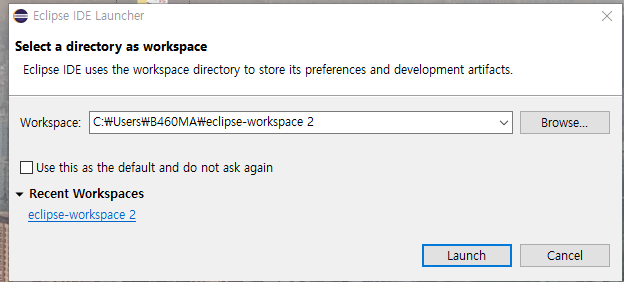
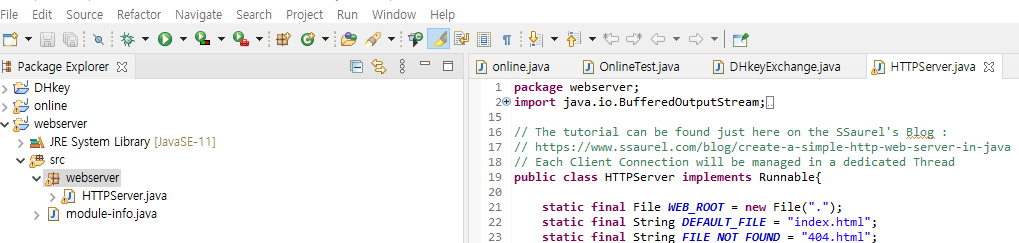
* HTTP 웹서버를 PC에 구축 (웹 서버로 사용할 PC는 공용 IP 주소를 사용하거나, 또는 브라우져로 사용할 핸드폰 (또는 노트북, 다른 PC)과 동일 서브넷에 있는 PC를 사용하여야 함)

1. 이클립스를 사용하여 워크스페이스 (작업 공간)를 Launch 하시오.



1. 아래 화면을 참고하여 Package webserver; 코드로 시작하는 HTTPServer.java를 만드시오.



1. 아래 코드를 참고하여 웹서버 JAVA 소스 구현 (아래 소스에서 ‘팀이름’은 실재 팀 이름으로 바꿀 것)

|  |
| --- |
| System.out.println("팀이름: Server started.\nListening for connections on port : " + PORT + " ...\n"); |
| package webserver;  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;  // The tutorial can be found just here on the SSaurel's Blog :  // https://www.ssaurel.com/blog/create-a-simple-http-web-server-in-java  // Each Client Connection will be managed in a dedicated Thread  public class HTTPServer implements Runnable{    static final File WEB\_ROOT = new File(".");  static final String DEFAULT\_FILE = "index.html";  static final String FILE\_NOT\_FOUND = "404.html";  static final String METHOD\_NOT\_SUPPORTED = "not\_supported.html";  // port to listen connection  static final int PORT = 8080;    // verbose mode  static final boolean verbose = true;    // Client Connection via Socket Class  private Socket connect;    public HTTPServer(Socket c) {  connect = c;  }    public static void main(String[] args) {  try {  ServerSocket serverConnect = new ServerSocket(PORT);  System.out.println("팀이름: Server started.\nListening for connections on port : " + PORT + " ...\n");    // we listen until user halts server execution  while (true) {  HTTPServer myServer = new HTTPServer(serverConnect.accept());    if (verbose) {  System.out.println("Connecton opened. (" + new Date() + ")");  }    // create dedicated thread to manage the client connection  Thread thread = new Thread(myServer);  thread.start();  }    } catch (IOException e) {  System.err.println("Server Connection error : " + e.getMessage());  }  }  @Override  public void run() {  // we manage our particular client connection  BufferedReader in = null; PrintWriter out = null; BufferedOutputStream dataOut = null;  String fileRequested = null;    try {  // we read characters from the client via input stream on the socket  in = new BufferedReader(new InputStreamReader(connect.getInputStream()));  // we get character output stream to client (for headers)  out = new PrintWriter(connect.getOutputStream());  // get binary output stream to client (for requested data)  dataOut = new BufferedOutputStream(connect.getOutputStream());    // get first line of the request from the client  String input = in.readLine();  // we parse the request with a string tokenizer  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")) {  if (verbose) {  System.out.println("501 Not Implemented : " + method + " method.");  }    // we return the not supported file to the client  File file = new File(WEB\_ROOT, METHOD\_NOT\_SUPPORTED);  int fileLength = (int) file.length();  String contentMimeType = "text/html";  //read content to return to client  byte[] fileData = readFileData(file, fileLength);    // we send HTTP Headers with data to client  out.println("HTTP/1.1 501 Not Implemented");  out.println("Server: Java HTTP Server from SSaurel : 1.0");  out.println("Date: " + new Date());  out.println("Content-type: " + contentMimeType);  out.println("Content-length: " + fileLength);  out.println(); // blank line between headers and content, very important !  out.flush(); // flush character output stream buffer  // file  dataOut.write(fileData, 0, fileLength);  dataOut.flush();    } else {  // 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 from SSaurel : 1.0");  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();  }    if (verbose) {  System.out.println("File " + fileRequested + " of type " + content + " returned");  }    }    } 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());  }    if (verbose) {  System.out.println("Connection closed.\n");  }  }      }    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 if (fileRequested.endsWith(".jpg") || fileRequested.endsWith(".jpeg"))  return "image/jpg";  else if (fileRequested.endsWith(".mpeg"))  return "video/mpeg";  else if (fileRequested.endsWith(".mp4"))  return "video/mp4";  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 from SSaurel : 1.0");  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();    if (verbose) {  System.out.println("File " + fileRequested + " not found");  }  }    } |

