Name: Jayvirsinh Raj
B00907110

Report on WebServer Java Code

Main Method:

1. It sets the port number to 8080 and creates a `ServerSocket` on this port to listen for incoming connections.
2. It enters an infinite loop to continuously accept incoming connections.
3. Upon accepting a connection, it prints the client's IP address.
4. It retrieves input and output streams from the socket for communication with the client.

Processing HTTP Requests:

1. It reads the HTTP request message sent by the client.
2. It parses the request line to extract the filename requested by the client.
3. It checks if the requested file exists.
4. Based on the file's existence, it constructs a response message:
5. If the file exists, it sets status line to "200 OK" and sends the file's content along with appropriate content type.
6. If the file does not exist, it sets status line to "404 Not Found" and sends a simple HTML error message.

Content-Type Determination:

1. The `contentType` method determines the content type based on the file extension.

Helper Methods:

1) `sendBytes`: Sends the content of a file as bytes to the client.
2) `contentType`: Determines the content type based on the file extension.

Closing Connections:

1. After sending the response, it closes the streams and the connection socket.

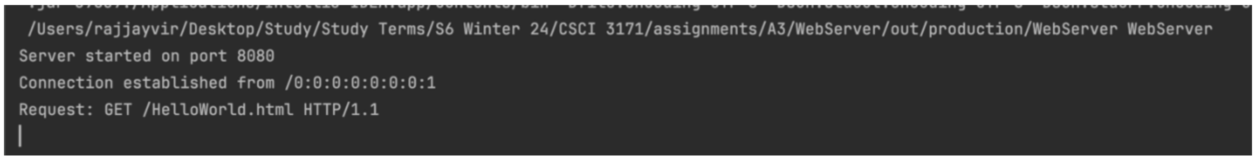
Testing Method:

1. Compile the `WebServer.java` file to generate the bytecode.
2. Run the compiled program using the Java interpreter.
3. Access the server from a web browser using `localhost:8080/HelloWorld.html`.
4. Test various scenarios:
   - Accessing existing HTML, image, and text files.
   - Accessing non-existent files.
   - Accessing files with different extensions.
5. Observe the behavior of the server, ensuring it responds correctly to different types of requests and files.

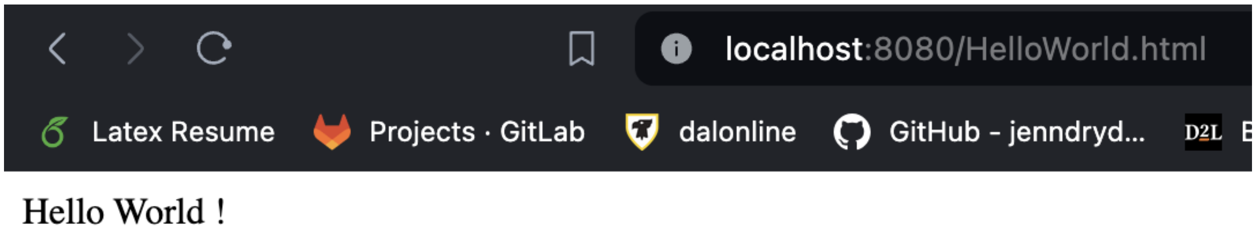
Expected Results:

I. Successful retrieval of existing files should result in the browser displaying the content correctly.
II. Requests for non-existent files should return a "404 Not Found" error.
III. The server should correctly determine the content type based on the file extension and send appropriate headers.
IV. The server should handle multiple concurrent connections gracefully.

When the file is found:



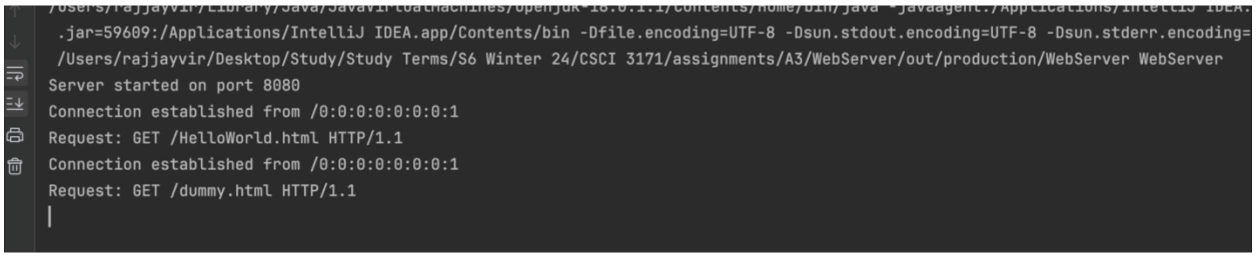
```
/Users/rajjayvir/Desktop/Study/Study Terms/S6 Winter 24/CSCI 3171/assignments/A3/WebServer/out/production/WebServer WebServer
Server started on port 8080
Connection established from /0:0:0:0:0:0:0:1
Request: GET /HelloWorld.html HTTP/1.1
|
```



