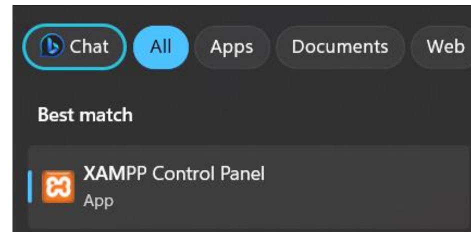
However, if you have NetBeans Installed on a given JDK environment, then you are good to go. There is no need to download the above JSDK. We will build our servlets based on NetBeans Projects options.

Installing a Servlet Engine

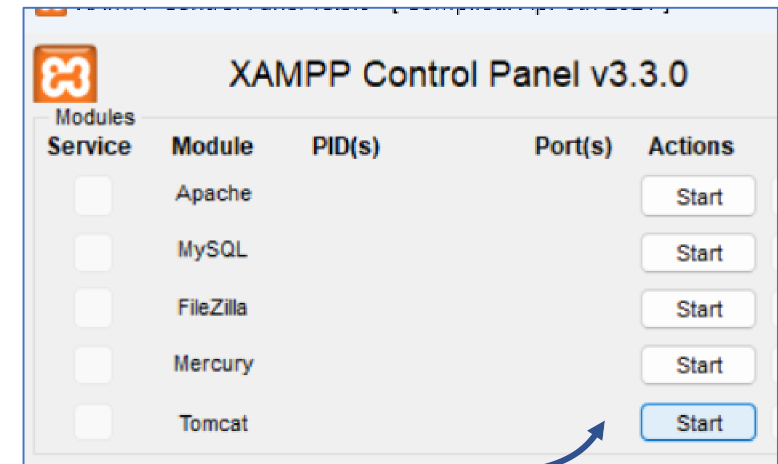
1. Make sure your xampp server(come with tomcat) or Tomcat server is installed
2. Make sure your NetBeans IDE is installed and running. For details on how to download and install the above, check on Lecture 3 of Object Oriented programming 1 (https://www.hufocw.org/Course/960) course by Elubu Joseph.

Configuring TomCat server in XAMPP

1. Open xampp

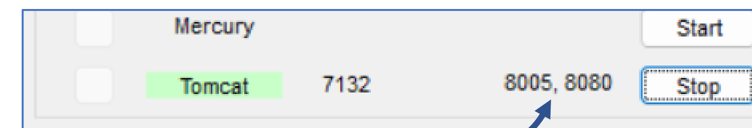


2. Click start Button on the right of Tomcat



3. If Tomcat is successfully started,

You will see the ID and Port numbers as below

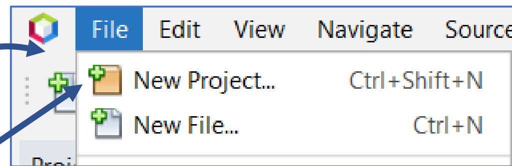


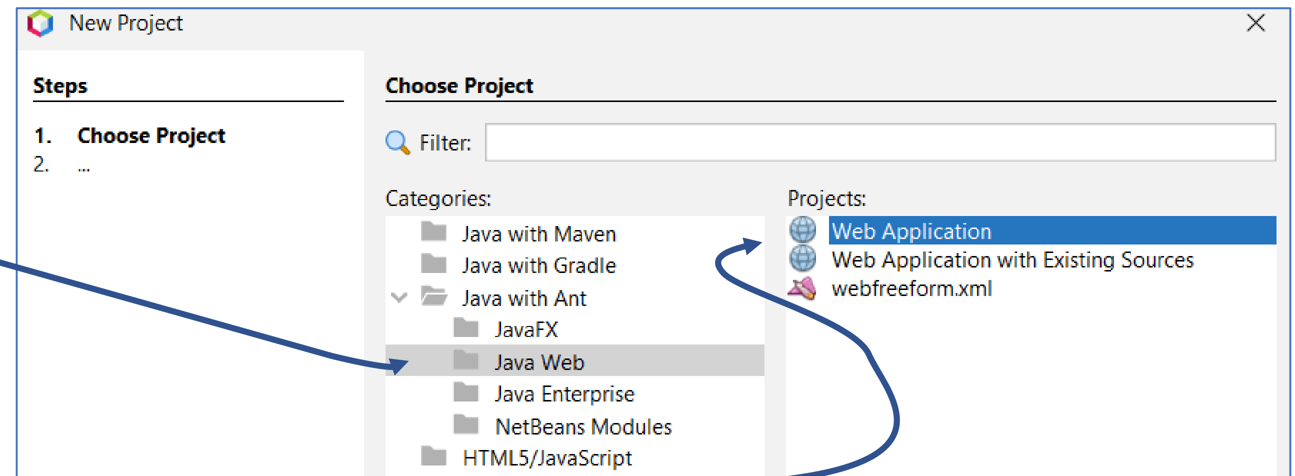
Configuring TomCat server in XAMPP+Creating Web service Project in Netbeans

