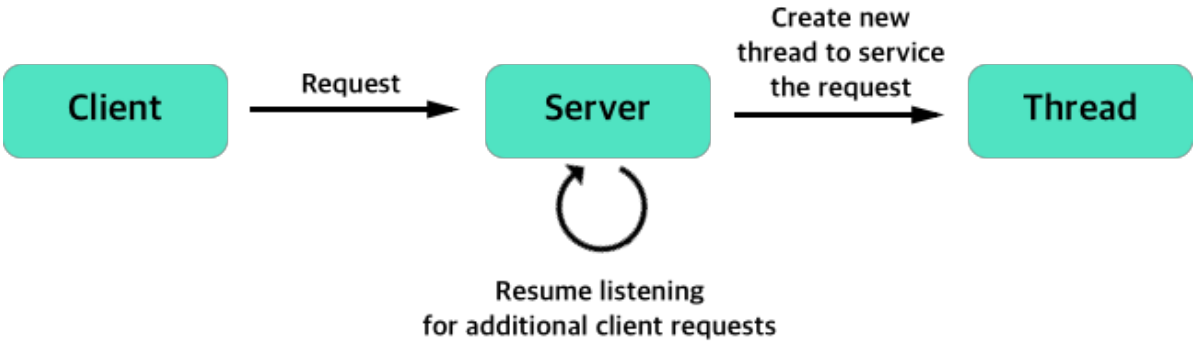
Building a Multi-Threaded Web Server

Principle of program operation



이번 과제에서는 네트워크 통신 프로토콜 중 HTTP(Hyper Text Transfer Protocol)를 이용해서 웹 서버를 구현한다. HTTP 는 포트 넘버 80 을 사용하지만 본 과제에서는 다른 request 충돌을 방지하기 위해 임의의 포트 넘버(4444 로 테스트함)를 사용했다. 위 그림에서 Server 는 항상 Client 의 Request(요청)을 기다리고 있다. Client 가 Server 에 어떤 요청을 하면 Server 는 Thread 로 이 요청을 넘기고 다시 다른 Client 의 요청을 받기 위해 기다린다. 웹 서버를 만들기 위해 Java 로 구현했는데 생성된 Thread 는 class 로 되어있기 때문에 여러 개가 생성될 수 있다(Object 가 여러 개 생성되는 원리). 생성된 Thread 들은 client 의 요청을 처리하게 되면 자동 소멸된다. 이러한 thread 를 Multi-thread 라 하고 Multi-thread 를 사용하는 웹 서버를 Multi-thread Web Server 라 한다.

Web Server in Java: Part A

1. WebServer.java

```
package multithread_web;

import java.net.*;
```

```java
public final class WebServer {
    private static ServerSocket ss; // make ServerSocket as ss

    public static void main(String argv[]) throws Exception {
        // set the port number
        int port = 4444;
        ss = new ServerSocket(port);

        while (true) {
            // listening clients request
            Socket sc = ss.accept();
            // Construct an object to process the HTTP request message.
            HttpRequest request = new HttpRequest(sc);
            // Create a new thread to process the request.
            Thread thread = new Thread(request);
            // Start the thread.
            thread.start();
        }

    }
}
```

Client 들의 request 를 받기 위해 WebServer 클래스를 만들었다. 포트 넘버는 4444 로 했고 ss 에 새로운 TCP ServerSocket(port)를 생성했다. While 문을 계속 돌리는 이유는 request 를 계속 받기 위함인데 ss.accept()로 sc 라는 소켓 변수에 request 받은 소켓을 넣고 있는 것이다. Request 받은 소켓 sc 는 HttpRequest class 의 생성자(constructor)로 인해 request 가 Setting 되고 thread 가 시작된다. Thread method 는 java.lang 라이브러이에 있는데 java.lang 은 import 하지 않고 사용가능하다.

2. HttpRequest.java(just for test Part A)

```java
package multithread_web;

import java.io.*;
import java.net.*;
//import java.util.*;

final class HttpRequest implements Runnable {
    final static String CRLF = "\r\n";
    Socket socket;

    // Constructor
    public HttpRequest(Socket socket) throws Exception {
```

```
        this.socket = socket;
    }

    // Implement the run() method of the Runnable interface.
    public void run() {
        try {
            processRequest();
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    private void processRequest() throws Exception {
        // Get a reference to the socket's input and output streams.
        InputStream is = socket.getInputStream();
        DataOutputStream os = new DataOutputStream(socket.getOutputStream());
        // Set up input stream filters.
        BufferedReader br = new BufferedReader(new InputStreamReader(is));
        // Get the request line of the HTTP request message.
        String requestLine = br.readLine();
        // Display the request line.
        System.out.println();
        System.out.println(requestLine);

        // Get and display the header lines.
        String headerLine = null;
        while ((headerLine = br.readLine()).length() != 0) {
            System.out.println(headerLine);
        }
        // Close streams and socket.
        os.close();
        br.close();
        socket.close();
    }
}
```

