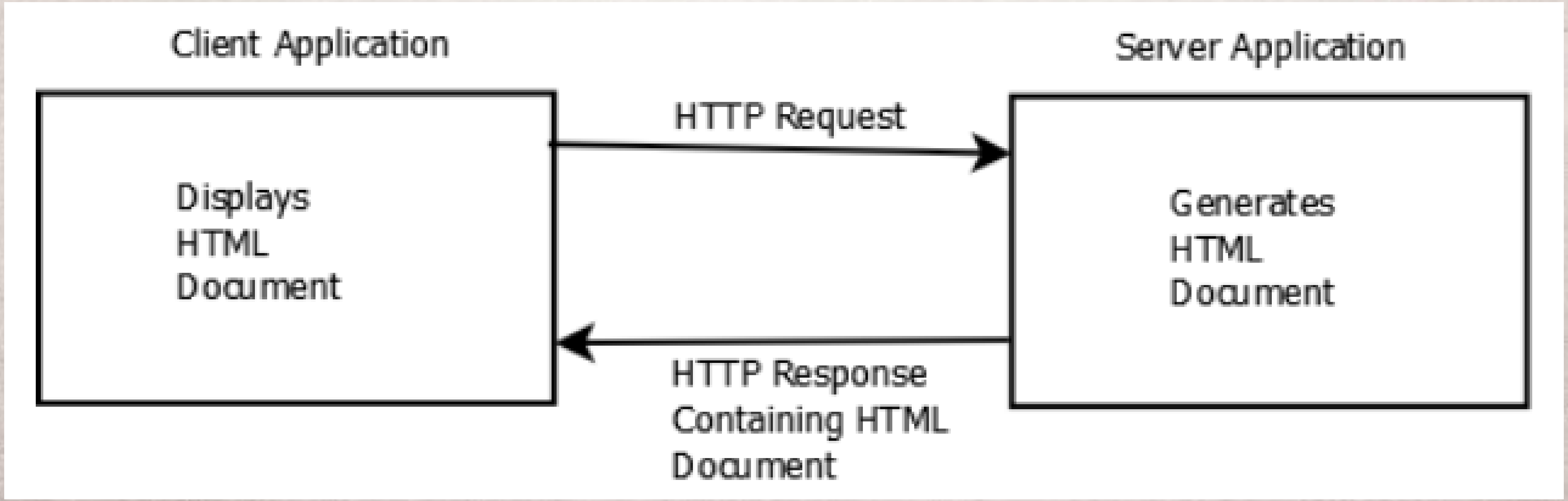
Building a simple HTTP Server/Client

comsi.java@gmail.com
박 경 태

http://github.com/hopypark

HTTP protocol structure

- **HTTP**는 **World Wide Web(WWW)**를 통해 리소스를 전달하기 위해 사용
- 리소스는 일반적으로 **HTML(HyperText Markup Language)** 파일이 사용되고, 내부적으로 이미지, 오디오, 비디오와 같은 다른 종류의 파일도 포함
- 리소스를 열기 위해 브라우저에서 **URL(Uniform Resource Locator)**을 사용



HTTP Evolvement

- HTTP/1.0은 1980년대~1990년대까지 정리되었고 1991년에 첫번째 문서가 배포
- HTTP/1.1 버전은 2014년 6월에 6개 부분으로 배포
- HTTP 2.0에 대한 Request For Comments(RFC) 문서가 2015년 5월에 배포

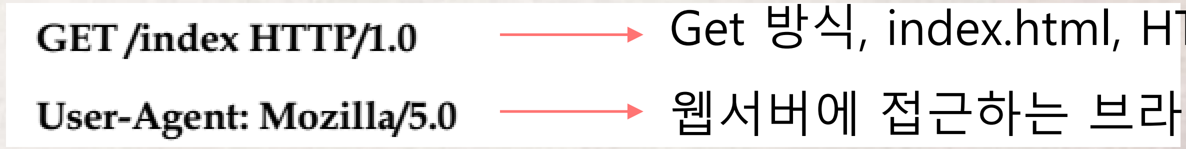
Version	Reference
HTTP 1.0	http://www.w3.org/Protocols/HTTP/1.0/spec.html
HTTP/1.1	http://tools.ietf.org/html/rfc2616
HTTP/2	https://en.wikipedia.org/wiki/HTTP/2

RFC – 컴퓨터 네트워크 공학 등에서 인터넷 기술에 적용 가능한 새로운 연구, 혁신, 기법 등을 아우르는 메모

Nature of HTTP messages

- **Message**는 서버가 클라이언트에게 보내는 **Response,** 클라이언트가 서버에게 보내는 **Request, 2** 종류가 있고 다음과 같은 구조를 가진다.
  - A line indicating the type of message
  - Zero or more header lines
  - A blank line
  - An optional message body containing data

Example of an HTTP request:



```
GET /index HTTP/1.0
User-Agent: Mozilla/5.0
```

→ Get 방식, index.html, HTTP/1.0

→ 웹서버에 접근하는 브라우저의 명칭

- Request message
  - initial request line and zero or more header lines
- Response message
  - initial response line(**status line**), zeros or more header lines, and optional message body

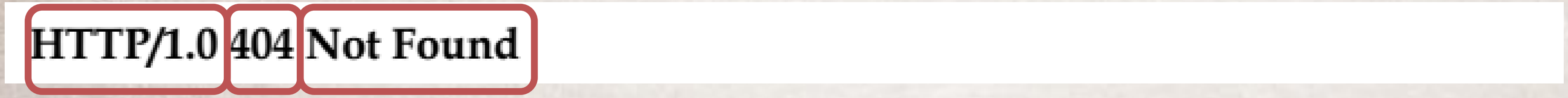
Initial request line format

- Request method name → GET, POST
- Local path of the resource → 접근하고자하는 리소스 위치
- The HTTP version

GET /index HTTP/1.0

Initial response line format

- The HTTP version
- A response status code
- A response phrase describing the code



```
HTTP/1.0 404 Not Found
```

list of more commonly used codes

Status code	Standard text	Meaning
200	**OK**	This indicates that the request was a success
400	**Bad Request**	This indicates that request access was incorrect
401	**Unauthorized**	This indicates that the resource is restricted often because the login attempt failed
403	**Forbidden**	This indicates that access to the requested resource is forbidden
404	**Not Found**	This indicates that the resource is no longer available
405	**Method Not Allowed**	A request method is not supported for the requested resource; for example, a GET request on a form which requires data to be presented via POST, or a PUT request on a read-only resource.
500	**Internal server error**	This reflects some sort of error with the server
502	**Service Gateway**	This indicates that the gateway server received an invalid response from another server
503	**Service Unavailable**	This indicates that the server is not available

Complete➔ list-https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

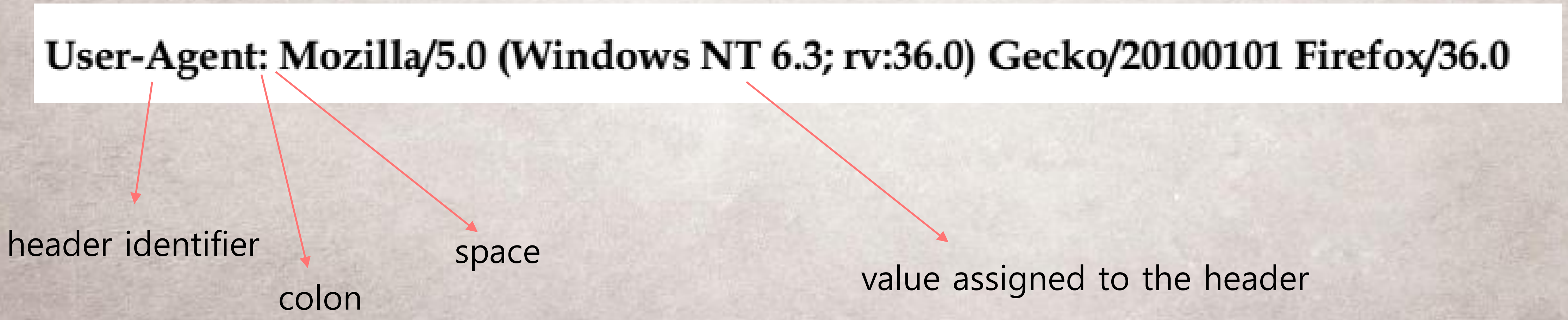
list of more commonly used codes

- The status code is a three-digit number. The first digit of this number reflects the category of the code:

  - 1xx: This represents an informational message
  - 2xx: This represents a success
  - 3xx: This redirects the client to another URL
  - 4xx: This represents a client error
  - 5xx: This represents a server error