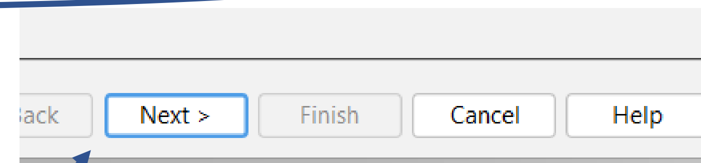
Open NetBeans

Create a Web Service Project

1. Click on File,
2. Click New Project
3. Under Java With Ant, Java Web
4. Under Projects, Select Web Application
5. Click Next

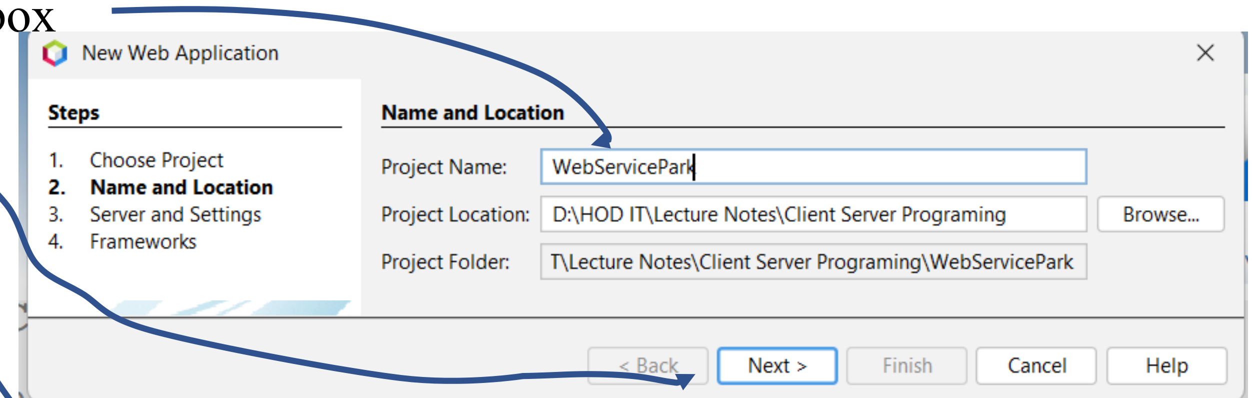


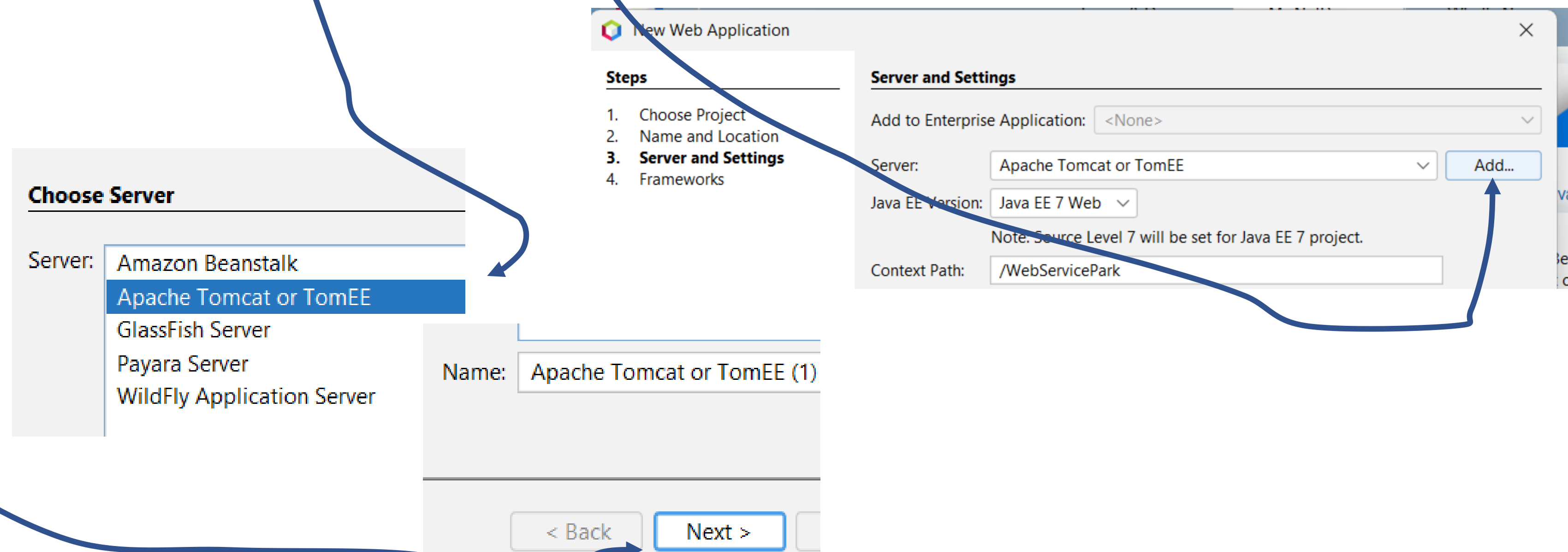




Configuring TomCat server in XAMPP+Creating Web service Project in Netbeans+

6. Enter Project Name in the Project Name text box
7. Click Next