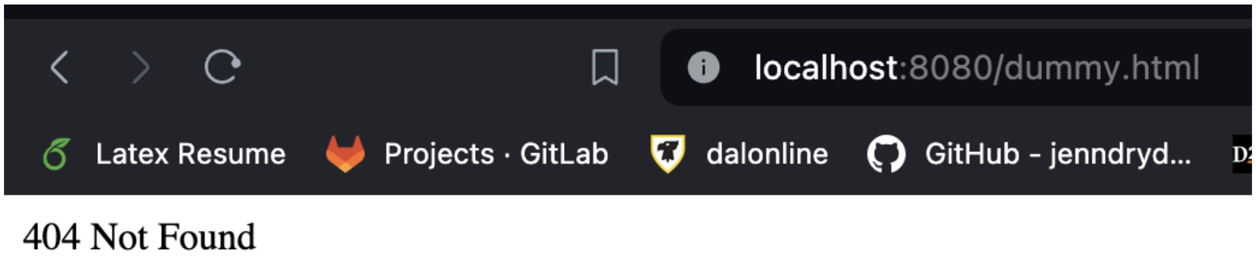
localhost:8080/HelloWorld.html

Hello World !

When the file is not found:



```
/Users/rajjayvir/Library/Java/JavaVirtualMachines/openjdk-18.0.1.1/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.
.jar=59609:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=
/Users/rajjayvir/Desktop/Study/Study Terms/S6 Winter 24/CSCI 3171/assignments/A3/WebServer/out/production/WebServer WebServer
Server started on port 8080
Connection established from /0:0:0:0:0:0:0:1
Request: GET /HelloWorld.html HTTP/1.1
Connection established from /0:0:0:0:0:0:0:1
Request: GET /dummy.html HTTP/1.1
|
```



localhost:8080/dummy.html

404 Not Found

Report on JavaClient Java Code

Code Explanation:

Main Method:

I. It checks if all required command-line arguments (server address, port, path) are provided. If not, it displays a usage message and exits.
II. It extracts the server address, port number, and path from the command-line arguments.
III. Inside a try-catch block, it establishes a TCP socket connection to the specified server and port.
IV. It obtains output stream to send data to the server and creates a PrintWriter to write HTTP GET request.
V. Sends an HTTP GET request to the server with the specified path and HTTP version.
VI. Creates an input stream to read data from the server's response.
VII. Reads and displays the server response line by line.
VIII. Finally, closes the streams and socket in a `finally` block to ensure proper cleanup, even in case of exceptions.

Testing Environment:

I. A separate machine or virtual machine acting as the server, **running the WebServer** program provided earlier.
II. Command-line access to run the JavaClient program.

Testing Method:

1. Compile the `JavaClient.java` file to generate the bytecode.
2. Start the server by running the WebServer program on a machine accessible from the client.
3. Run the compiled `JavaClient` program from the command line with appropriate arguments:

(See image)

```
java JavaClient <server_address> <port> <path>
```

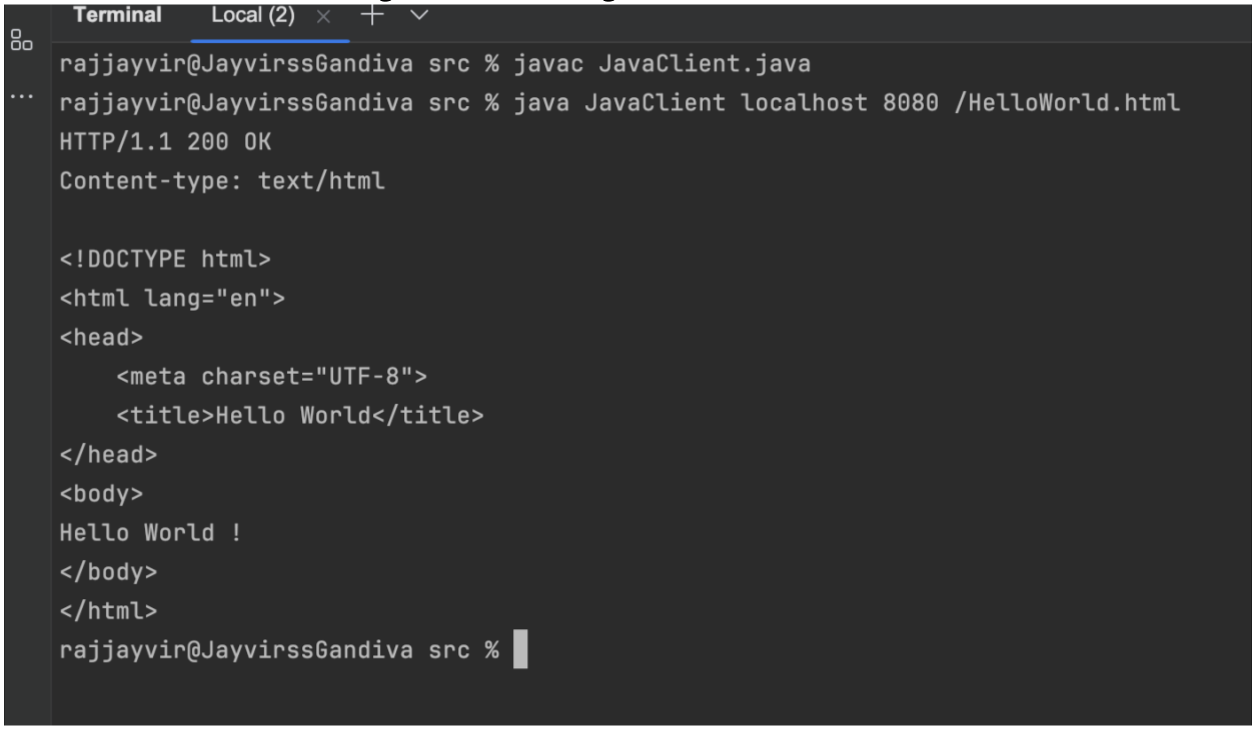
4. Observe the output to ensure that the client successfully connects to the server, sends the HTTP GET request, and displays the server's response.
5. Test with various combinations of server addresses, ports, and paths to ensure robustness.

Expected Results:

I. The client should establish a connection to the server without errors.
II. It should send the HTTP GET request correctly.
III. The server should respond with the requested content, which the client should display.
IV. The client should handle errors gracefully, displaying appropriate messages if the server is unreachable or if there are issues with the connection.

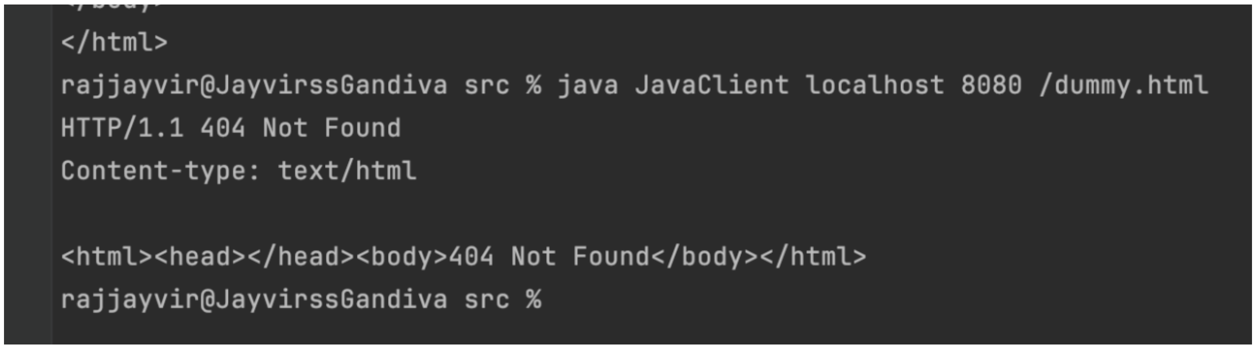
Conclusion:
The `JavaClient` class provides a simple HTTP client implementation in Java. It connects to a specified server over TCP, sends an HTTP GET request for a given path, and displays the server's response. Proper error handling ensures that the client behaves robustly in various scenarios. Testing in a suitable environment ensures the client's functionality and reliability.

This is achieved with running server on background.



```
rajjayvir@JayvirssGandiva src % javac JavaClient.java
rajjayvir@JayvirssGandiva src % java JavaClient localhost 8080 /HelloWorld.html
HTTP/1.1 200 OK
Content-type: text/html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello World</title>
</head>
<body>
Hello World !
</body>
</html>
rajjayvir@JayvirssGandiva src %
```



```
</html>
rajjayvir@JayvirssGandiva src % java JavaClient localhost 8080 /dummy.html
HTTP/1.1 404 Not Found
Content-type: text/html

<html><head></head><body>404 Not Found</body></html>
rajjayvir@JayvirssGandiva src %
```