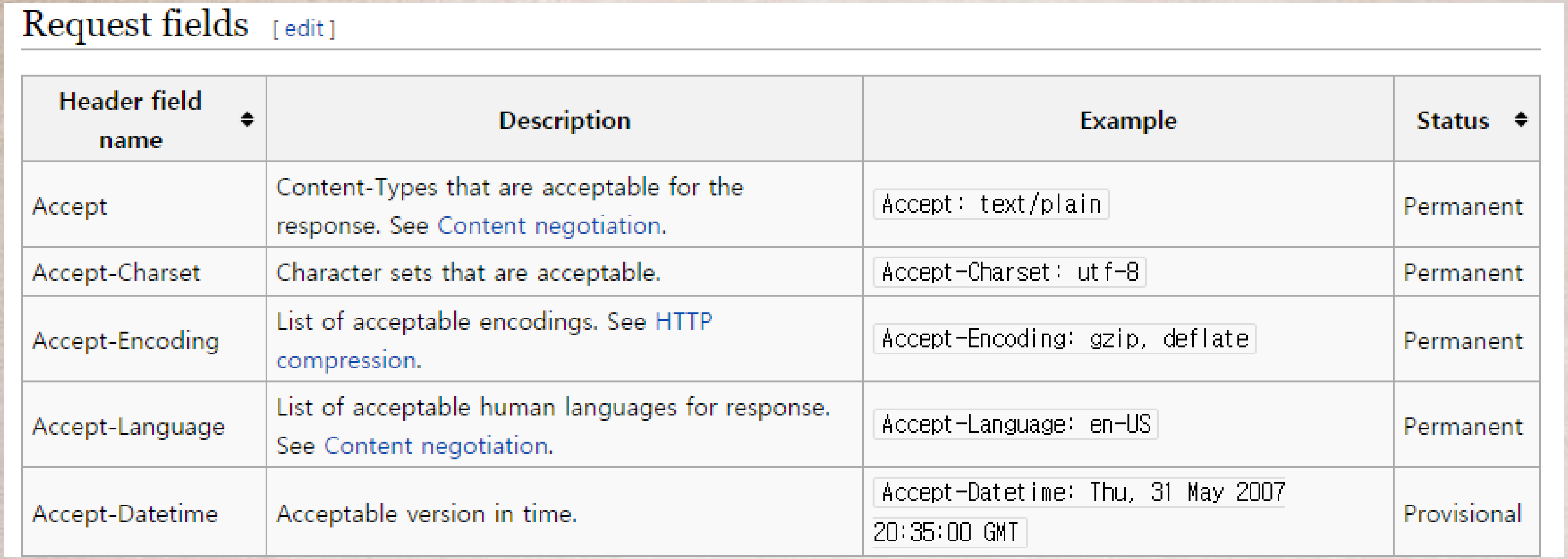
Header lines

- Headers lines provide information regarding the request or response, such as e-mail address, application identifier
- the header consists of a single line:
  - header identifier,
  - a colon,
  - spaces,
  - and then the value assigned to the header



User-Agent: Mozilla/5.0 (Windows NT 6.3; rv:36.0) Gecko/20100101 Firefox/36.0

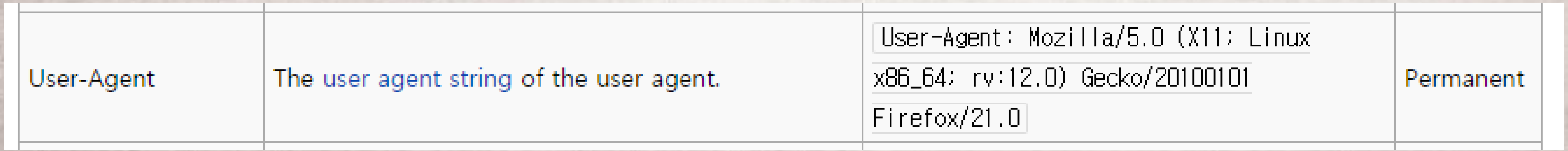
Header lines – request fields



Request fields [edit]

Header field name	Description	Example	Status
Accept	Content-Types that are acceptable for the response. See Content negotiation.	`Accept: text/plain`	Permanent
Accept-Charset	Character sets that are acceptable.	`Accept-Charset: utf-8`	Permanent
Accept-Encoding	List of acceptable encodings. See HTTP compression.	`Accept-Encoding: gzip, deflate`	Permanent
Accept-Language	List of acceptable human languages for response. See Content negotiation.	`Accept-Language: en-US`	Permanent
Accept-Datetime	Acceptable version in time.	`Accept-Datetime: Thu, 31 May 2007 20:35:00 GMT`	Provisional

• • •

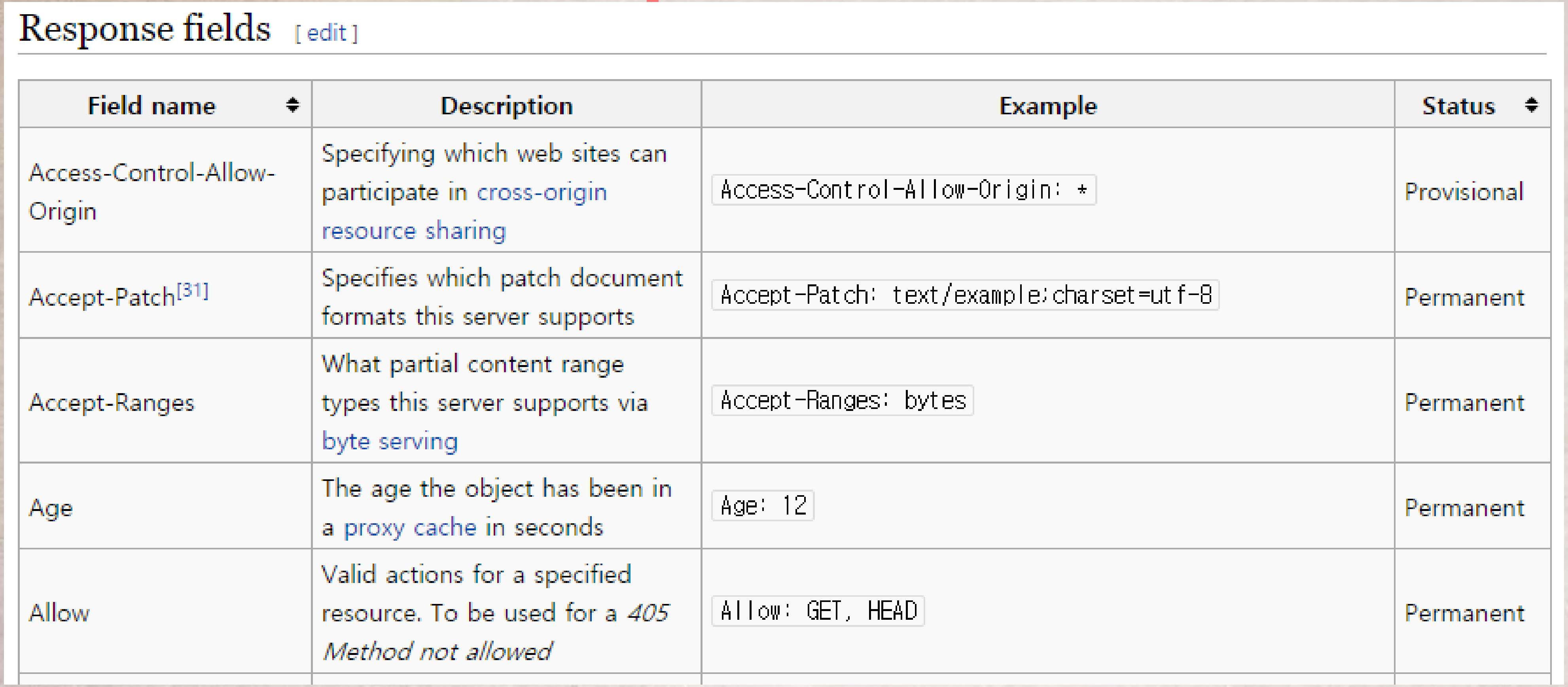


User-Agent	The user agent string of the user agent.	`User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/21.0`	Permanent

• • •

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

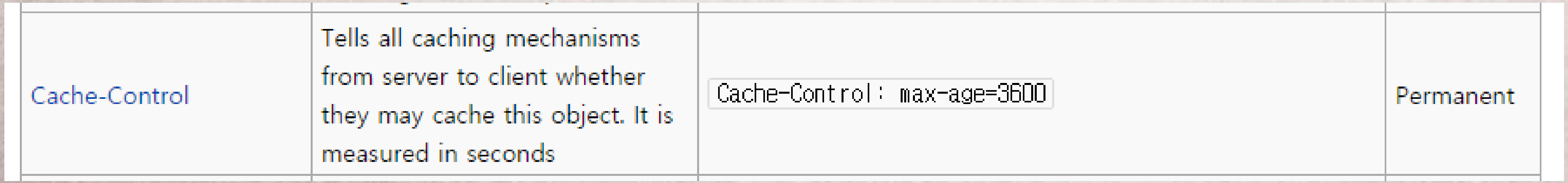
Header lines – response fields



Response fields [edit]

Field name	Description	Example	Status
Access-Control-Allow-Origin	Specifying which web sites can participate in cross-origin resource sharing	`Access-Control-Allow-Origin: *`	Provisional
Accept-Patch[31]	Specifies which patch document formats this server supports	`Accept-Patch: text/example;charset=utf-8`	Permanent
Accept-Ranges	What partial content range types this server supports via byte serving	`Accept-Ranges: bytes`	Permanent
Age	The age the object has been in a proxy cache in seconds	`Age: 12`	Permanent
Allow	Valid actions for a specified resource. To be used for a *405 Method not allowed*	`Allow: GET, HEAD`	Permanent

• • •



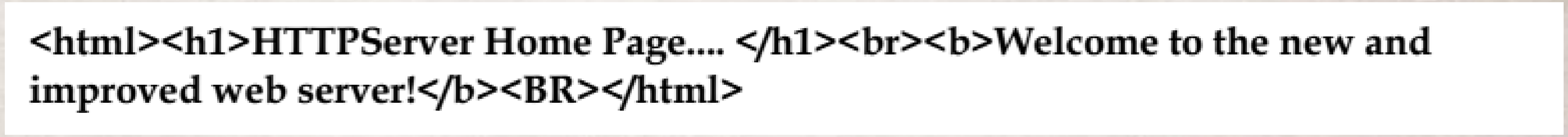
Cache-Control	Tells all caching mechanisms from server to client whether they may cache this object. It is measured in seconds	`Cache-Control: max-age=3600`	Permanent