8. Under Server and Settings, Click Add button,
9. Select Apache Tomcat or Tom EE
10. Click Next





Configuring TomCat server in XAMPP+Creating Web service Project in Netbeans++

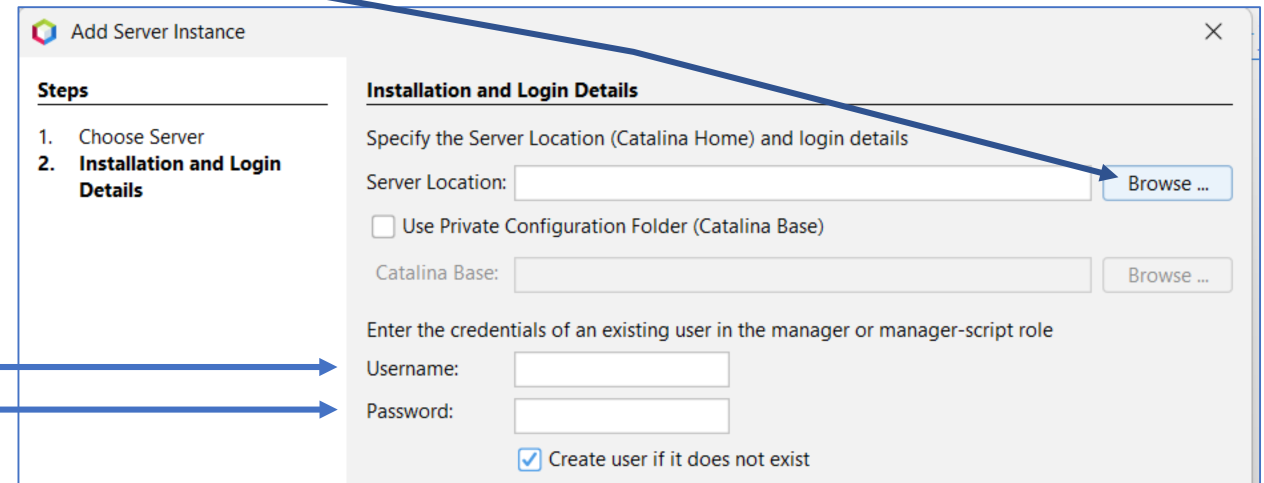
11. Under installation and Login details, click Browse button

12. Navigate to the folder containing Tomcat

Tomcat Installation folder e.g.

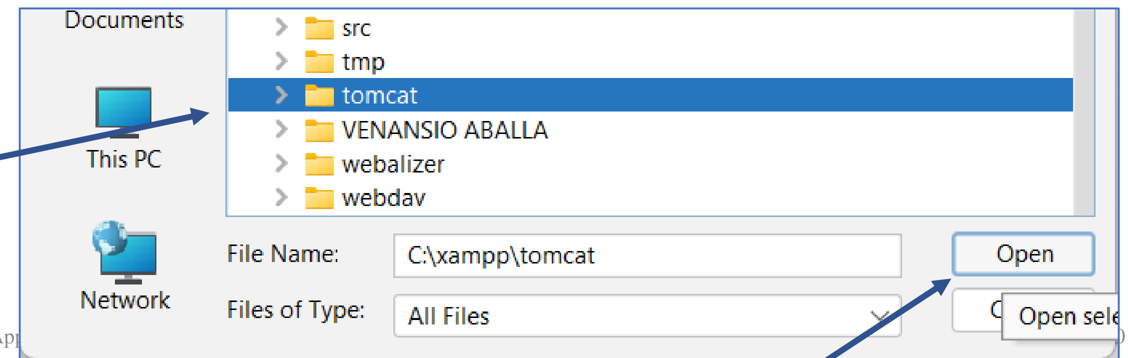
C:\xampp\tomcat, if one installed xampp in C:\