Implements Runnable 로 thread 를 만들면 재사용과 일관성을 높일 수 있기 때문에 Runnable 을 이용했다. 먼저 인스턴스(객체)를 초기화하기 위해 생성자(constructor)를 만들고 public void run()을 통해 processRequest 를 하게 된다. getInputStream()을 이용해서 is 에 저장, getOutputStream()과 DataOutputStream()을 이용해서 바이트를 기본자료형으로 os 에 저장, 마지막으로 is 를 이용해서 input 스트림을 해석해 br 에 넣어준다. Br 에는 HTTP request message 가 있는데 이것을 자료형이 스트링인 requestLine 에 저장하고 출력한다. headerLine 를 받기 위해 null 값으로 초기화한 변수를 선언하고 While 문을 이용해 headeLine 을 받고 출력한다.

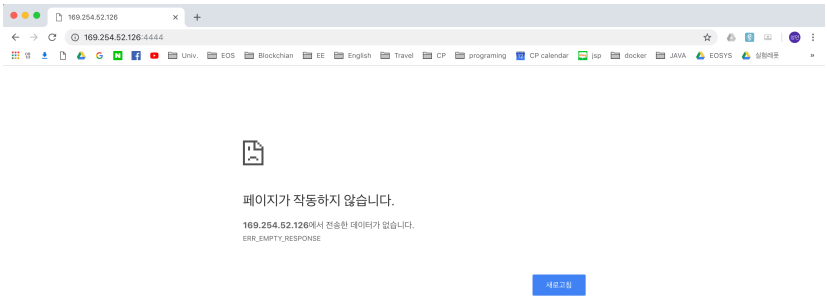
requestlLine 과 headerLine 을 출력하면 다음과 같다.

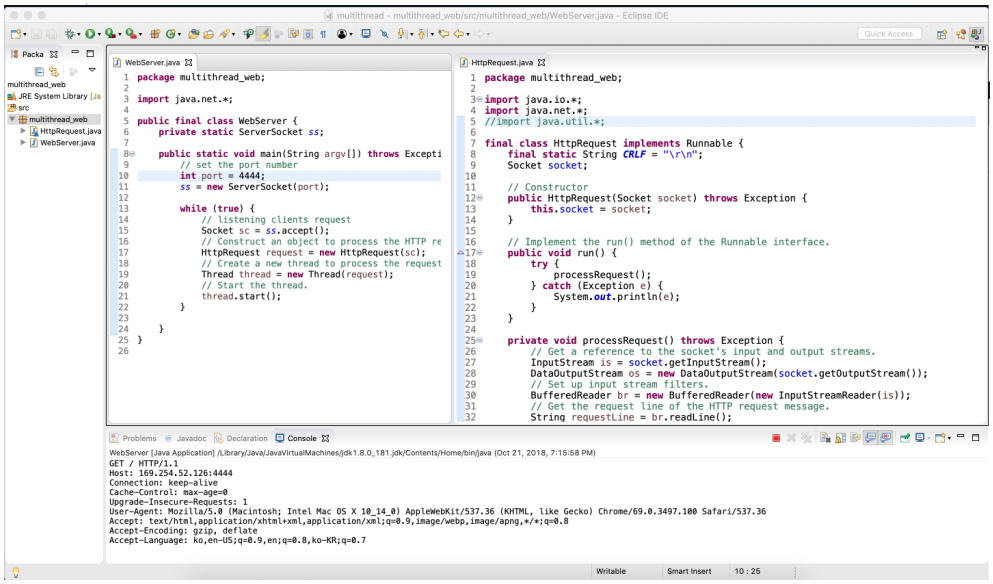
```
GET /index.html HTTP/1.1  -> requestLine

Host: 169.254.52.126:4444
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_0) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/69.0.3497.100 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: ko,en-US;q=0.9,en;q=0.8,ko-KR;q=0.7
```

첫 줄이 requestLine 이고 나머지가 headerLine 이다. 이제 스트림을 먼저 닫아주고(close) 소켓을 닫아주면 된다.

3. Request & Output





html 등 다른 파일 요청을 처리해주는 코드를 구성하지 않았기 때문에 위와 같이 서버에서만 구동되는 것을 볼 수 있다.