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Message body

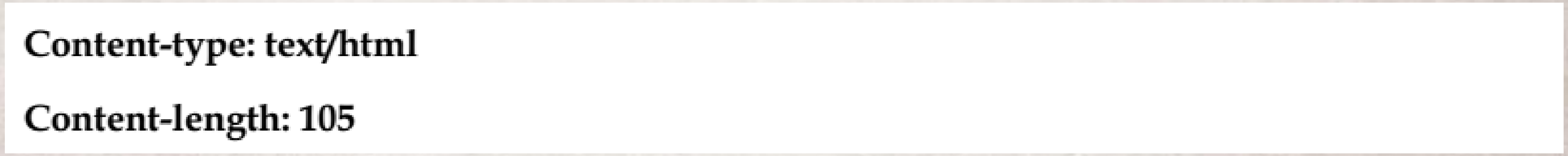
- Message body is normally included, it is optional and is not needed for some messages.
- When a body is included, the Content-Type and Content-Length header

message body (HTML)



```
<html><h1>HTTPServer Home Page.... </h1><br><b>Welcome to the new and
improved web server!</b><BR></html>
```

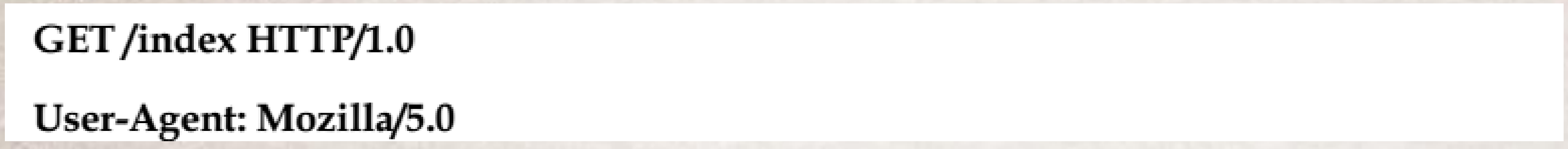
Header



```
Content-type: text/html

Content-length: 105
```

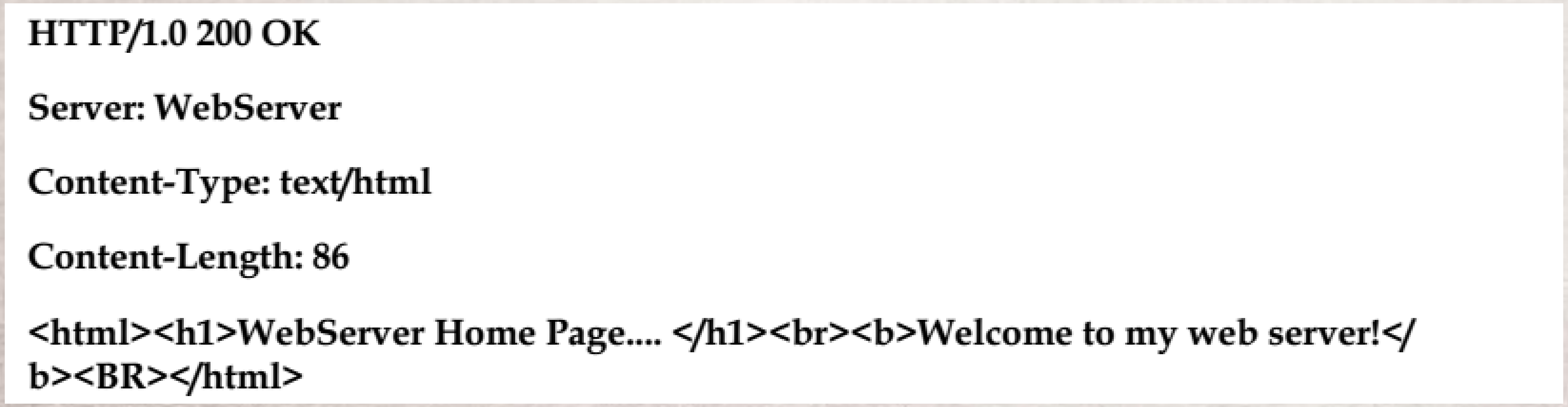
Client/Server interaction example

- A client sending a request and the server responding.
  - he client request message uses the GET method against a path of /index:



```
GET /index HTTP/1.0

User-Agent: Mozilla/5.0
```

  - The server will respond with the following message, assuming that it was able to process the request.



```
HTTP/1.0 200 OK

Server: WebServer

Content-Type: text/html

Content-Length: 86

<html><h1>WebServer Home Page.... </h1><br><b>Welcome to my web server!</
b><BR></html>
```

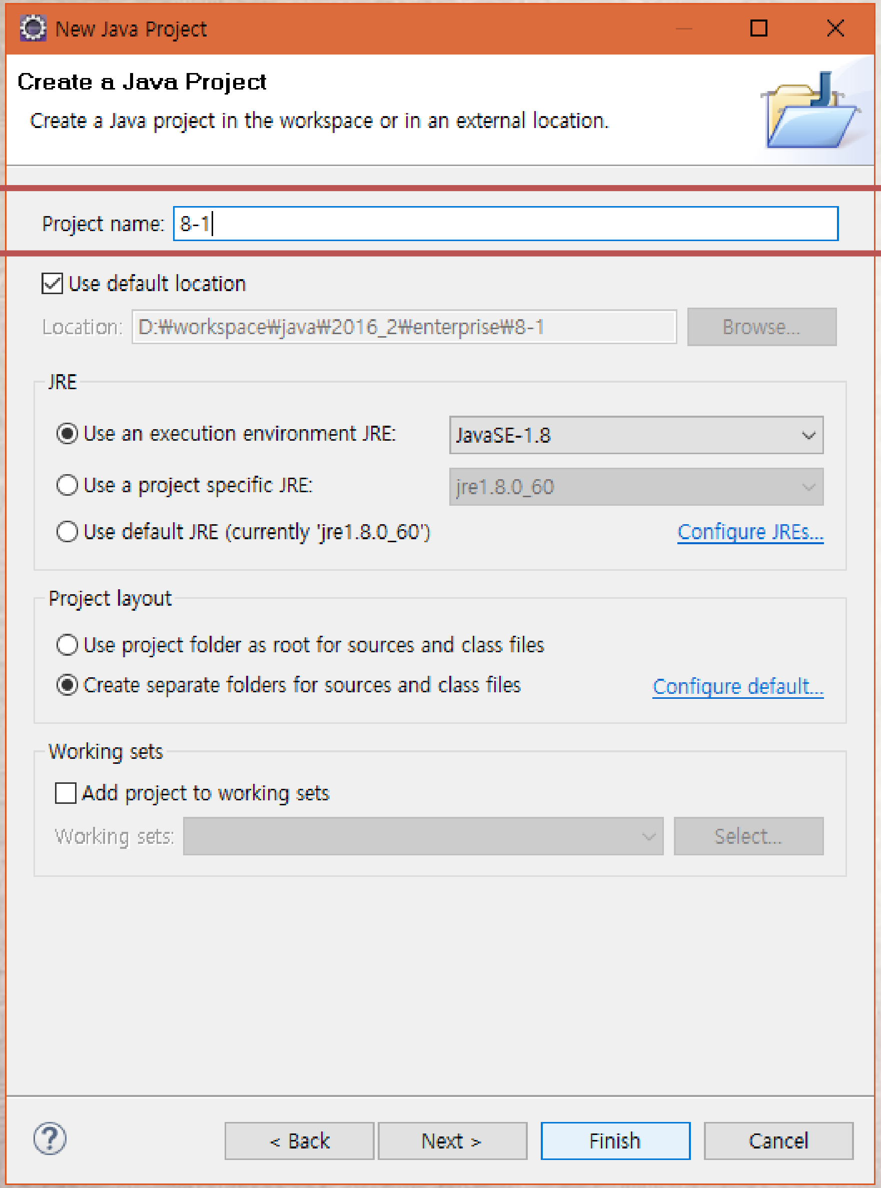
Java socket support for HTTP client/server applications

- HTTP client will make a connection to an HTTP server
- The client will send a request message to the server
- The server will send back a response message, as an HTML document.
- In the early HTTP version, once the response was sent, the server would terminate the connection(stateless protocol)
- **With HTTP/1.1, persistent connections can be maintained.**
  - improves the performance by eliminating the need to open and close connections when multiple pieces of data need to be transferred between the server and a client.
- Our server will support a subset of the HTTP/1.0 specification.
  - it helps illustrate the nature of HTTP Requests.

Building a simple HTTP server

- We will use a class called WebServer to support the HTTP/1.0 protocol.
- The server will use a ClientHandler class to handle a client.
- The server will be limited to handling only GET requests.
- Support of other methods can be easily added.

Building a simple HTTP server



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: 8-1

[✓] Use default location

Location: D:₩workspace₩java₩2016_2₩enterprise₩8-1

JRE

(●) Use an execution environment JRE: JavaSE-1.8

( ) Use a project specific JRE: jre1.8.0_60

( ) Use default JRE (currently 'jre1.8.0_60') Configure JREs...