13. Click Open, the finsih



14. Enter User Name

15. Enter Password



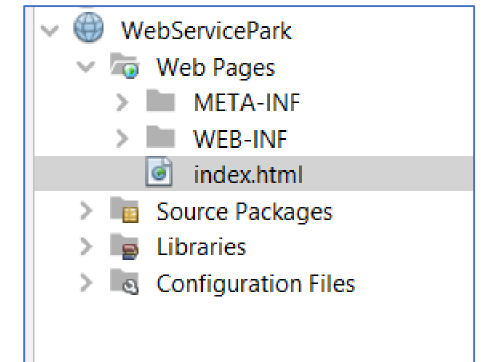
Configuring TomCat server in XAMPP+Creating Web service Project in Netbeans++++

16. Then Click Finish

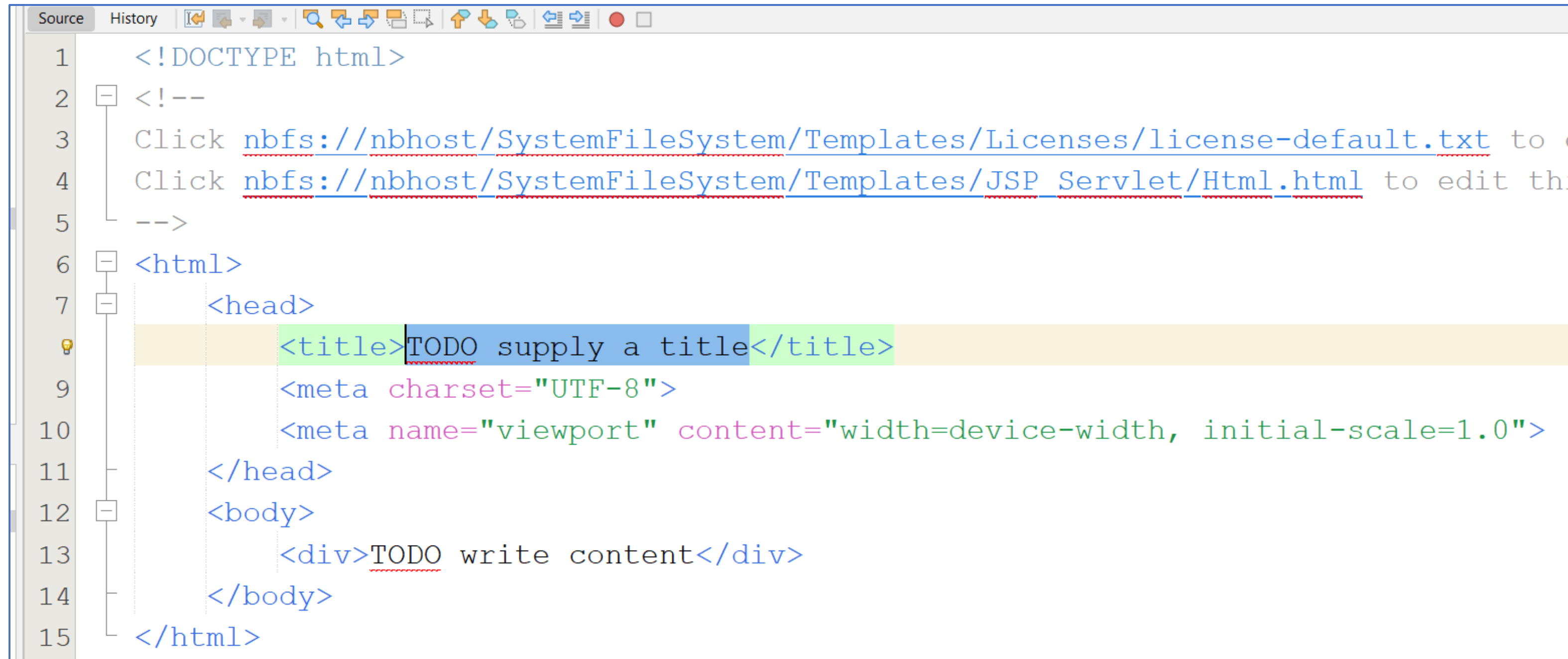
17. Click Next

18. Then click Finish

19. Successful? You should be seeing on the projects panel.



Configuring TomCat server in XAMPP+Creating Web service Project in Netbeans, done



If you are seeing the page on the right hand side of your screen then we have successfully configured our tomcat server to the web service. We can now go a head and create our First servlet.

