Day - 3
Servlet Life Cycle

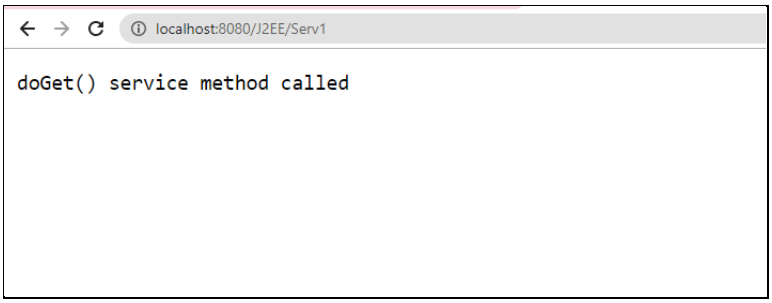
In the previous session, we had seen how we can map the url pattern with the Servlet so that when a client requests for a particular url-pattern, the deployment descriptor(web.xml) will map to the FirstServlet.java and it was executed but the output of the execution was displayed on the console present in the server computer i.e., eclipse and not on the browser window.

So how do we modify the code such that the output should appear on the client side(browser window) ?

If anything has to appear on the browser, then what must be sent should always be a HTML because browsers can render only HTML files and this can be achieved by a special class called **PrintWriter**, which can be assumed as an HTML pen to write on the browser window.
Let us now see how we can achieve that-
Whenever the client or browser wants to send something then we have use **Request object req**, which is of type **HttpServletRequest** and whenever the servlet has to give something back to the client we have use **Response object resp** which is of type **HttpServletResponse** . In this case as we have send something to the client, we have to use **resp.**

```
package com.tap.servlets;

import java.io.IOException;
import java.io.PrintWriter;

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


public class FirstServlet extends HttpServlet {
	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {
		PrintWriter writer = resp.getWriter();
		writer.println("doGet() service method called");
```

Output:



Now we can see that the output is displayed on the client window and since PrintWriter class can take HTML as input,then we can also send tags in **getWriter()** as shown below-

```
PrintWriter writer = resp.getWriter();
writer.println("<h3>doGet() service method called</h3>");
```

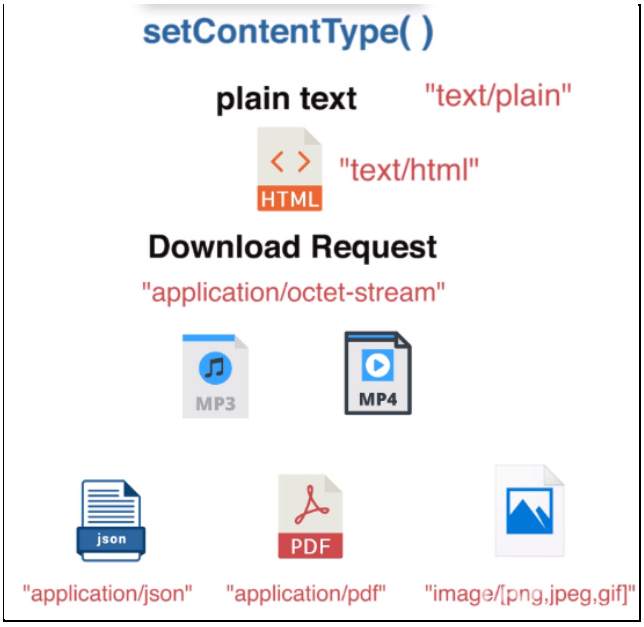
But when we execute this, we will not get the expected output but it will be displayed as plain text because the browser has interpreted the text as plain text rather than considering it to be an HTML text. Browsers can accept different types of responses like-

- ➔ Plain Text
- ➔ HTML content
- ➔ Download request
  - ◆ Audio files
  - ◆ Video files
- ➔ JSON files
- ➔ PDF files
- ➔ Images

So it is very important that the server which is sending a response to the browser to also inform the type of data it is sending.