Project layout

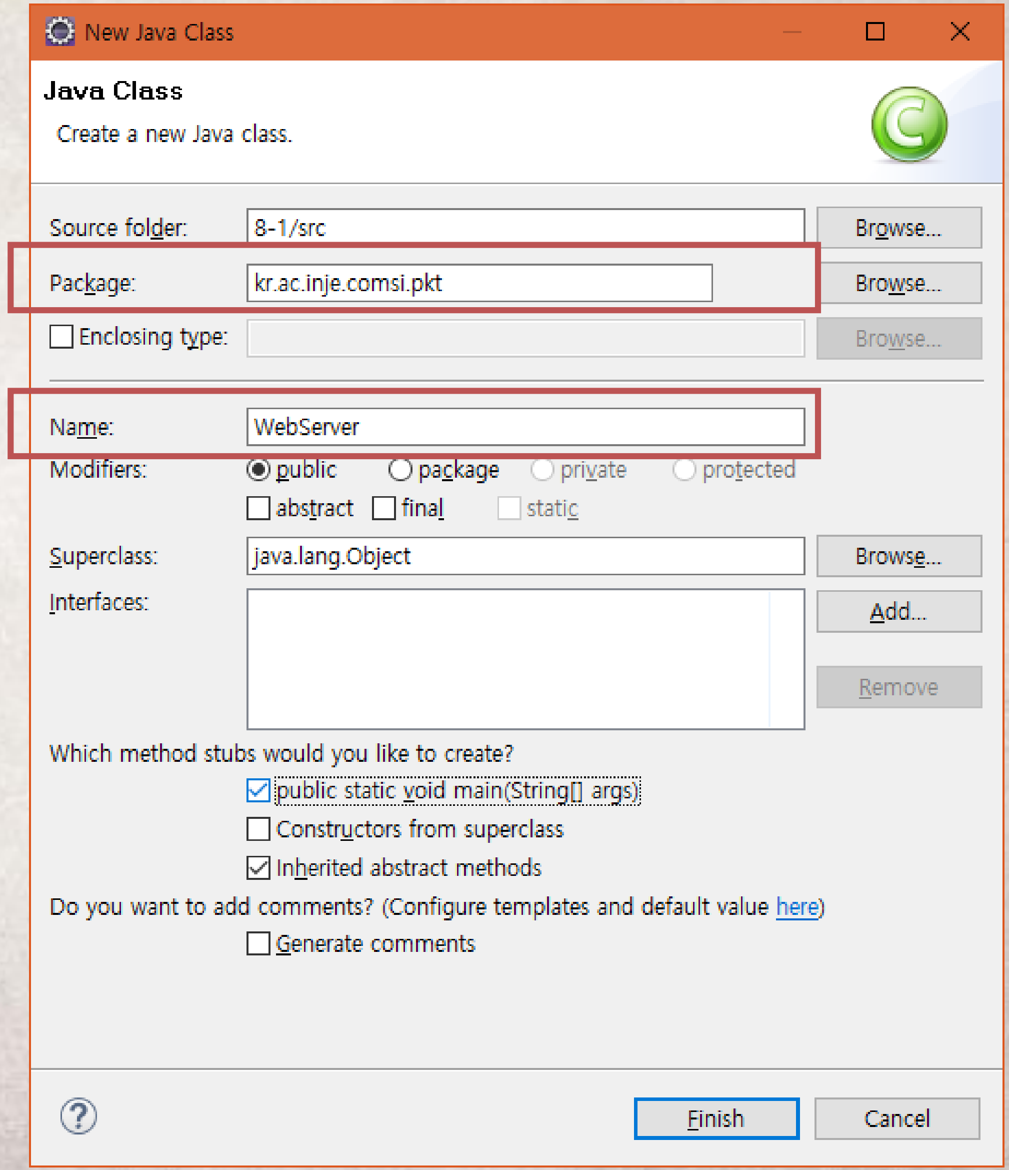
( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files Configure default...

Working sets

[ ] Add project to working sets

Working sets:



New Java Class

Java Class

Create a new Java class.

Source folder: 8-1/src

Package: kr.ac.inje.comsi.pkt

[ ] Enclosing type:

Name: WebServer

Modifiers: (●) public ( ) package ( ) private ( ) protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

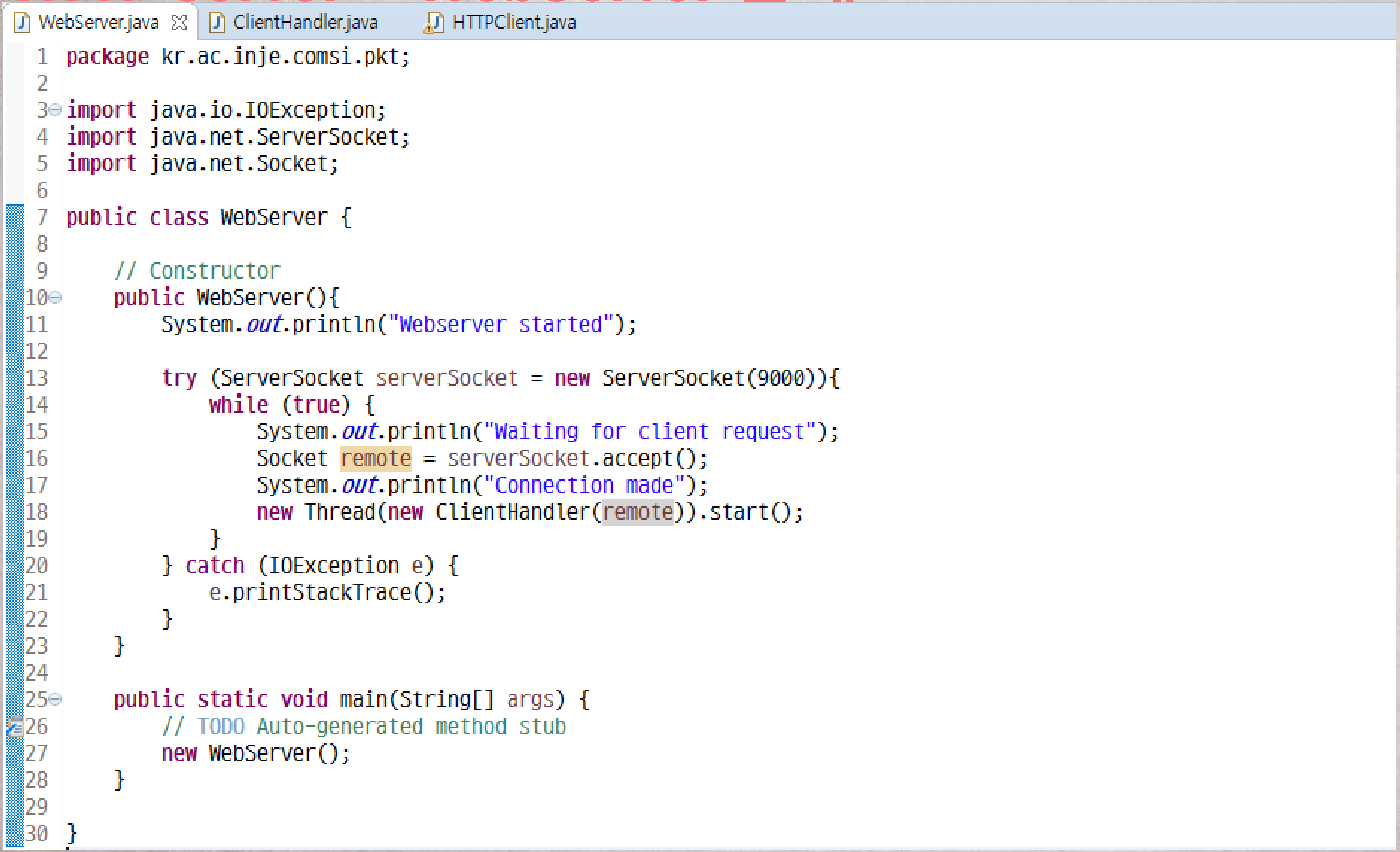
[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