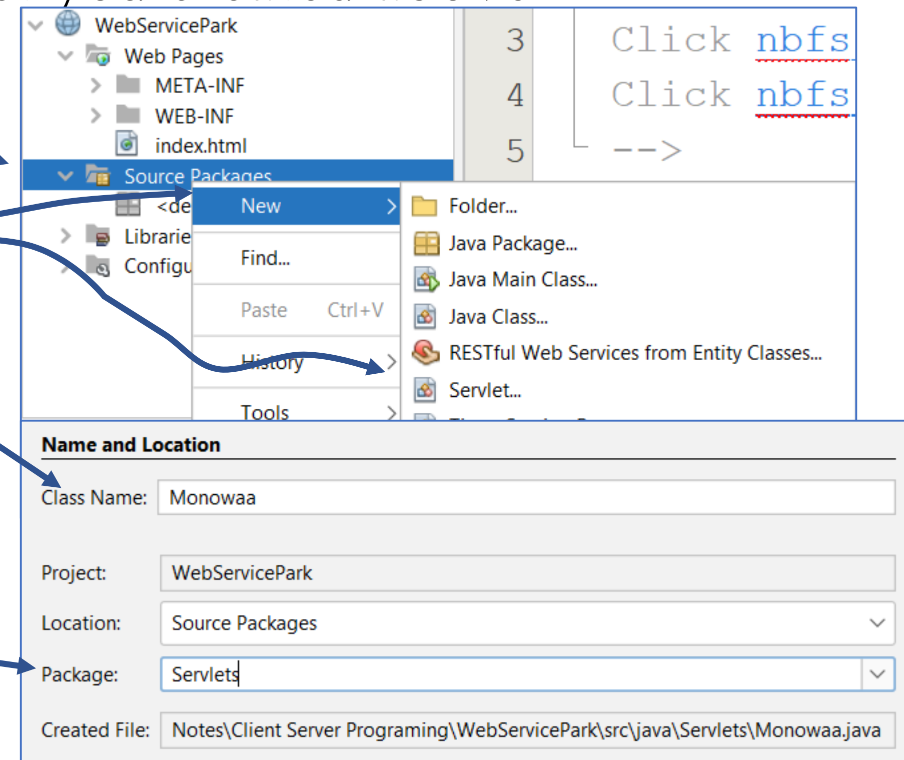
Writing a Simple Servlet and Running it,

Before we start creating a servlet, we need to note that Users of Tomcat 10 onwards, as a result of the move from Java EE to Jakarta EE as part of the transfer of Java EE to the Eclipse Foundation, the primary package for all implemented APIs has changed from javax.* to jakarta.*. This will almost certainly require code changes to enable applications to migrate from Tomcat 9 and earlier to Tomcat 10 and later. A migration tool has been developed to aid this process.

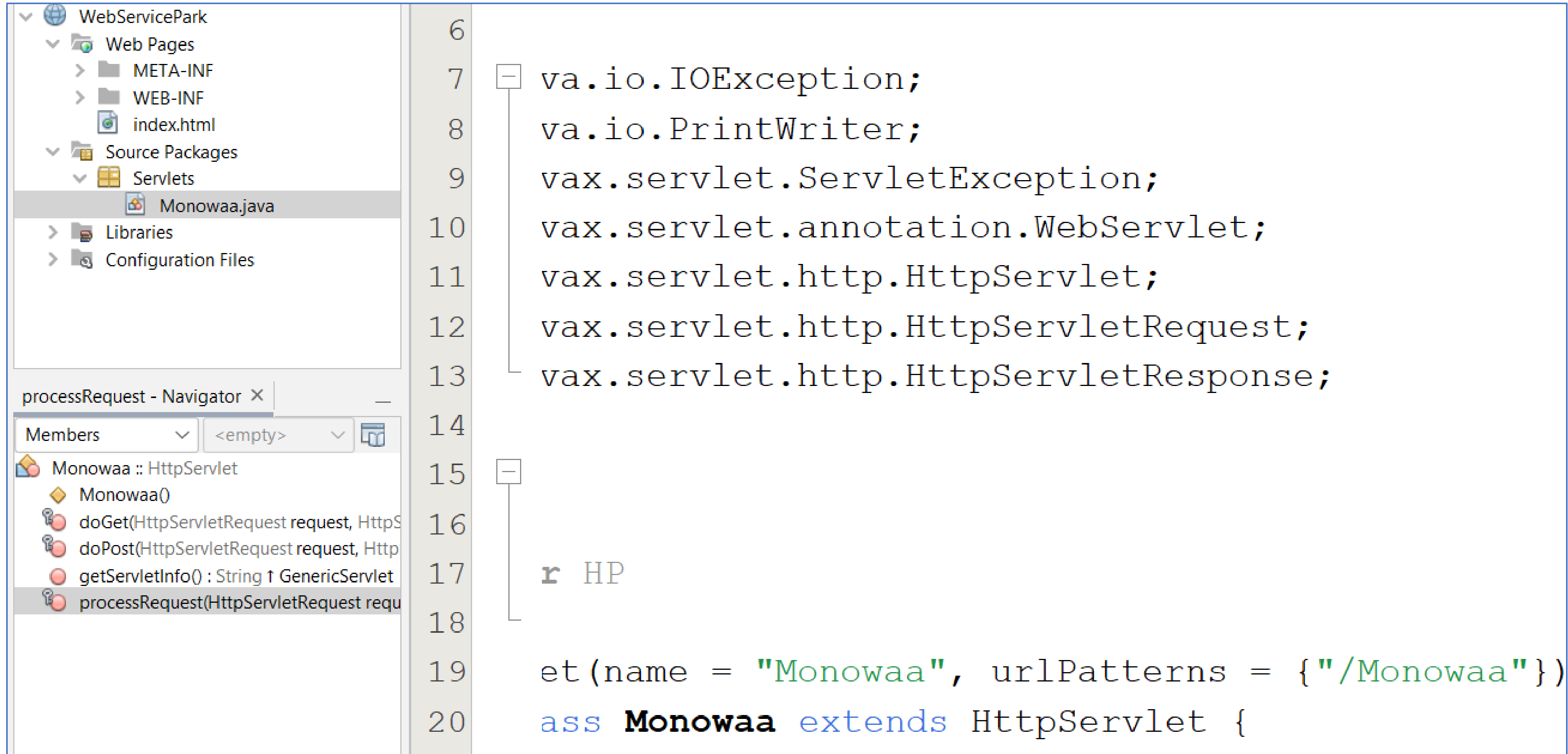
Note! in our case we will not need to correct our code directly since we are not running Tomcat 10++

