Saját HTTP szerver mintapélda

Készítsünk egy üres Java projektet, hozzunk létre egy 404.html és egy index.html tartalmú fájlt az alábbi tartalommal. Majd figyeljük meg hogy a hibakezelés redundáns, azaz kétszer van megírva 404 és 501-es hiba kezelése. **Feladat**: készítsen egy általános hibakezelő függvényt a hibák kliens oldalra visszadására.

404.html tartalma:

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

index.html tartalma:

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

JavaHTTPServer.java tartalma:

```
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();
    }

}
```

Feladat: refaktoráljuk a kódot és szüntessük meg a többszörözött részeket. Készítsünk egy külön függvényt a HTTP válasz általános visszadásához:

```
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
```

Feladat: módosítsuk a forráskódot, hogy képeket is vissza tudjon adni. Ehhez először <img src="…> kiegészítést adjunk hozzá a index.html-nek és másoljunk egy tetszőleges képet a html-ek mellé.