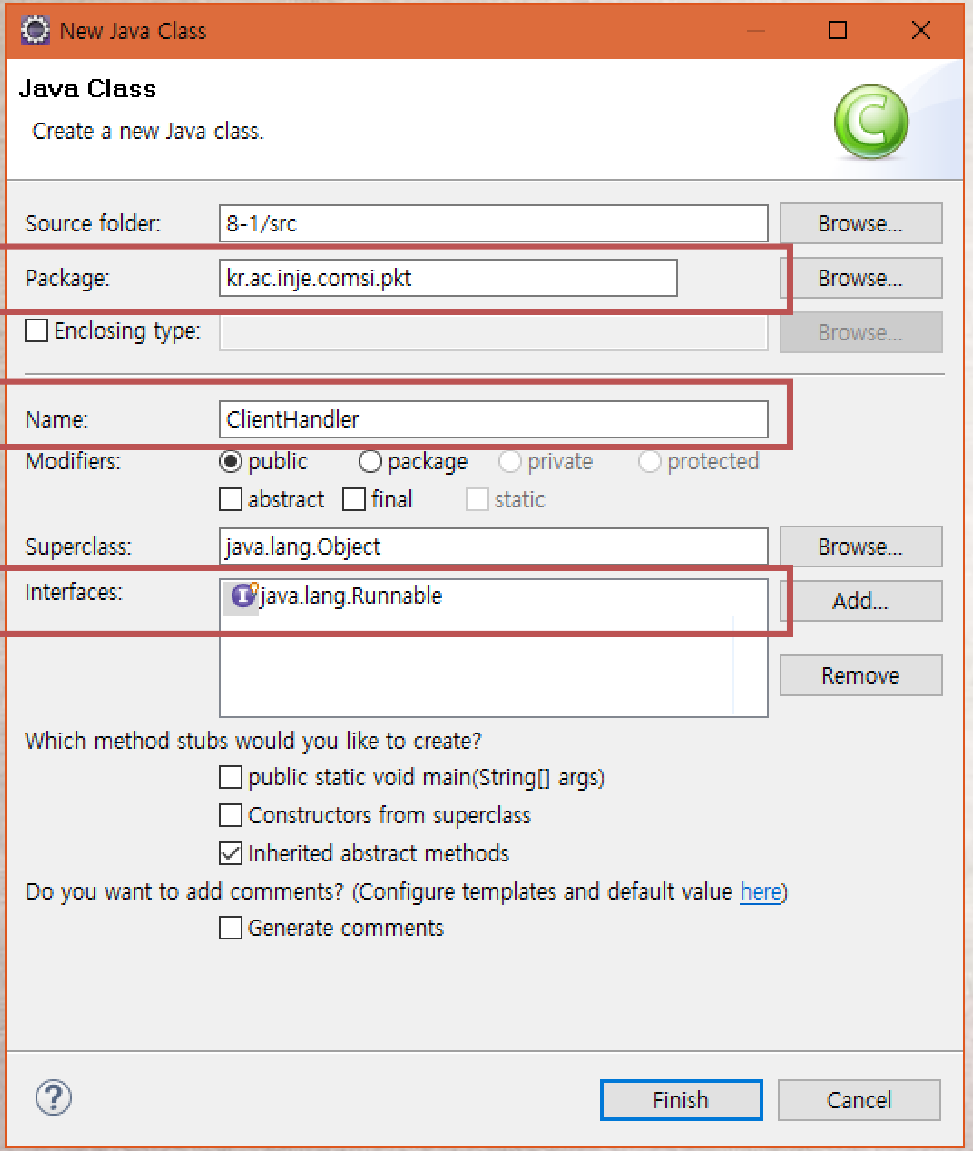
[ ] Generate comments

HTTP server – WebServer 클래스



```java
package kr.ac.inje.comsi.pkt;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class WebServer {

    // Constructor
    public WebServer(){
        System.out.println("Webserver started");

        try (ServerSocket serverSocket = new ServerSocket(9000)){
            while (true) {
                System.out.println("Waiting for client request");
                Socket remote = serverSocket.accept();
                System.out.println("Connection made");
                new Thread(new ClientHandler(remote)).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        new WebServer();
    }

}
```

HTTP server – ClientHandler 클래스


New Java Class

Java Class

Create a new Java class.

Source folder: 8-1/src

Package: kr.ac.inje.comsi.pkt

☐ Enclosing type:

Name: ClientHandler

Modifiers: [✓ public] package private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces: java.lang.Runnable

Which method stubs would you like to create?

☐ public static void main(String[] args)

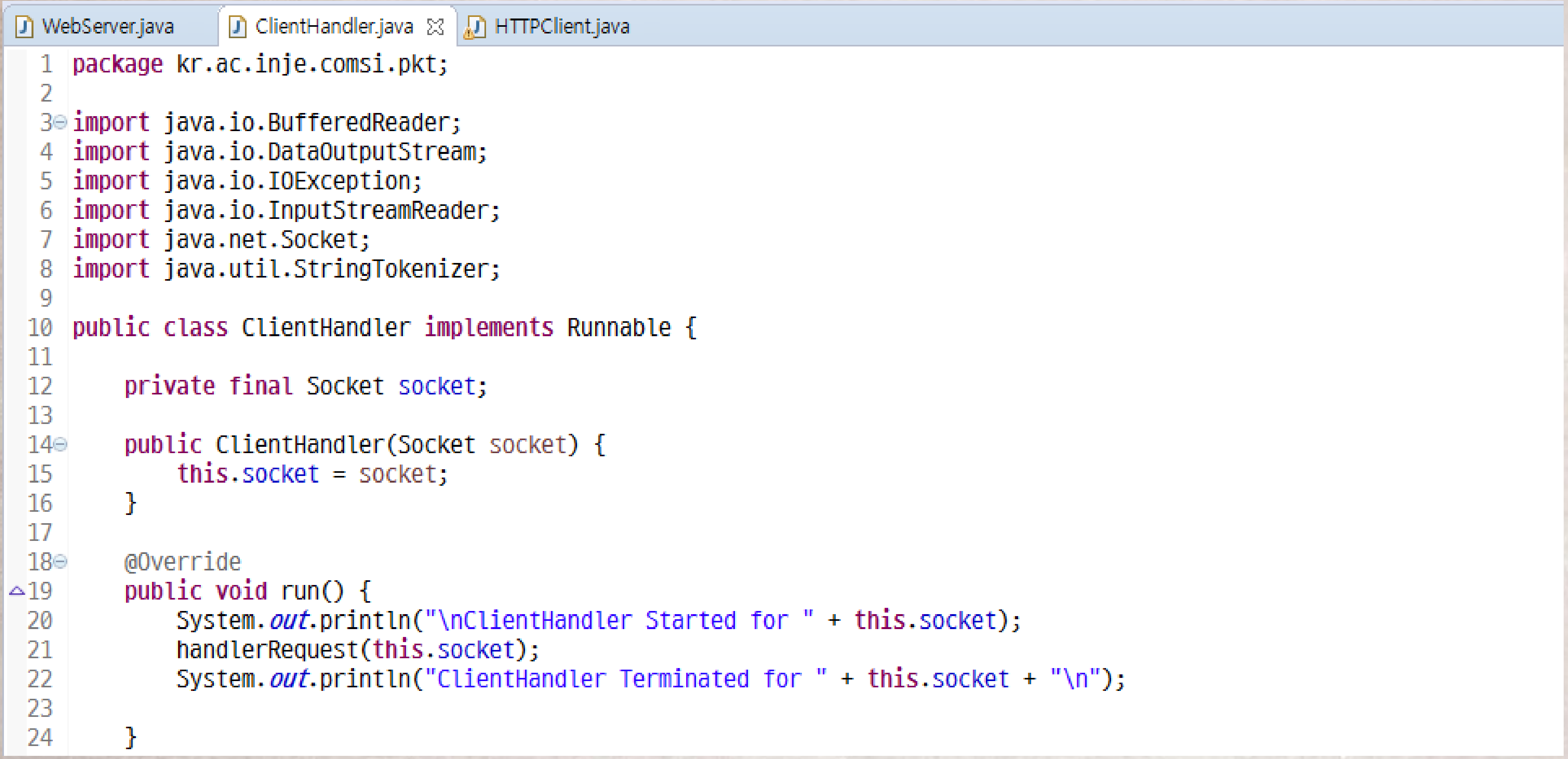
☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