Writing a Simple Servlet and Running it - **Monowaa** Servlet

1. Right click on the Source Packages of the project you created above
2. Point at New
3. Click Servlet
4. Enter the Servlet name into
5. the Class name TextBox e.g. Monowaa
6. Enter package name if you don’t have already
7. Click Next
8. Click Finish



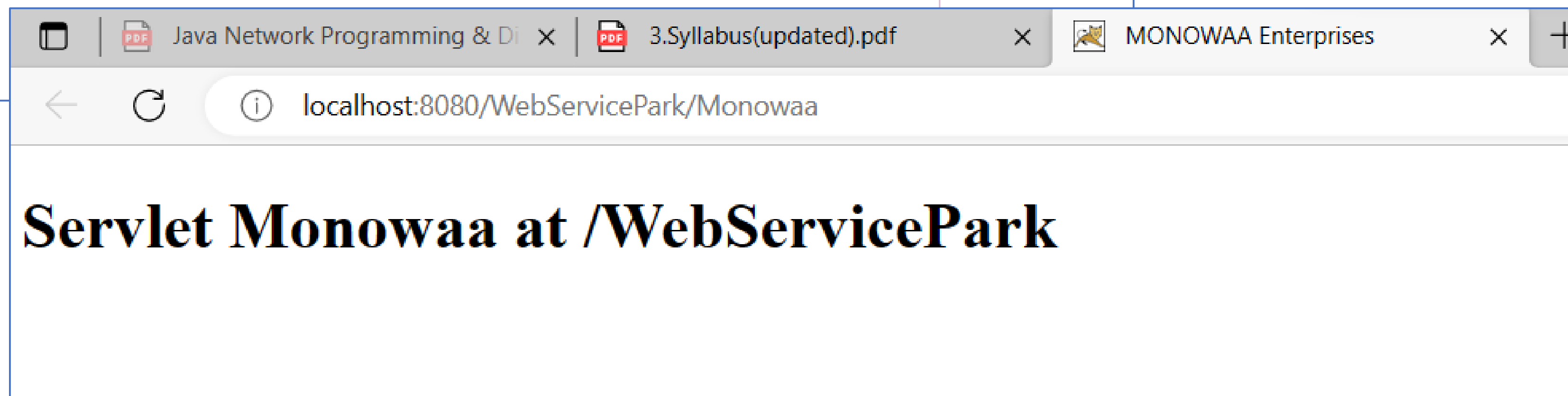
Writing a Simple Servlet and Running it - Monowaa Servlet+code



Writing a Simple Servlet and Running it - Monowaa Servlet+code

```
31    cted void processRequest(HttpServletRequest request, HttpServletResponse respons
32       throws ServletException, IOException {
33    esponse.setContentType("text/html;charset=UTF-8");
34    ry ( PrintWriter out = response.getWriter()) {
35       /* TODO output your page here. You may use following sample code. */
36       out.println("<!DOCTYPE html>");
37       out.println("<html>");
38       out.println("<head>");
39       out.println("<title>MONOWAA Enterprises</title>");
40       out.println("</head>");
41       out.println("<body>");
         out.println("<h1>Servlet Monowaa at " + request.getContextPath() + "</h1>");
43       out.println("</body>");
44       out.println("</html>");
45
```

Running Servlet on the web browser



Single Thread Model,

The SingleThreadModel interface in the Java Servlet API was introduced to address a potential issue with thread safety in servlets.

In the early versions of the Servlet API, a single instance of a servlet was created to handle multiple concurrent requests. This meant that if multiple requests were received at the same time, the servlet instance had to handle them concurrently, leading to potential thread safety problems.