For that in Servlets we have a method called **setContentType( )**, in which we can set the type of content we want to send as a response.

The setContentType( ) accepts different types of input for different types of responses as shown below -

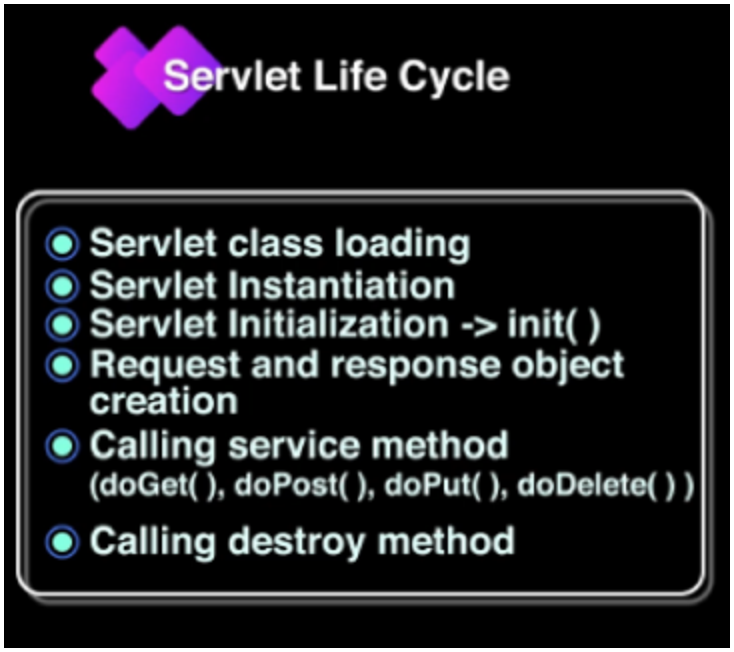


Since we want to send **HTML** as response we have to pass **"text/html"** as parameter inside the setContentType( ) method, as shown below-

```
resp.setContentType("text/html");
PrintWriter writer = resp.getWriter();
writer.println("<h3>doGet() service method called</h3>");
```

Output:

How is it possible for the servlet to get executed even without the main method?
For that we first have to understand **Servlet Life Cycle,** which is a very important concept.

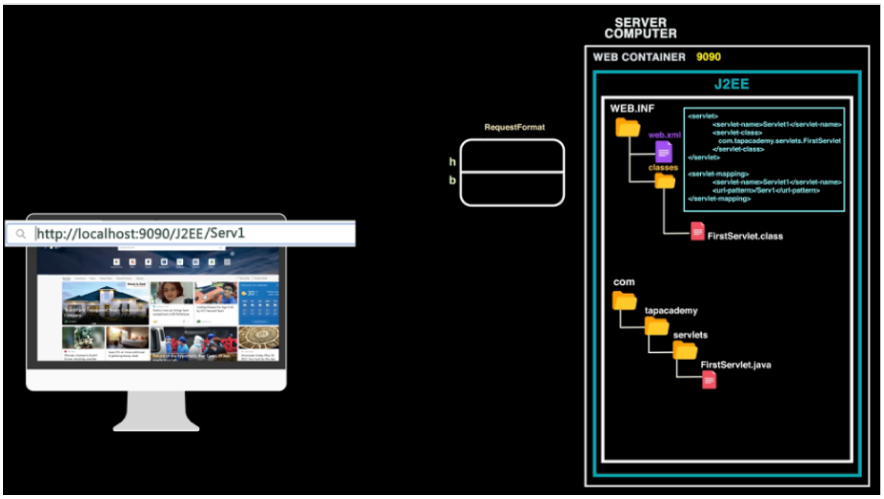


Life cycle of a Servlet is managed by the Web Container
Let us now try to understand the life cycle of a servlet.
We will override init( ), doGet( ) and destroy( ) methods in this example-

```
package com.tap.servlets;

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


public class FirstServlet extends HttpServlet {

	@Override
	public void init() throws ServletException {
		System.out.println("init() method called");
	}

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		System.out.println("doGet() service method called");
	}

	@Override
```

```
	public void destroy() {
		System.out.println("destroy() method called");
	}

}
```