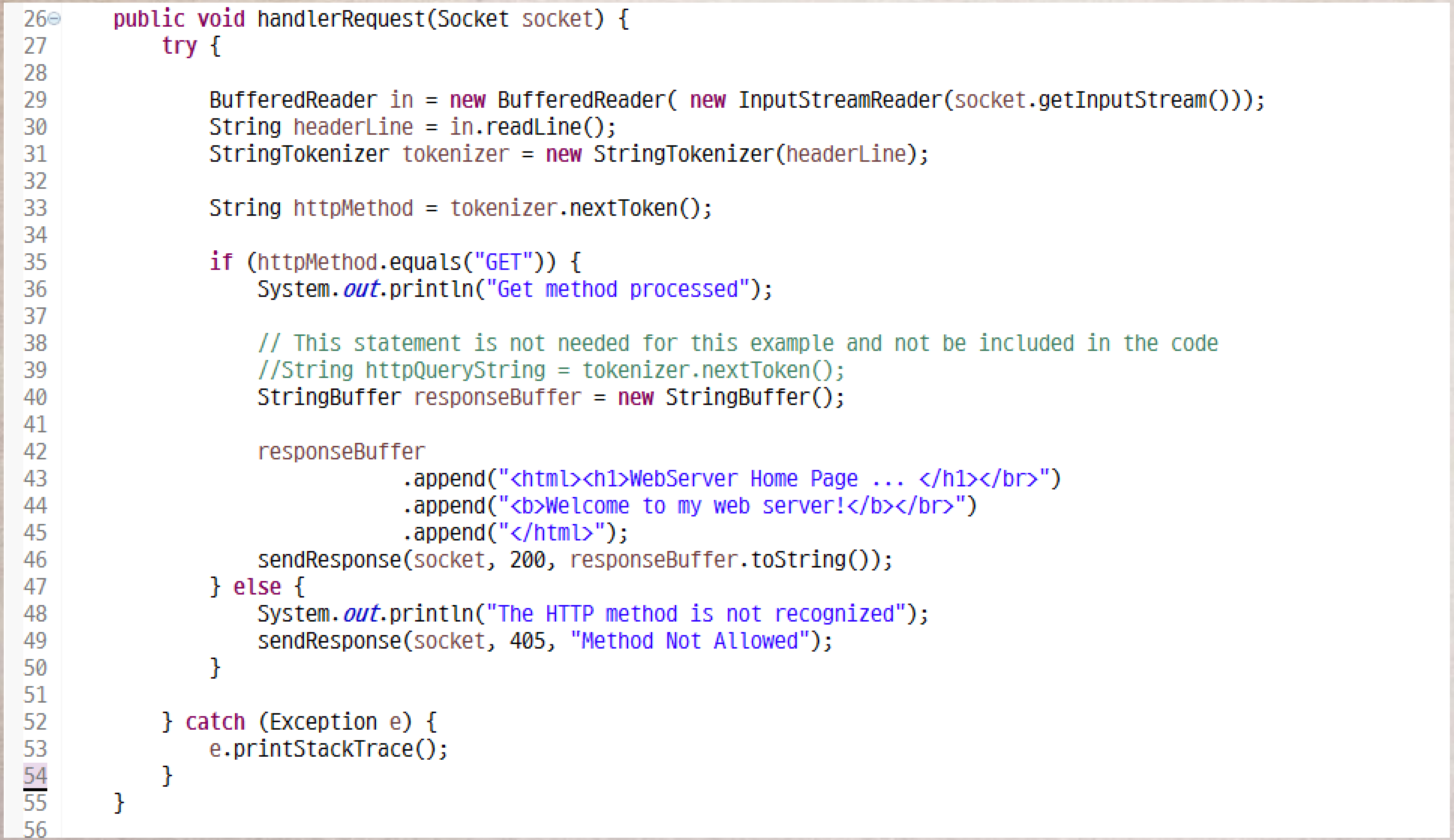
☐ Generate comments

ClientHandler 클래스



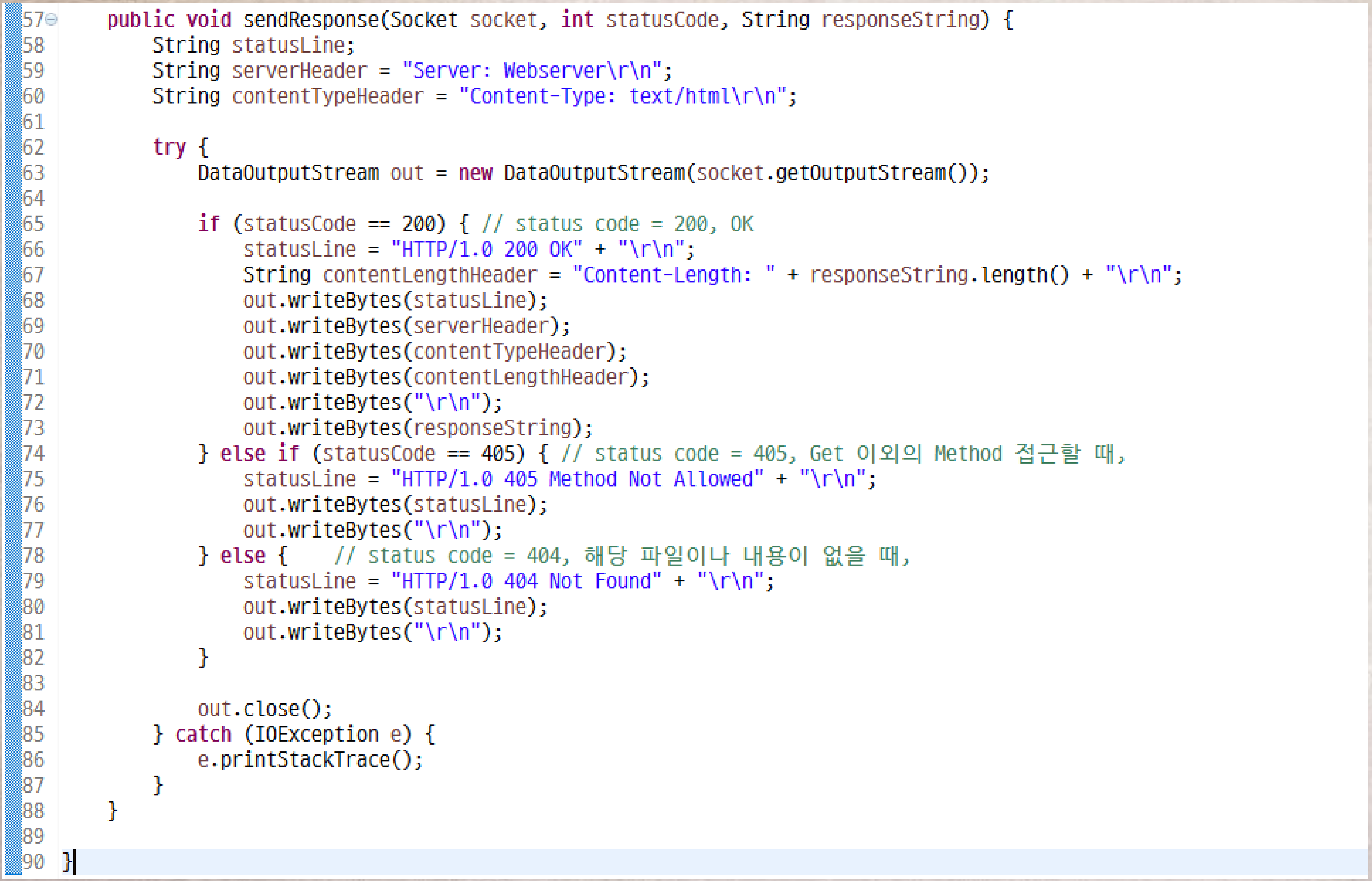
```java
package kr.ac.inje.comsi.pkt;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.util.StringTokenizer;

public class ClientHandler implements Runnable {

    private final Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        System.out.println("\nClientHandler Started for " + this.socket);
        handlerRequest(this.socket);
        System.out.println("ClientHandler Terminated for " + this.socket + "\n");

    }
```

ClientHandler 클래스



```java
    public void handlerRequest(Socket socket) {
        try {

            BufferedReader in = new BufferedReader( new InputStreamReader(socket.getInputStream()));
            String headerLine = in.readLine();
            StringTokenizer tokenizer = new StringTokenizer(headerLine);

            String httpMethod = tokenizer.nextToken();

            if (httpMethod.equals("GET")) {
                System.out.println("Get method processed");

                // This statement is not needed for this example and not be included in the code
                //String httpQueryString = tokenizer.nextToken();
                StringBuffer responseBuffer = new StringBuffer();

                responseBuffer
                            .append("<html><h1>WebServer Home Page ... </h1></br>")
                            .append("<b>Welcome to my web server!</b></br>")
                            .append("</html>");
                sendResponse(socket, 200, responseBuffer.toString());
            } else {
                System.out.println("The HTTP method is not recognized");
                sendResponse(socket, 405, "Method Not Allowed");
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

ClientHandler 클래스

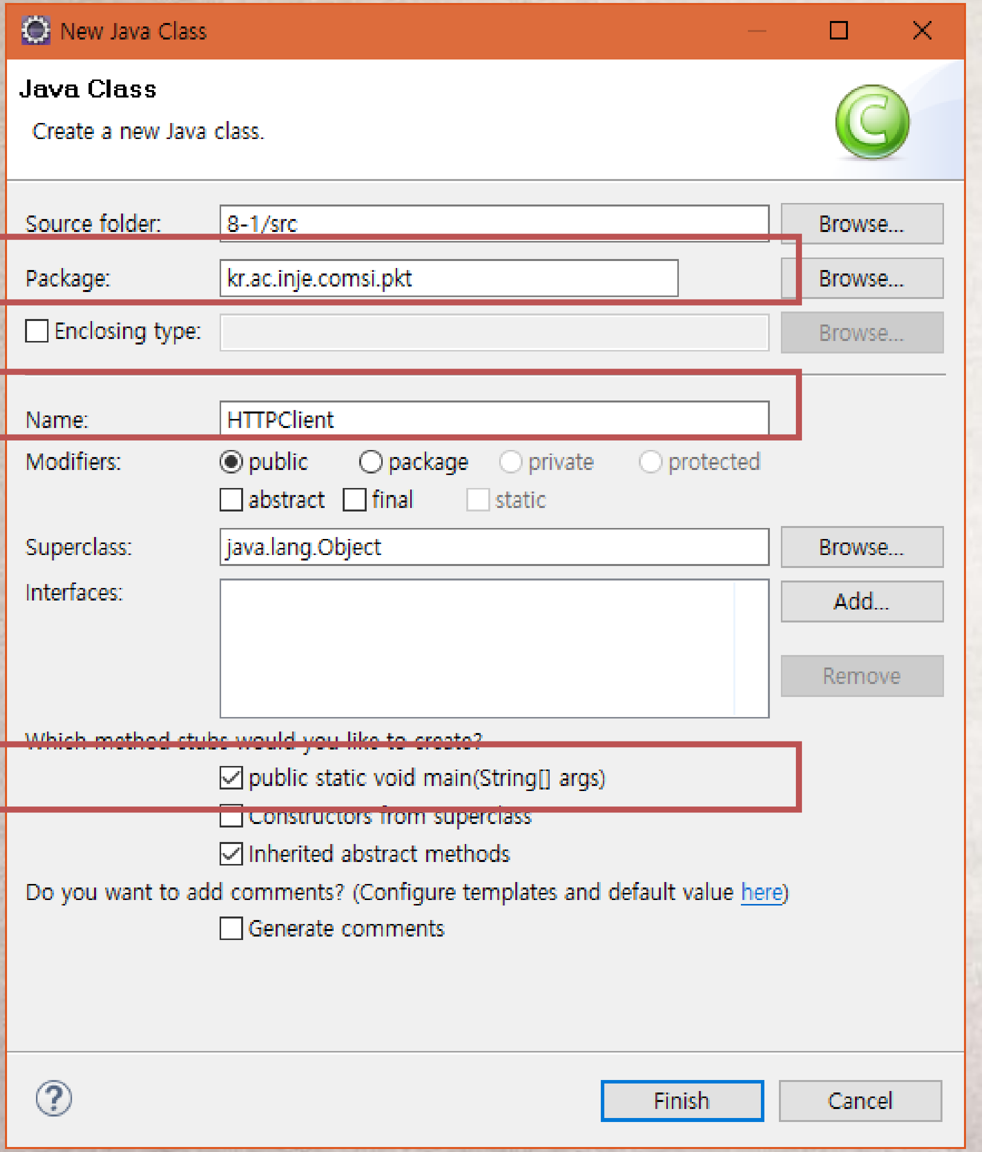


```java
    public void sendResponse(Socket socket, int statusCode, String responseString) {
        String statusLine;
        String serverHeader = "Server: Webserver\r\n";
        String contentTypeHeader = "Content-Type: text/html\r\n";

        try {
            DataOutputStream out = new DataOutputStream(socket.getOutputStream());

            if (statusCode == 200) { // status code = 200, OK
                statusLine = "HTTP/1.0 200 OK" + "\r\n";
                String contentLengthHeader = "Content-Length: " + responseString.length() + "\r\n";
                out.writeBytes(statusLine);
                out.writeBytes(serverHeader);
                out.writeBytes(contentTypeHeader);
                out.writeBytes(contentLengthHeader);
                out.writeBytes("\r\n");
                out.writeBytes(responseString);
            } else if (statusCode == 405) { // status code = 405, Get 이외의 Method 접근할 때,
                statusLine = "HTTP/1.0 405 Method Not Allowed" + "\r\n";
                out.writeBytes(statusLine);
                out.writeBytes("\r\n");
            } else {    // status code = 404, 해당 파일이나 내용이 없을 때,
                statusLine = "HTTP/1.0 404 Not Found" + "\r\n";
                out.writeBytes(statusLine);
                out.writeBytes("\r\n");
            }

            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

Building a simple HTTP client

- HTTPClient class to access our HTTP server.
- In its constructor, a socket connecting to the server is created.
- The Socket class's getInputStream and getOutputStream return input and output streams for the socket, respectively
- The sendGet method is called, which sends a request to server.
- The getResponse method returns the response, which is then displayed:

HTTPClient 클래스


New Java Class

Java Class

Create a new Java class.