<아래는 위와 같은 내용임, 라인별로 코드 표시한 내용>

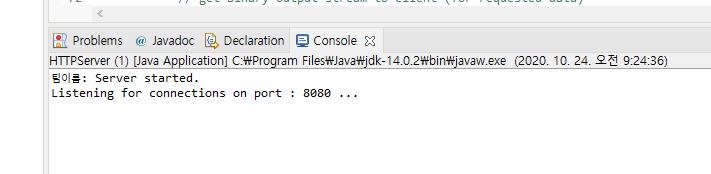
1. package webserver;
2. import java.io.BufferedOutputStream;
3. import java.io.BufferedReader;
4. import java.io.File;
5. import java.io.FileInputStream;
6. import java.io.FileNotFoundException;
7. import java.io.IOException;
8. import java.io.InputStreamReader;
9. import java.io.OutputStream;
10. import java.io.PrintWriter;
11. import java.net.ServerSocket;
12. import java.net.Socket;
13. import java.util.Date;
14. import java.util.StringTokenizer;
15. // The tutorial can be found just here on the SSaurel's Blog :
16. // https://www.ssaurel.com/blog/create-a-simple-http-web-server-in-java
17. // Each Client Connection will be managed in a dedicated Thread
18. public class HTTPServer implements Runnable{
20. static final File WEB\_ROOT = new File(".");
21. static final String DEFAULT\_FILE = "index.html";
22. static final String FILE\_NOT\_FOUND = "404.html";
23. static final String METHOD\_NOT\_SUPPORTED = "not\_supported.html";
24. // port to listen connection
25. static final int PORT = 8080;
27. // verbose mode
28. static final boolean verbose = true;
30. // Client Connection via Socket Class
31. private Socket connect;
33. public HTTPServer(Socket c) {
34. connect = c;
35. }
37. public static void main(String[] args) {
38. try {
39. ServerSocket serverConnect = new ServerSocket(PORT);
40. System.out.println("팀이름: Server started.\nListening for connections on port : " + PORT + " ...\n");
42. // we listen until user halts server execution
43. while (true) {
44. HTTPServer myServer = new HTTPServer(serverConnect.accept());
46. if (verbose) {
47. System.out.println("Connecton opened. (" + new Date() + ")");
48. }
50. // create dedicated thread to manage the client connection
51. Thread thread = new Thread(myServer);
52. thread.start();
53. }
55. } catch (IOException e) {
56. System.err.println("Server Connection error : " + e.getMessage());
57. }
58. }
59. @Override
60. public void run() {
61. // we manage our particular client connection
62. BufferedReader in = null; PrintWriter out = null; BufferedOutputStream dataOut = null;
63. String fileRequested = null;
65. try {
66. // we read characters from the client via input stream on the socket
67. in = new BufferedReader(new InputStreamReader(connect.getInputStream()));
68. // we get character output stream to client (for headers)
69. out = new PrintWriter(connect.getOutputStream());
70. // get binary output stream to client (for requested data)
71. dataOut = new BufferedOutputStream(connect.getOutputStream());
73. // get first line of the request from the client
74. String input = in.readLine();
75. // we parse the request with a string tokenizer
76. StringTokenizer parse = new StringTokenizer(input);