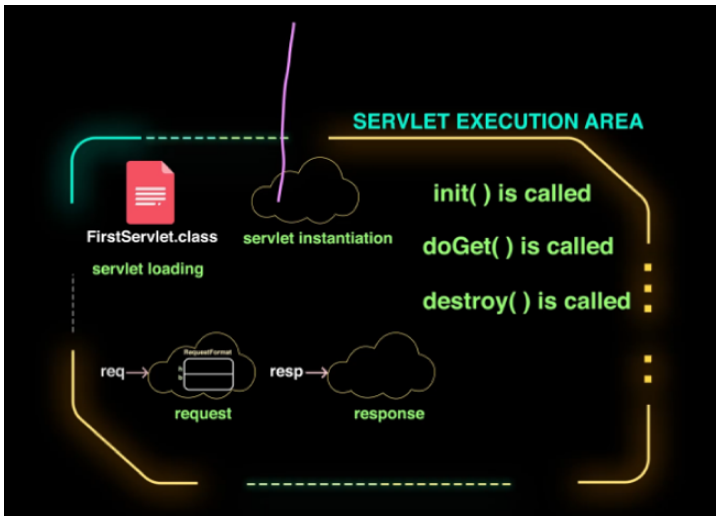
So whenever a request is sent from the client to the server, a **RequestFormat** is generated and it is what is shared to the server. Now the deployment descriptor will accept the URL and map it with Servlet which is mapped in the url-pattern.
But the **FirstServlet.java** is a high level java file and we know that high level code cannot be executed and it has to be converted into low level. So automatically web.xml will ensure that the java file is compiled by the compiler and we know that whenever a java program is compiled, a **.class file** is generated and this class file will be stored in a hidden folder called **classes** in the **WEB-INF folder.**



Now that the class file is generated, let us now see how the execution takes place.
Step 1: Servlet Loading:
The class file will be loaded into the Servlet Execution Area. The class will be loaded with the help of Class.forName( ).
Step 2: Servlet Instantiation:
Now the web container will create an object of the class
Step 3: Servlet Initialization:
As soon as the servlet instantiation is completed, servlet initialization i.e., init( ) will be called
Step 4: Service method will be called:
In our case, doGet( ) will be called
Step 5: Request and Response object will be created:
Request object is used to collect the information from the client and Response object is used to send back the response back to the client.
So the RequestFormat which was sent from the client will be present inside the Request object. The references **req** and **resp** will be sent to the **doGet( )** service method.
A separate thread will be created in the class object and we call the service methods “n” number of times.

Step 6: destroy( ) will be called:

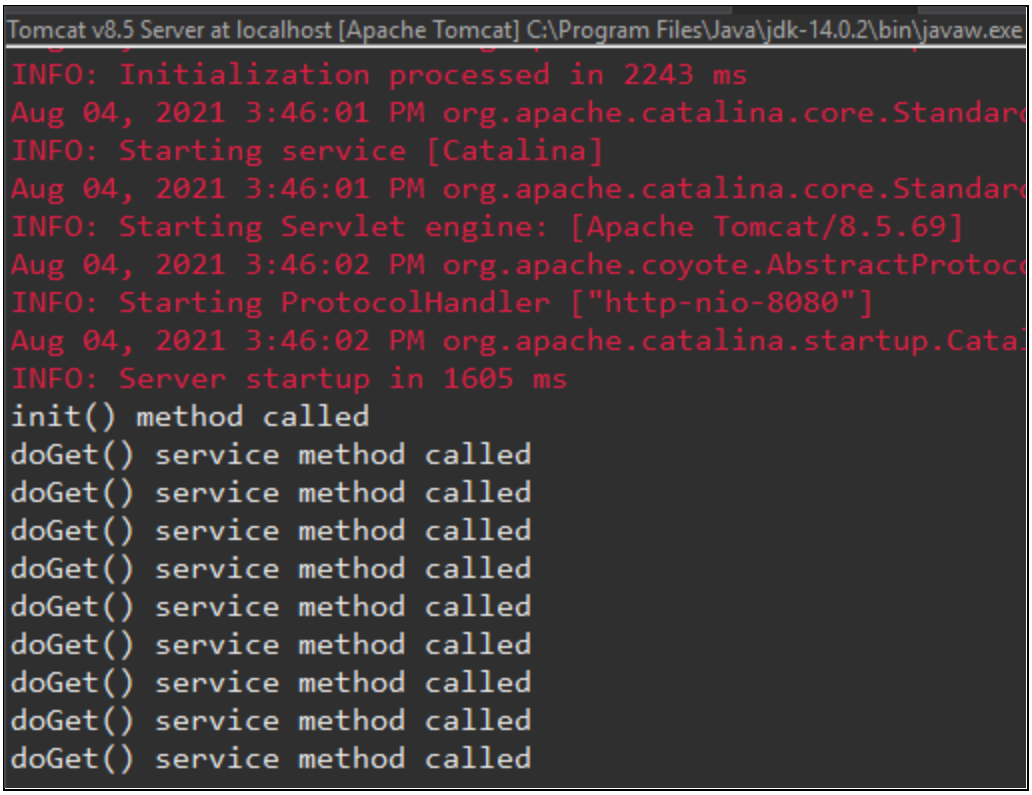
Once the execution of doGet( ) is done, then the Request, Response and Server objects should be deleted i.e., deallocated. For that destroy( ) will be invoked and it deallocates the objects.



And when we the execute the program, then we will get the output on the console and not on the browser as we have used System.out.println( ) in our code

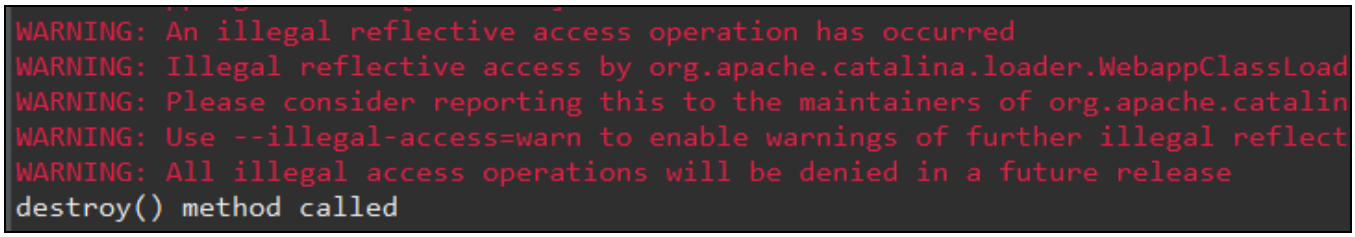
```
Tomcat v8.5 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe
Aug 04, 2021 3:46:01 PM org.apache.catalina.core.Standard
INFO: Starting Servlet engine: [Apache Tomcat/8.5.69]
Aug 04, 2021 3:46:02 PM org.apache.coyote.AbstractProtoco
INFO: Starting ProtocolHandler ["http-nio-8080"]
Aug 04, 2021 3:46:02 PM org.apache.catalina.startup.Catal
INFO: Server startup in 1605 ms
init() method called
doGet() service method called
```

We can execute the doGet( ) many times since the Server object is already created



So when will the destroy( ) method be called?

The destroy( ) method will be called, when the server stops accepting the input from the client and for that we have to stop the server and before the server is stopped, destroy( ) will be called.



So init( ) and destroy( ) will be called only once but the service method doGet( ) can be called as many times as required.