Source folder: 8-1/src

Package: kr.ac.inje.comsi.pkt

☐ Enclosing type:

Name: HTTPClient

Modifiers: (● public) ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

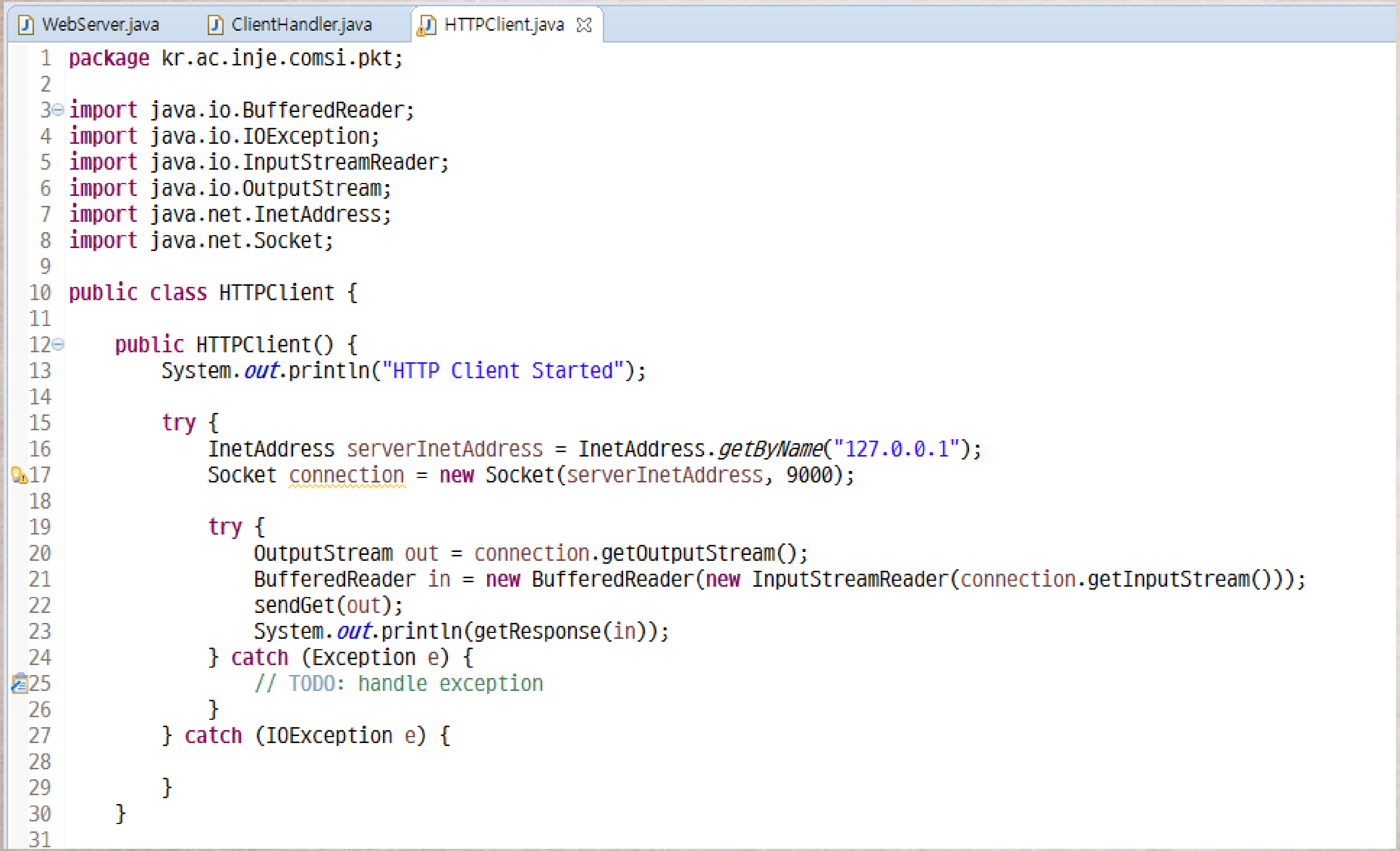
☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