To mitigate these issues, the SingleThreadModel interface was introduced. When a servlet implements this interface, the servlet container ensures that only one thread executes the service methods (such as doGet() or doPost()) at a time for that particular servlet instance. Essentially, it enforces a single-threaded processing model for the servlet.

Single Thread Model,

By using the SingleThreadModel, the servlet container creates multiple instances of the servlet and assigns one instance to each incoming request, effectively serializing the processing of the requests. This way, even if multiple requests are received simultaneously, each request will be handled by a separate servlet instance, eliminating the need for explicit synchronization within the servlet.

It's important to note that the SingleThreadModel interface has **been deprecated since** the Servlet API version 2.4 and is not recommended for use in modern servlet development. The reason for deprecating it is mainly due to its scalability limitations and potential performance issues.

Note! Servlet containers are now designed to handle concurrent requests efficiently without the need for explicit synchronization within servlets.

Servlet Request and http Servlet Request

Servlet Request and http Servlet Request

ServletRequest and **HttpServletRequest** are interfaces in the Java Servlet API that represent the client's request to the server. They provide methods to retrieve information about the request, such as request parameters, headers, cookies, and the request body.

ServletRequest is the base interface for representing a client's request. It provides methods to access general request information, such as the protocol, scheme, server name, server port, and the input and output streams associated with the request.

It also allows retrieval of request attributes, which are objects that can be set by the server or other components to store and share data during the processing of a request.

Servlet Request and http Servlet Request+

HttpServletRequest is a subinterface of ServletRequest that extends its functionality by providing additional methods specifically for HTTP requests. It includes methods to retrieve HTTP-specific information, such as the request method (GET, POST, etc.), request URI, query parameters, and the client's IP address. It also offers methods to access HTTP headers, cookies, and session-related information.

Here are some commonly used methods provided by HttpServletRequest:

1. **getMethod():** Returns the HTTP method (GET, POST, PUT, DELETE, etc.) of the request.
2. **getRequestURI():** Returns the portion of the request URL from the context path to the end of the URL.
3. **getParameter(String name):** Retrieves a request parameter by its name.
4. **getHeader(String name):** Returns the value of the specified HTTP header.

Servlet Request and http Servlet Request++

5. **getCookies():** Returns an array of Cookie objects sent by the client.
6. **getSession(boolean create):** Retrieves the HttpSession associated with the request. If create is true and no session exists, a new session will be created.
7. **getInputStream():** Returns the input stream that contains the request body.

Both **ServletRequest** and **HttpServletRequest** are implemented by the **servlet container** and are passed as parameters to the **service()** or doxxx() methods of a servlet, allowing the servlet to access and process the client's request data.

Servlet Response and http Response

A Servlet Response and an HTTP Response are related concepts in web development, but they represent different aspects of the request-response cycle.

1. **Servlet Response:** is an object that a servlet generates to send back to the client in response to an HTTP request. It represents the response data that will be sent from the server to the client. When a servlet receives a request, it processes it and generates a response by manipulating the Servlet Response object. The Servlet Response contains information such as the response headers, response status code, and the response body.

The Servlet Response object provides methods to set response headers, such as Content-Type, Content-Length, and Cache-Control. It also allows you to get the output stream or writer to write the response content. Additionally, you can set the response status code to indicate the status of the request, such as 200 for a successful response, 404 for a resource not found, or 500 for an internal server error.

Servlet Response Sample code

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

public class MyServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
{

        // Set the response content type
        response.setContentType("text/html");
```

Servlet Response Sample code

```
        // Get the response writer
        PrintWriter out = response.getWriter();

        // Generate the response HTML
        out.println("<html>");
        out.println("<head><title> Sample Servlet Response </title></head>");
        out.println("<body>");
        out.println("<h1> Hello, Servlet! </h1>");
        out.println("</body>");
        out.println("</html>");

        // Close the writer
        out.close();
    }
}
```

Servlet Response Sample code-Explained

In the example code above,

1. We create a class **MyServlet** that extends **HttpServlet** to handle HTTP requests.
2. We override the **doGet** method, which is called when a GET request is received.
3. Inside the **doGet** method, we set the content type of the response to "**text/html**" using the **setContentType** method of the response object.
4. We obtain the **PrintWriter** object from the response using **response.getWriter(),** which allows us to write the response content.
5. We generate a simple HTML response using the println method of the **PrintWriter**. In this case, we print a basic HTML page with a title and a heading.
6. Finally, we close the **PrintWriter**.

When a client makes a GET request to the servlet, the servlet will generate this HTML response and send it back to the client. The client's browser will then render the HTML and display the "Hello, Servlet!" message.

Servlet Response and http Response+

2. HTTP Response: An HTTP Response is the actual response sent by the server to the client in the HTTP protocol. It consists of a status line, response headers, and an optional response body. The HTTP Response is responsible for carrying the Servlet Response generated by the servlet back to the client.

The status line in an HTTP Response includes the HTTP version, status code, and a reason phrase. **For example: HTTP/1.1 200 OK**

The response headers provide additional information about the response, such as Content-Type, Content-Length, and Cache-Control. They help the client understand how to interpret the response and handle it appropriately.

The response body contains the actual content of the response, such as HTML, JSON, or binary data. It can be empty if the response doesn't include any content.

In summary, a **Servlet Response** is an object generated by a servlet to represent the response data, while an **HTTP Response** is the actual response sent over the HTTP protocol from the server to the client, carrying the Servlet Response data.

sample code snippet that demonstrates the usage of HTTP Response in