Web Server in Java: Part B

1. HttpRequest.java(full version)

```java
package multithread_web;

import java.io.*;
import java.net.*;
import java.util.*;

final class HttpRequest implements Runnable {
    final static String CRLF = "\r\n";
    Socket socket;

    // Constructor
    public HttpRequest(Socket socket) throws Exception {
        this.socket = socket;
    }

    // Implement the run() method of the Runnable interface.
    public void run() {
        try {
            processRequest();
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    private void processRequest() throws Exception {
        // Get a reference to the socket's input and output streams.
        InputStream is = socket.getInputStream();
        DataOutputStream os = new DataOutputStream(socket.getOutputStream());
        // Set up input stream filters.
        BufferedReader br = new BufferedReader(new InputStreamReader(is));
        // Get the request line of the HTTP request message.
        String requestLine = br.readLine();
        // Display the request line.
        System.out.println();
        System.out.println(requestLine);

        // Get and display the header lines.
        String headerLine = null;
        while ((headerLine = br.readLine()).length() != 0) {
            System.out.println(headerLine);
        }

        // Extract the filename from the request line.
        StringTokenizer tokens = new StringTokenizer(requestLine);
        tokens.nextToken(); // skip over the method, which should be "GET"
        String fileName = tokens.nextToken();
        // Prepend a "." so that file request is within the current directory.
```

```
        fileName = "." + fileName;

        // Open the requested file.
        FileInputStream fis = null;
        boolean fileExists = true;
        try {
            fis = new FileInputStream(fileName);
        } catch (FileNotFoundException e) {
            fileExists = false;
        }

        // Construct the response message.
        String statusLine = null;
        String contentTypeLine = null;
        String entityBody = null;
        if (fileExists) {
            statusLine = "HTTP/1.1 200 OK: ";
            contentTypeLine = "Content-type: " + contentType(fileName) + CRLF;
        } else {
            statusLine = "HTTP/1.1 404 Not Found: ";
            contentTypeLine = "Content-type: text/html" + CRLF;
            entityBody = "<HTML>" + "<HEAD><TITLE>Not Found</TITLE></HEAD>" +
"<BODY>Not Found</BODY></HTML>";
        }

        // Send the status line.
        os.writeBytes(statusLine);
        // Send the content type line.
        os.writeBytes(contentTypeLine);
        // Send a blank line to indicate the end of the header lines.
        os.writeBytes(CRLF);

        // Send the entity body.
        if (fileExists) {
            sendBytes(fis, os);
            fis.close();
        } else {
            os.writeBytes(entityBody);
        }

        // Close streams and socket.
        os.close();
        br.close();
        socket.close();
    }

    private static void sendBytes(FileInputStream fis, OutputStream os) throws
Exception {
        // Construct a 1K buffer to hold bytes on their way to the socket.
        byte[] buffer = new byte[1024];
```

```
        int bytes = 0;
        // Copy requested file into the socket's output stream.
        while ((bytes = fis.read(buffer)) != -1) {
            os.write(buffer, 0, bytes);
        }
    }

    private static String contentType(String fileName) {
        if (fileName.endsWith(".htm") || fileName.endsWith(".html")) {
            return "text/html";
        }
        if (fileName.endsWith(".jpg")) {
            return "image/jpeg";
        }
        if (fileName.endsWith(".mp3")) {
            return "audio/mpeg3";
        }
        return "application/octet-stream";
    }
} // The end
```

2. 추가된 부분만 설명

1) request line 에서 파일 이름 찾아내기

```
        // Extract the filename from the request line.
        StringTokenizer tokens = new StringTokenizer(requestLine);
        tokens.nextToken(); // skip over the method, which should be "GET"
        String fileName = tokens.nextToken();
        // Prepend a "." so that file request is within the current directory.
        fileName = "." + fileName;
```

StringTokenizer 는 java.util 에 포함되어 있는 자체 클래스이다. nextToken()으로 다음 문자로 넘어갈 수 있는데 두 번 넘어가면 filename 이 나오므로 저장해주고 나중에 .fileName 으로 사용해야 해서 "."을 붙여준다.

2) 요청된 파일 open

```
        // Open the requested file.
        FileInputStream fis = null;
        boolean fileExists = true;
        try {
            fis = new FileInputStream(fileName);
        } catch (FileNotFoundException e) {
            fileExists = false;
        }
```

fileName 으로 file 을 open 하는데 try-catchr 구문을 이용해서 file 존재 여부를 FileNotFoundException 으로 알려준다.