77. String method = parse.nextToken().toUpperCase(); // we get the HTTP method of the client
78. // we get file requested
79. fileRequested = parse.nextToken().toLowerCase();
81. // we support only GET and HEAD methods, we check
82. if (!method.equals("GET") && !method.equals("HEAD")) {
83. if (verbose) {
84. System.out.println("501 Not Implemented : " + method + " method.");
85. }
87. // we return the not supported file to the client
88. File file = new File(WEB\_ROOT, METHOD\_NOT\_SUPPORTED);
89. int fileLength = (int) file.length();
90. String contentMimeType = "text/html";
91. //read content to return to client
92. byte[] fileData = readFileData(file, fileLength);
94. // we send HTTP Headers with data to client
95. out.println("HTTP/1.1 501 Not Implemented");
96. out.println("Server: Java HTTP Server from SSaurel : 1.0");
97. out.println("Date: " + new Date());
98. out.println("Content-type: " + contentMimeType);
99. out.println("Content-length: " + fileLength);
100. out.println(); // blank line between headers and content, very important !
101. out.flush(); // flush character output stream buffer
102. // file
103. dataOut.write(fileData, 0, fileLength);
104. dataOut.flush();
106. } else {
107. // GET or HEAD method
108. if (fileRequested.endsWith("/")) {
109. fileRequested += DEFAULT\_FILE;
110. }
112. File file = new File(WEB\_ROOT, fileRequested);
113. int fileLength = (int) file.length();
114. String content = getContentType(fileRequested);
116. if (method.equals("GET")) { // GET method so we return content
117. byte[] fileData = readFileData(file, fileLength);
119. // send HTTP Headers
120. out.println("HTTP/1.1 200 OK");
121. out.println("Server: Java HTTP Server from SSaurel : 1.0");
122. out.println("Date: " + new Date());
123. out.println("Content-type: " + content);
124. out.println("Content-length: " + fileLength);
125. out.println(); // blank line between headers and content, very important !
126. out.flush(); // flush character output stream buffer
128. dataOut.write(fileData, 0, fileLength);
129. dataOut.flush();
130. }
132. if (verbose) {
133. System.out.println("File " + fileRequested + " of type " + content + " returned");
134. }
136. }
138. } catch (FileNotFoundException fnfe) {
139. try {
140. fileNotFound(out, dataOut, fileRequested);
141. } catch (IOException ioe) {
142. System.err.println("Error with file not found exception : " + ioe.getMessage());
143. }
145. } catch (IOException ioe) {
146. System.err.println("Server error : " + ioe);
147. } finally {
148. try {
149. in.close();
150. out.close();
151. dataOut.close();
152. connect.close(); // we close socket connection
153. } catch (Exception e) {
154. System.err.println("Error closing stream : " + e.getMessage());
155. }
157. if (verbose) {
158. System.out.println("Connection closed.\n");
159. }
160. }