```
import java.io.IOException; import javax.servlet.ServletException; import javax.servlet.http.HttpServlet;

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

public class MyServlet extends HttpServlet {

    @Override

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

        // Set the response status code

        response.setStatus(HttpServletResponse.SC_OK);

}
```

sample code snippet that demonstrates the usage of HTTP Response

```
// Set the response headers
    response.setHeader("Content-Type", "text/html");
    response.setHeader("Cache-Control", "no-cache");
    // Get the response writer
    PrintWriter out = response.getWriter();

    // Generate the response body
    out.println("<html>");
    out.println("<head><title> Sample HTTP Response </title></head>");
out.println("<body>");
        out.println("<h1> Hello, HTTP Response! </h1>");
        out.println("</body>");
        out.println("</html>");
        // Close the writer
        out.close();
    }
}
```

sample code snippet that demonstrates the usage of HTTP Response Explained

In this example,

1. We create a class **MyServlet** that extends **HttpServlet** to handle HTTP requests.
2. We override the **doGet** method, which is called when a GET request is received.
3. Inside the **doGet** method, we set the response status code to **HttpServletResponse.SC_OK**, which represents a successful response with a status code of 200.
4. We set the response headers using the **setHeader()** method of the **response** object. In this case, we set the "Content-Type" header to "text/html" to indicate that the response content is HTML, and we set the "Cache-Control" header to "no-cache" to disable caching.

sample code snippet that demonstrates the usage of HTTP Response Explained

In this example,

5.We obtain the **PrintWriter** object from the response using **response.getWriter()**, which allows us to write the response content.

6.We generate a simple HTML response using the **println** method of the **PrintWriter**. In this case, we print a basic HTML page with a title and a heading.

7.Finally, we close the **PrintWriter**.

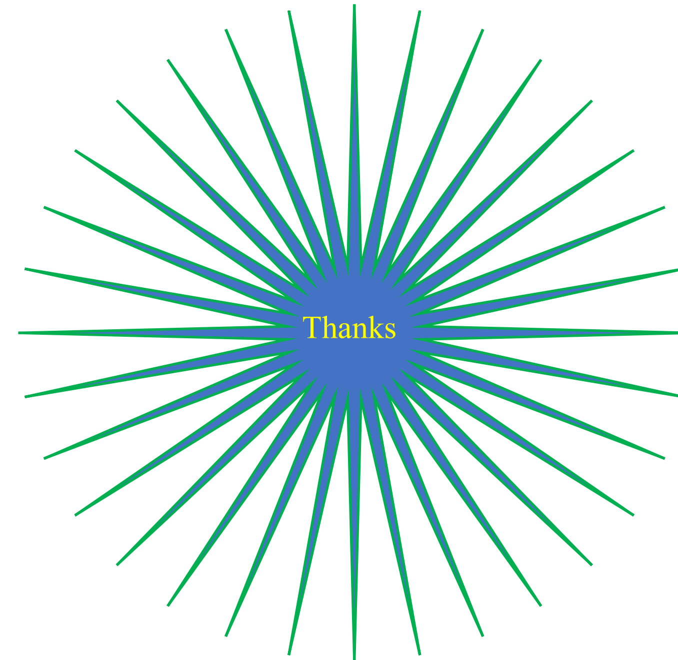
When a client makes a GET request to the servlet, the servlet will send an HTTP response with the specified status code, headers, and HTML content. The client's browser will then render the HTML and display the "Hello, HTTP Response!" message.

Summary

Summary

1. Java Servlets Overview
2. How Servlets Work,
3. Using Servlets,
4. Writing a Simple Servlet and Running them,
5. Single Thread Model,
6. Servlet Request and http Servlet Request,
7. Servlet Response and http Response

Thank you for Listening



References

Java™ Network Programming and Distributed Computing, (David R.,Michael R. 2002), Publisher : Addison Wesley; ISBN: 0201710374

Apache Tomcat®. Apache Tomcat® - Apache Tomcat 10 Software Downloads. (n.d.). https://tomcat.apache.org/download-10.cgi