

'My own HTTP' server sample

Let's create an empty Java project, Then create files called *404.html* and an *index.html* with the following content. Let's note that the error handling is redundant, i.e. the handling of *errors 404* and *501* is written twice.

Content of *404.html*:

```
<html>
  <body>
    <h1>File not found error</h1>
  </body>
</html>
```

Content of *index.html*:

```
<html>
<body>
<h1>My HTTP server works</h1>
<a href="/dead_link.html">Link not working</a>
</body>
</html>
```

Content of *JavaHTTPServer.java*:

```
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;
import java.util.StringTokenizer;

public class JavaHTTPServer implements Runnable {

    static final File WEB_ROOT = new File(".");
    static final String DEFAULT_FILE = "index.html";
    static final String FILE_NOT_FOUND = "404.html";
```

```
private Socket connect;

public JavaHTTPServer(Socket c) {
    connect = c;
}

public static void main(String[] args) {
    try {
        ServerSocket serverConnect = new ServerSocket(8080);
        System.out.println("Server started.\nListening for connections
on port : 8080 ... \n");

        while (true) {
            JavaHTTPServer myServer = new
JavaHTTPServer(serverConnect.accept());

            Thread thread = new Thread(myServer);
            thread.start();
        }
    } catch (IOException e) {
        System.err.println("Server Connection error : " +
e.getMessage());
    }
}

@Override
public void run() {
    BufferedReader in = null;
    PrintWriter out = null;
    BufferedOutputStream dataOut = null;
    String fileRequested = null;

    try {
        in = new BufferedReader(new
InputStreamReader(connect.getInputStream()));
        out = new PrintWriter(connect.getOutputStream());
        dataOut = new BufferedOutputStream(connect.getOutputStream());

        String input = in.readLine();

        StringTokenizer parse = new StringTokenizer(input);
        String method = parse.nextToken().toUpperCase(); // we get the
HTTP method of the client
        // we get file requested
        fileRequested = parse.nextToken().toLowerCase();

        // we support only GET and HEAD methods, we check
        if (method.equals("GET") || method.equals("HEAD")) {
            // GET or HEAD method
            if (fileRequested.endsWith("/")) {
                fileRequested += DEFAULT_FILE;
            }
        }
    }
}
```

```
    }

    File file = new File(WEB_ROOT, fileRequested);
    int fileLength = (int) file.length();
    String content = getContentType(fileRequested);

    if (method.equals("GET")) { // GET method so we return
content
        byte[] fileData = readFileData(file, fileLength);

        // send HTTP Headers
        out.println("HTTP/1.1 200 OK");
        out.println("Server: Java HTTP Server v1.0");
        out.println("Date: " + new Date());
        out.println("Content-type: " + content);
        out.println("Content-length: " + fileLength);
        out.println(); // blank line between headers and
content, very important
        out.flush(); // flush character output stream buffer

        dataOut.write(fileData, 0, fileLength);
        dataOut.flush();
    }
}

} catch (FileNotFoundException fnfe) {
    try {
        fileNotFound(out, dataOut, fileRequested);
    } catch (IOException ioe) {
        System.err.println("Error with file not found exception : "
+ ioe.getMessage());
    }

} catch (IOException ioe) {
    System.err.println("Server error : " + ioe);
} finally {
    try {
        in.close();
        out.close();
        dataOut.close();
        connect.close(); // we close socket connection
    } catch (Exception e) {
        System.err.println("Error closing stream : " +
e.getMessage());
    }
}

}

private byte[] readFileData(File file, int fileLength) throws
IOException {
```

```
        FileInputStream fileIn = null;
        byte[] fileData = new byte[fileLength];

        try {
            fileIn = new FileInputStream(file);
            fileIn.read(fileData);
        } finally {
            if (fileIn != null)
                fileIn.close();
        }

        return fileData;
    }

    // return supported MIME Types
    private String getContentType(String fileRequested) {
        if (fileRequested.endsWith(".htm") ||
fileRequested.endsWith(".html"))
            return "text/html";
        else
            return "text/plain";
    }

    private void fileNotFound(PrintWriter out, OutputStream dataOut, String
fileRequested) throws IOException {
        File file = new File(WEB_ROOT, FILE_NOT_FOUND);
        int fileLength = (int) file.length();
        String content = "text/html";
        byte[] fileData = readFileData(file, fileLength);

        out.println("HTTP/1.1 404 File Not Found");
        out.println("Server: Java HTTP Server v1.0");
        out.println("Date: " + new Date());
        out.println("Content-type: " + content);
        out.println("Content-length: " + fileLength);
        out.println(); // blank line between headers and content, very
important
        out.flush(); // flush character output stream buffer

        dataOut.write(fileData, 0, fileLength);
        dataOut.flush();
    }
}
```

Task 1.: create a generic error handling function to return errors to the client side.

Task 2.: refactor the code and eliminate duplicate parts. Let's create a separate function to return a general HTTP response:

```
out.println("HTTP/1.1 200 OK");
out.println("Server: Java HTTP Server v1.0");
out.println("Date: " + new Date());
out.println("Content-type: " + content);
out.println("Content-length: " + fileLength);
out.println(); // blank line between headers and content, very important
out.flush(); // flush character output stream buffer
```

Task 3.: modify the source code so that it can also return images. To do this, first add `` to `index.html` and copy an arbitrary image next to the `html-s`.

From:

<https://edu.iit.uni-miskolc.hu/> - **Institute of Information Science - University of Miskolc**

Permanent link:

https://edu.iit.uni-miskolc.hu/tanszek:oktatas:iss_t:http_server

Last update: **2026/03/09 09:57**