163. }
165. private byte[] readFileData(File file, int fileLength) throws IOException {
166. FileInputStream fileIn = null;
167. byte[] fileData = new byte[fileLength];
169. try {
170. fileIn = new FileInputStream(file);
171. fileIn.read(fileData);
172. } finally {
173. if (fileIn != null)
174. fileIn.close();
175. }
177. return fileData;
178. }
180. // return supported MIME Types
181. private String getContentType(String fileRequested) {
182. if (fileRequested.endsWith(".htm") || fileRequested.endsWith(".html"))
183. return "text/html";
184. else if (fileRequested.endsWith(".jpg") || fileRequested.endsWith(".jpeg"))
185. return "image/jpg";
186. else if (fileRequested.endsWith(".mpeg"))
187. return "video/mpeg";
188. else if (fileRequested.endsWith(".mp4"))
189. return "video/mp4";
190. else
191. return "text/plain";
192. }
194. private void fileNotFound(PrintWriter out, OutputStream dataOut, String fileRequested) throws IOException {
195. File file = new File(WEB\_ROOT, FILE\_NOT\_FOUND);
196. int fileLength = (int) file.length();
197. String content = "text/html";
198. byte[] fileData = readFileData(file, fileLength);
200. out.println("HTTP/1.1 404 File Not Found");
201. out.println("Server: Java HTTP Server from SSaurel : 1.0");
202. out.println("Date: " + new Date());
203. out.println("Content-type: " + content);
204. out.println("Content-length: " + fileLength);
205. out.println(); // blank line between headers and content, very important !
206. out.flush(); // flush character output stream buffer
208. dataOut.write(fileData, 0, fileLength);
209. dataOut.flush();
211. if (verbose) {
212. System.out.println("File " + fileRequested + " not found");
213. }
214. }
215. 엡서버를 새로 돌릴 때, 도스창에서 기존 돌고 있는 8080포트 번호의 PID를 찾아 이를 죽이고 새로 웹서버를 RUN 해야 함 또 웹서버의 IP 주소는 도스창에서 ipconfig로 확인 가능 항)

텍스트이(가) 표시된 사진

자동 생성된 설명

1. RUN 시킨 후에 서버가 8080 로 들어오는 HTTP 요청을 대기한다는 다음 표시 확인



1. 그 다음 자기 팀원의 이름을 명시한 간단한 index.html 파일을 다음과 같이 만들 것

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>여기는 팀이름 홈페이지 입니다.</title>  </head>  <body>    <p> <strong><span style = " font-size:1.3em; color: blue;">2020년 팀 이름 과제 멤버:</span></strong></p>  <pre>  <strong>  <dt>팀장: 홍길동  팀원: 김철수  팀원: 이 영희</dt>    </dl></strong>  </pre>  </body>  </html> |

1. 위에서 만든 index.html 파일을 구축한 웹 서버 경로를 찾아 올리기 할 것 (경로를 잘 확인할 것)

|  |
| --- |
|  |

1. 웹서를 구축한 pc에서 브라우저를 사용해서 로컬 호스트 웹이 정상 동작하는지 index.html를 확인할 것

|  |
| --- |
|  |

1. 웹서버에 3가지 종류 이미지 파일과 두 가지 종류의 20Mbyte 용량 이하의 mp4 파일 올리기

|  |
| --- |
|  |
| 서로 용량이 다른 jpg 사진 파일 3개를 구하는데, 파일의 크기가 각각, 정확하지는 않아도 되나,  첫째 사진 w.jpg는 용량이 100Kbytes 부근, 사진 f.jpg는 용량이 500Kbytes 부근, 사진 n.jpg는 용량이 1000Kbytes 부근의 크기를 갖는 파일로 올릴 것  둘째, 동영상 mp4 파일 두개를 촬영하여 올릴것 |

1. 브라우져로 사용할 핸드폰 (또는 노트북, 다른 PC)를 통하여

<http://xxx.xxx.xxx.xxx:8080>

을 입력하여 index.html 확인

1. 브라우져로 사용할 핸드폰 (또는 노트북, 다른 PC)를 통하여

<http://xxx.xxx.xxx.xxx:8080/w.jpg>

[http://xxx. http://xxx.xxx.xxx.xxx:8080/f.jpg](http://xxx. http://xxx.xxx.xxx.xxx:8080/f.jpgxxx.xxx.xxx:8080/n.jpg)

[xxx.xxx.xxx:8080/n.jpg](http://xxx. http://xxx.xxx.xxx.xxx:8080/f.jpgxxx.xxx.xxx:8080/n.jpg)을 입력하여 3개의 사진 확인

1. 브라우져로 사용할 핸드폰 (또는 노트북, 다른 PC)를 통하여

http://xxx.xxx.xxx.xxx:8080/p1.mp4 을 입력하여 2개의 동영상 확인