3) status 출력

```
// Construct the response message.
        String statusLine = null;
        String contentTypeLine = null;
        String entityBody = null;
        if (fileExists) {
            statusLine = "HTTP/1.1 200 OK: ";
            contentTypeLine = "Content-type: " + contentType(fileName) + CRLF;
        } else {
            statusLine = "HTTP/1.1 404 Not Found: ";
            contentTypeLine = "Content-type: text/html" + CRLF;
            entityBody = "<HTML>" + "<HEAD><TITLE>Not Found</TITLE></HEAD>" +
"<BODY>Not Found</BODY></HTML>";
        }
```

앞쪽에서 boolean 변수로 잡아준 fileExists 를 이용해서 파일이 있으면 200 메세지를 statusLine 에 추가하고 contentTypeLine 에 file 의 type 을 추가해준다. 파일이 없으면 404 메세지를 statusLine 에 추가하고 contentTypeLine 에 text/html 을 추가해준다. 이러한 이유는 "Not Found" 메세지가 html 로 entityBody 를 형성하고 있기 때문이다.

4) client 에게 status 와 file 전송

```
        // Send the status line.
        os.writeBytes(statusLine);
        // Send the content type line.
        os.writeBytes(contentTypeLine);
        // Send a blank line to indicate the end of the header lines.
        os.writeBytes(CRLF);

        // Send the entity body.
        if (fileExists) {
            sendBytes(fis, os);
            fis.close();
        } else {
            os.writeBytes(entityBody);
        }
```

위에서 정의한 statusLine 과 contentTypeLine 을 먼저 보내주고 CRLF="₩r₩n"을 보내준다. 이 CRLF 는 개행 문자로 텍스트의 한 줄이 끝남을 알려준다. 곧바로 file 의 존재함을 판단해 존재하면 file 을 전송하고 존재하지 않으면 "Not Found" 메세지를 client 브라우저에 띄워준다.

5) file 을 전송하는 sendBytes method

```
    private static void sendBytes(FileInputStream fis, OutputStream os) throws
Exception {
        // Construct a 1K buffer to hold bytes on their way to the socket.
        byte[] buffer = new byte[1024]; int bytes = 0;
        // Copy requested file into the socket's output stream.
        while ((bytes = fis.read(buffer)) != -1) {
            os.write(buffer, 0, bytes);
        }
    }
```

FileInputStream 과 OutputStream 을 받아서 1KB 크기의 buffer 로 client 에게 전송한다.

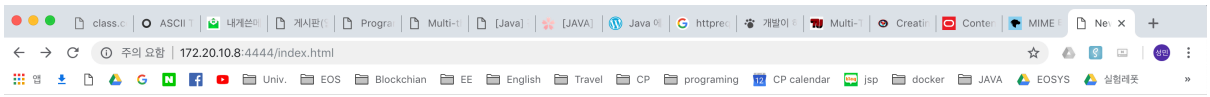
6) file 의 type 을 알아내는 contentType method

```
private static String contentType(String fileName) {
        if (fileName.endsWith(".htm") || fileName.endsWith(".html")) {
            return "text/html";
        }
        if (fileName.endsWith(".jpg")) {
            return "image/jpeg";
        }
        if (fileName.endsWith(".mp3")) {
            return "audio/mpeg3";
        }
        return "application/octet-stream";
    }
}
```

fileName 을 받아서 if 문으로 file 의 확장자를 알아내서 MIME 형태로 변환 후 return 해준다. 이 서버에서는 위의 확장자를 가진 파일들만 open 할 수 있다.

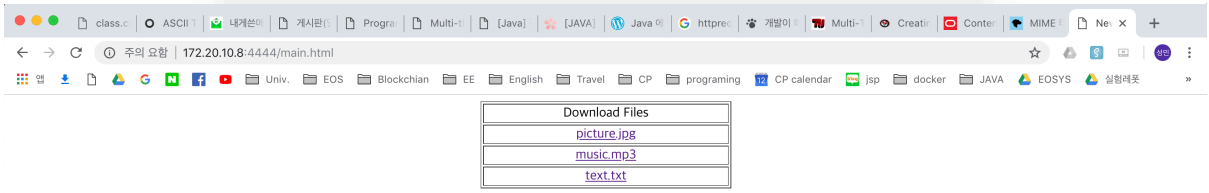
3. Request & Output

차례대로 index.html / main.html / picture.jpg / music.mp3 / text.txt 이다.



<< Computer Network Project Page >>

enter



Download Files
picture.jpg
music.mp3
text.txt