**Simple Java Web Server**

**1 Overview**

The Simple Java Web Server is a minimalistic implementation of a web server capable of handling basic HTTP/1.1 requests. The server is designed to read and respond to requests for files from a web browser.

**2 Project Structure**

The project consists of the following main files:

1. **WebServer.java**: The main class that initializes and runs the server.
2. **StopWatchLabel.java**: A custom component for a simple stop-watch.
3. **TestStopWatchRunner.java**: An applet to test the StopWatchLabel component.
4. **ReadRequest.java**: A program to read and display HTTP requests for testing purposes.

**3 Running the Server**

**WebServer.java**: This class contains the **main** method and serves as the entry point for the web server. It initializes a **ServerSocket** to listen for incoming connections and continuously handles connections in a loop.

To run the server, execute the **main** method in this class.

**4 Handling Connections**

**handleConnection(Socket connection)**: This method is responsible for handling communication with a connected client. It reads the HTTP request, validates it, and responds accordingly. If an error occurs during the handling process, an appropriate error response is sent.

It uses the **BufferedReader** and **PrintWriter** classes to read from and write to the client's socket.

**5 Sending Files**

**sendFile(File file, PrintWriter out)**: This method sends a requested file to the client as an HTTP response. It first sends the necessary HTTP headers, such as status line, content length, and content type. Then, it sends the actual file content.

The file is read using a **BufferedInputStream**, and its content is written to the client's socket.

**6 Error Handling**

**sendErrorResponse(int errorCode, PrintWriter out)**: This method sends an error response to the client based on the provided error code. It sends appropriate HTTP headers and an HTML body explaining the error.

Common error codes include 400 (Bad Request), 404 (Not Found), and 500 (Internal Server Error).

**7 MIME Type Determination**

**getMimeType(String fileName)**: This method determines the MIME type of a file based on its extension. It is used to set the **Content-Type** header in the HTTP response.

The MIME type is determined based on the file extension, and a default value of "application/octet-stream" is used for unknown extensions.

**8 Running Additional Components**

**StopWatchLabel and TestStopWatchRunner**: These files contain a simple stop-watch component and an applet to test it. They are not directly related to the web server functionality but can be used for testing.

**9 Configuration**

**LISTENING\_PORT**: The port on which the server listens for incoming connections. The default value is set to 8080, but you can choose any available port.

Update this value to use a different port.

private static final int LISTENING\_PORT = 8080;  
  
  
**10 Program**

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

public class WebServer {

    private static final int LISTENING\_PORT = 8080; // You can choose any available port

    public static void main(String[] args) {

        ServerSocket serverSocket;

        try {

            serverSocket = new ServerSocket(LISTENING\_PORT);

        } catch (IOException e) {

            System.out.println("Failed to create listening socket.");

            return;

        }

        System.out.println("Listening on port " + LISTENING\_PORT);

        try {

            while (true) {

                Socket connection = serverSocket.accept();

                System.out.println("\nConnection from " + connection.getRemoteSocketAddress());

                handleConnection(connection);

            }

        } catch (IOException e) {

            System.out.println("Server socket shut down unexpectedly!");

            System.out.println("Error: " + e);

            System.out.println("Exiting.");

        }

    }

    private static void handleConnection(Socket connection) {

        try {

            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));

            PrintWriter out = new PrintWriter(connection.getOutputStream());

            // Read the request line

            String requestLine = in.readLine();

            if (requestLine == null) {

                // No data received; close the connection

                connection.close();

                return;

            }

            // Split the request line into tokens

            String[] tokens = requestLine.split("\\s+");

            // Check if the request is a valid GET request

            if (tokens.length != 3 || !tokens[0].equals("GET")) {

                sendErrorResponse(400, out);

                connection.close();

                return;

            }

            // Extract the path to the requested file

            String pathToFile = tokens[1];

            // Resolve the requested file

            File file = new File("src/web/" + pathToFile);

            // Check if the file exists and is readable

            if (!file.exists() || !file.isFile() || !file.canRead()) {

                sendErrorResponse(404, out);

                connection.close();

                return;

            }

            // Send the file as a response

            sendFile(file, out);

        } catch (IOException e) {

            System.out.println("Error while communicating with client: " + e);

        } finally {

            try {

                connection.close();

            } catch (IOException e) {

                // Ignore

            }

            System.out.println("Connection closed.");

        }

    }

    private static void sendFile(File file, PrintWriter out) throws IOException {

        try (BufferedInputStream fileInput = new BufferedInputStream(new FileInputStream(file))) {

            // Send the HTTP headers

            out.println("HTTP/1.1 200 OK");

            out.println("Connection: close");

            out.println("Content-Length: " + file.length());

            out.println("Content-Type: " + getMimeType(file.getName()));

            out.println(); // Empty line to indicate the end of headers

            out.flush();

            // Send the file content

            byte[] buffer = new byte[1024];

            int bytesRead;

            while ((bytesRead = fileInput.read(buffer)) != -1) {

                out.write(new String(buffer, 0, bytesRead));

            }

            out.flush();

        } catch (FileNotFoundException e) {

            sendErrorResponse(404, out);

