Servlet Lifecycle

The lifecycle of a Java Servlet is a series of steps that a servlet goes through from its initialization to handling client requests and finally being destroyed. I'll explain the servlet lifecycle in detail and provide two code examples to illustrate it.

The servlet lifecycle consists of the following stages:

1. **Initialization (init):** This stage is where the servlet is created and initialized. The `init` method is called by the servlet container when the servlet is first loaded into memory. It is typically used for one-time initialization tasks.

2. **Request Handling (service):** Once the servlet is initialized, it can handle client requests. The `service` method is called for each client request, and it is responsible for processing the request and generating a response. The `service` method typically delegates the actual request handling to other methods like `doGet`, `doPost`, etc., based on the HTTP method used in the request.

3. **Request Handling Methods (doGet, doPost, etc.):** These methods are called by the `service` method based on the HTTP method of the request. For example, `doGet` is called for HTTP GET requests, and `doPost` is called for HTTP POST requests. You can override these methods to implement the specific logic for handling those request types.

4. **Destruction (destroy):** The `destroy` method is called by the servlet container when it decides to remove the servlet from memory. This typically happens when the application is being shut down or undeployed. It's used to perform cleanup tasks, such as releasing resources.

Now, let's look at two code examples to illustrate the servlet lifecycle. We'll create a simple servlet that echoes back the client's request with some information about the servlet's lifecycle events.

Example 1: Servlet without web.xml (Annotation-based)

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

@WebServlet("/lifecycleServlet")
public class LifecycleServlet extends HttpServlet {
```

```
    public void init() {
        System.out.println("Servlet initialization");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException {
        System.out.println("Handling GET request");
        response.getWriter().write("Hello, this is a GET request handled by the servlet!");
    }

    public void destroy() {
        System.out.println("Servlet destruction");
    }
}
```

Example 2: Servlet with web.xml configuration

```xml
<!-- web.xml -->
<web-app>
    <servlet>
        <servlet-name>LifecycleServlet</servlet-name>
        <servlet-class>com.example.LifecycleServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>LifecycleServlet</servlet-name>
        <url-pattern>/lifecycleServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

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

public class LifecycleServlet extends HttpServlet {

    public void init() {
        System.out.println("Servlet initialization");
    }
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException {
        System.out.println("Handling GET request");
        response.getWriter().write("Hello, this is a GET request handled by the servlet!");
    }

    public void destroy() {
        System.out.println("Servlet destruction");
    }
}
```

In both examples, the `init`, `doGet`, and `destroy` methods are overridden to print messages at key points in the servlet's lifecycle. The servlet is mapped to the URL "/lifecycleServlet" and responds to HTTP GET requests. When you access this URL, you will see the lifecycle events being printed in the console.