        }

    }

    private static void sendErrorResponse(int errorCode, PrintWriter out) {

        String statusMessage;

        String errorMessage;

        switch (errorCode) {

            case 400:

                statusMessage = "Bad Request";

                errorMessage = "The request syntax is invalid.";

                break;

            case 404:

                statusMessage = "Not Found";

                errorMessage = "The requested resource could not be found on this server.";

                break;

            case 500:

                statusMessage = "Internal Server Error";

                errorMessage = "The server encountered an unexpected condition that prevented it from fulfilling the request.";

                break;

            default:

                statusMessage = "Internal Server Error";

                errorMessage = "An unknown error occurred.";

        }

        // Send the HTTP headers for the error response

        out.println("HTTP/1.1 " + errorCode + " " + statusMessage);

        out.println("Connection: close");

        out.println("Content-Type: text/html");

        out.println(); // Empty line to indicate the end of headers

        // Send the HTML content for the error message

        out.println("<html><head><title>Error</title></head><body>");

        out.println("<h2>Error: " + errorCode + " " + statusMessage + "</h2>");

        out.println("<p>" + errorMessage + "</p>");

        out.println("</body></html>");

        out.flush();

    }

    // ... (other methods)

    private static String getMimeType(String fileName) {

        int pos = fileName.lastIndexOf('.');

        if (pos < 0) {

            // No file extension found, default to binary data

            return "application/octet-stream";

        }

        String ext = fileName.substring(pos + 1).toLowerCase();

        switch (ext) {

            case "txt":

                return "text/plain";

            case "html":

            case "htm":

                return "text/html";

            case "css":

                return "text/css";

            case "js":

                return "text/javascript";

            case "java":

                return "text/x-java";

            case "jpeg":

            case "jpg":

                return "image/jpeg";

            case "png":

                return "image/png";

            case "gif":

                return "image/gif";

            case "ico":

                return "image/x-icon";

            case "class":

                return "application/java-vm";

            case "jar":

                return "application/java-archive";

            case "zip":

                return "application/zip";

            case "xml":

                return "application/xml";

            case "xhtml":

                return "application/xhtml+xml";

            default:

                // Unknown file extension, default to binary data

                return "application/octet-stream";

        }}

}

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Scanner;

/\*\*

 \* The main() program in this class is designed to read requests from

 \* a Web browser and display the requests on standard output.  The

 \* program sets up a listener on port 50505.  It can be contacted

 \* by a Web browser running on the same machine using a URL of the

 \* form  http://localhost:505050/path/to/resource.html  This method

 \* does not return any data to the web browser.  It simply reads the

 \* request, writes it to standard output, and then closes the connection.

 \* The program continues to run, and the server continues to listen

 \* for new connections, until the program is terminated (by clicking the

 \* red "stop" square in Eclipse or by Control-C on the command line).

 \*/

public class ReadRequest {

    /\*\*

     \* The server listens on this port.  Note that the port number must

     \* be greater than 1024 and lest than 65535.

     \*/

    private final static int LISTENING\_PORT = 50505;

    /\*\*

     \* Main program opens a server socket and listens for connection

     \* requests.  It calls the handleConnection() method to respond

     \* to connection requests.  The program runs in an infinite loop,

     \* unless an error occurs.

     \* @param args ignored

     \*/

    public static void main(String[] args) {

        ServerSocket serverSocket;

        try {

            serverSocket = new ServerSocket(LISTENING\_PORT);

        }

        catch (Exception e) {

            System.out.println("Failed to create listening socket.");

            return;

        }

        System.out.println("Listening on port " + LISTENING\_PORT);

        try {

            while (true) {

                Socket connection = serverSocket.accept();

                System.out.println("\nConnection from "

                        + connection.getRemoteSocketAddress());

                handleConnection(connection);

            }

        }

        catch (Exception e) {

            System.out.println("Server socket shut down unexpectedly!");

            System.out.println("Error: " + e);

            System.out.println("Exiting.");

        }

    }

    /\*\*

     \* Handle commuincation with one client connection.  This method reads

     \* lines of text from the client and prints them to standard output.

     \* It continues to read until the client closes the connection or

     \* until an error occurs or until a blank line is read.  In a connection

     \* from a Web browser, the first blank line marks the end of the request.

     \* This method can run indefinitely,  waiting for the client to send a

     \* blank line.

     \* NOTE:  This method does not throw any exceptions.  Exceptions are

     \* caught and handled in the method, so that they will not shut down

     \* the server.

     \* @param connection the connected socket that will be used to

     \*    communicate with the client.

     \*/

    private static void handleConnection(Socket connection) {

        try {

            Scanner in = new Scanner(connection.getInputStream());

            while (true) {

                if ( ! in.hasNextLine() )

                    break;

                String line = in.nextLine();

                if (line.trim().length() == 0)

                    break;

                System.out.println("   " + line);

            }

        }

        catch (Exception e) {

            System.out.println("Error while communicating with client: " + e);

        }

        finally {  // make SURE connection is closed before returning!

            try {

                connection.close();

            }

            catch (Exception e) {

            }

            System.out.println("Connection closed.");

        }

    }

}

import java.awt.event.\*;

import javax.swing.\*;

/\*\*

 \* A custom component that acts as a simple stop-watch.  When the user clicks

 \* on it, this componet starts timing.  When the user clicks again,

 \* it displays the time between the two clicks.  Clicking a third time

 \* starts another timer, etc.  While it is timing, the label just

 \* displays the message "Timing....".

 \*/

public class StopWatchLabel extends JLabel implements MouseListener {

   private long startTime;   // Start time of timer.

                             //   (Time is measured in milliseconds.)

   private boolean running;  // True when the timer is running.

   /\*\*

    \* Constructor sets initial text on the label to

    \* "Click to start timer." and sets up a mouse listener

    \* so the label can respond to clicks.

    \*/

   public StopWatchLabel() {

      super("  Click to start timer.  ", JLabel.CENTER);

      addMouseListener(this);

   }

   /\*\*

    \* Tells whether the timer is currently running.

    \*/

   public boolean isRunning() {

      return running;

   }

   /\*\*

    \* React when the user presses the mouse by starting

    \* or stopping the timer and changing the text that

    \* is shown on the label.

    \*/

   public void mousePressed(MouseEvent evt) {

      if (running == false) {

            // Record the time and start the timer.

         running = true;

         startTime = evt.getWhen();  // Time when mouse was clicked.

         setText("Timing....");

      }

      else {

            // Stop the timer.  Compute the elapsed time since the

            // timer was started and display it.

         running = false;

         long endTime = evt.getWhen();

         double seconds = (endTime - startTime) / 1000.0;

         setText("Time: " + seconds + " sec.");

      }

   }

   public void mouseReleased(MouseEvent evt) { }

   public void mouseClicked(MouseEvent evt) { }

   public void mouseEntered(MouseEvent evt) { }

   public void mouseExited(MouseEvent evt) { }

}

 /\*

       A trivial applet that tests the StopWatchTimer component.

       The applet just creates and shows a StopWatchTimer.

    \*/

    import java.awt.\*;

    import java.applet.\*;

    public class TestStopWatchRunner extends Applet {

       public void init() {

          StopWatchRunner watch = new StopWatchRunner();

          watch.setFont( new Font("SansSerif", Font.BOLD, 24) );

          watch.setBackground(Color.white);

          watch.setForeground( new Color(180,0,0) );

          setBackground(Color.white);

          setLayout(new BorderLayout() );

          add(watch, BorderLayout.CENTER);

       }

    }

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

  </head>

  <body>

    <p>

      Lorem ipsum dolor sit amet consectetur, adipisicing elit. Voluptates amet

      nulla excepturi quod recusandae temporibus, corrupti corporis quas facere

      sed magnam veniam nisi magni possimus, exercitationem hic! Vitae, aliquid

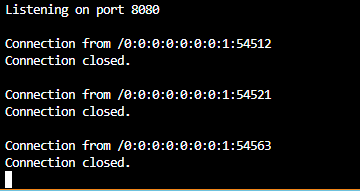
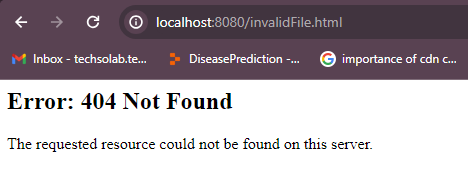
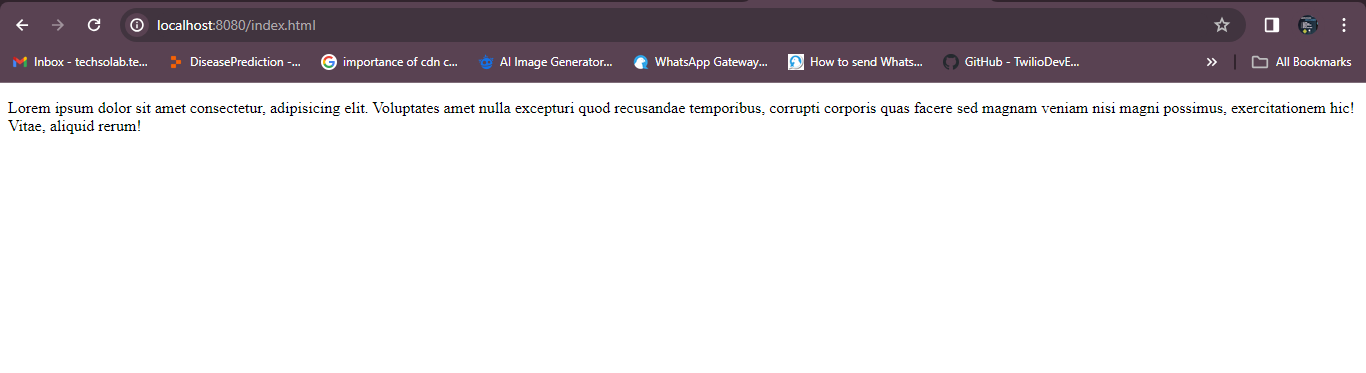
      rerum!

    </p>

  </body>

</html>

**11 Output**



**The End**