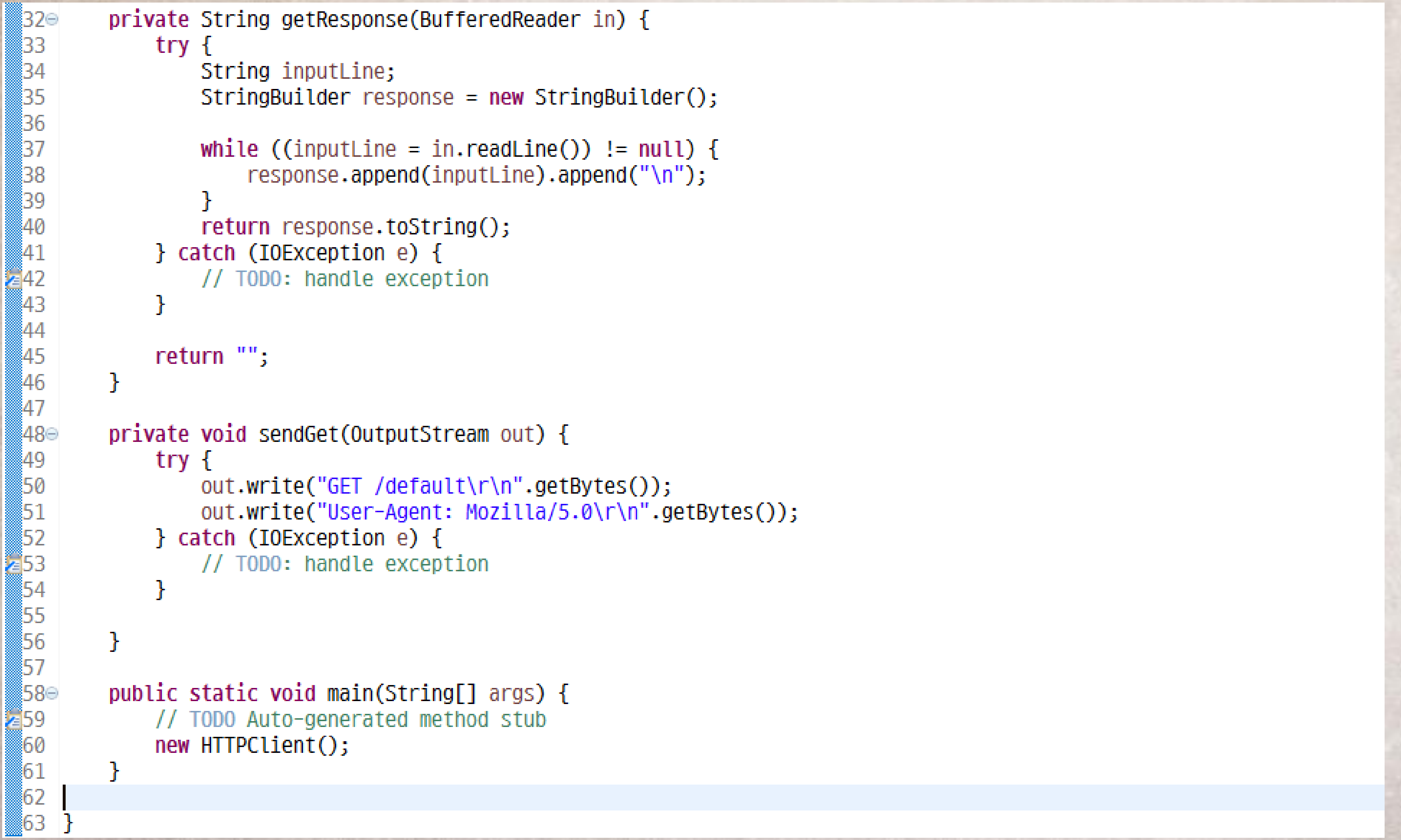
☐ Generate comments

HTTPClient 클래스



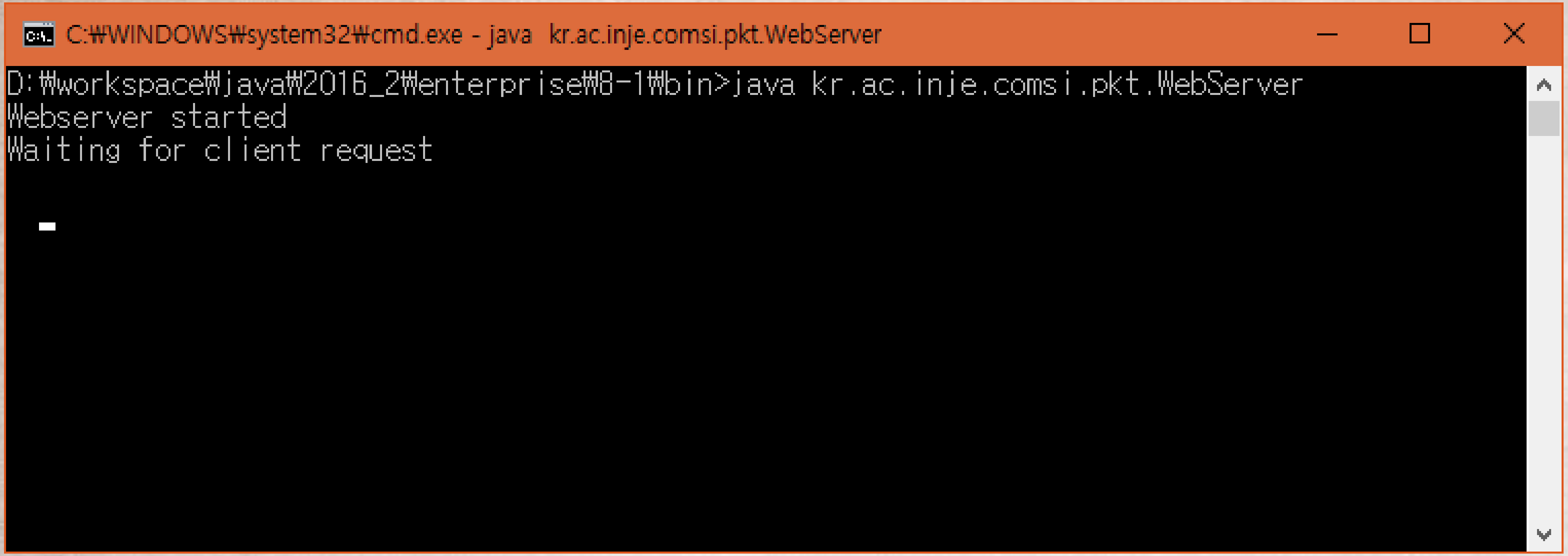
```java
package kr.ac.inje.comsi.pkt;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.Socket;

public class HTTPClient {

    public HTTPClient() {
        System.out.println("HTTP Client Started");

        try {
            InetAddress serverInetAddress = InetAddress.getByName("127.0.0.1");
            Socket connection = new Socket(serverInetAddress, 9000);

            try {
                OutputStream out = connection.getOutputStream();
                BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
                sendGet(out);
                System.out.println(getResponse(in));
            } catch (Exception e) {
                // TODO: handle exception
            }
        } catch (IOException e) {

        }
    }

```

HTTPClient 클래스



```
    private String getResponse(BufferedReader in) {
        try {
            String inputLine;
            StringBuilder response = new StringBuilder();

            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine).append("\n");
            }
            return response.toString();
        } catch (IOException e) {
            // TODO: handle exception
        }

        return "";
    }

    private void sendGet(OutputStream out) {
        try {
            out.write("GET /default\r\n".getBytes());
            out.write("User-Agent: Mozilla/5.0\r\n".getBytes());
        } catch (IOException e) {
            // TODO: handle exception
        }

    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        new HTTPClient();
    }

}
```

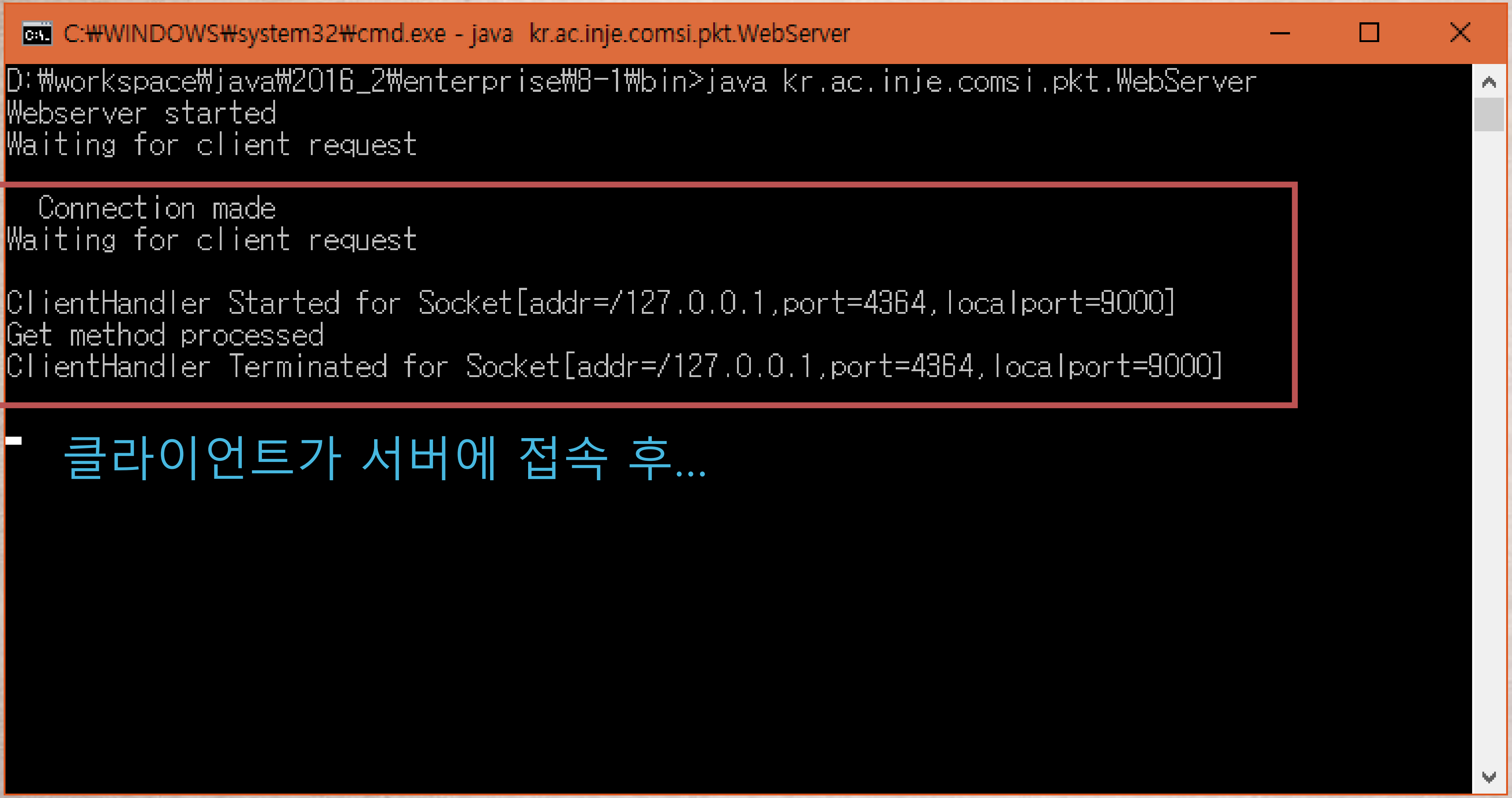
Server started.



C:₩WINDOWS₩system32₩cmd.exe - java kr.ac.inje.comsi.pkt.WebServer

```
D:₩workspace₩java₩2016_2₩enterprise₩8-1₩bin>java kr.ac.inje.comsi.pkt.WebServer
Webserver started
Waiting for client request
```

Server started

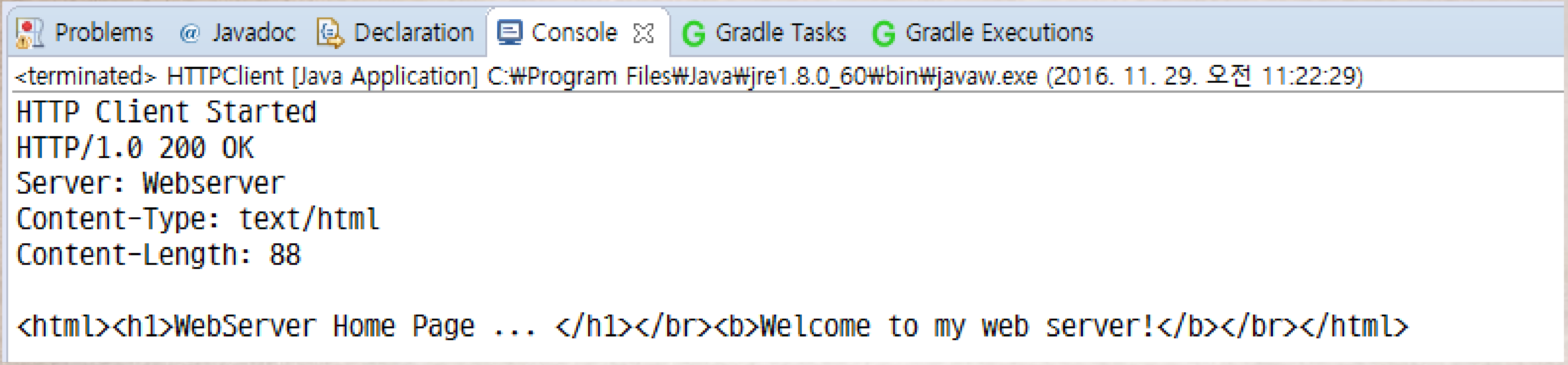


C:￦WINDOWS￦system32￦cmd.exe - java kr.ac.inje.comsi.pkt.WebServer

```
D:￦workspace￦java￦2016_2￦enterprise￦8-1￦bin>java kr.ac.inje.comsi.pkt.WebServer
Webserver started
Waiting for client request

  Connection made
Waiting for client request

ClientHandler Started for Socket[addr=/127.0.0.1,port=4364,localport=9000]
Get method processed
ClientHandler Terminated for Socket[addr=/127.0.0.1,port=4364,localport=9000]
```

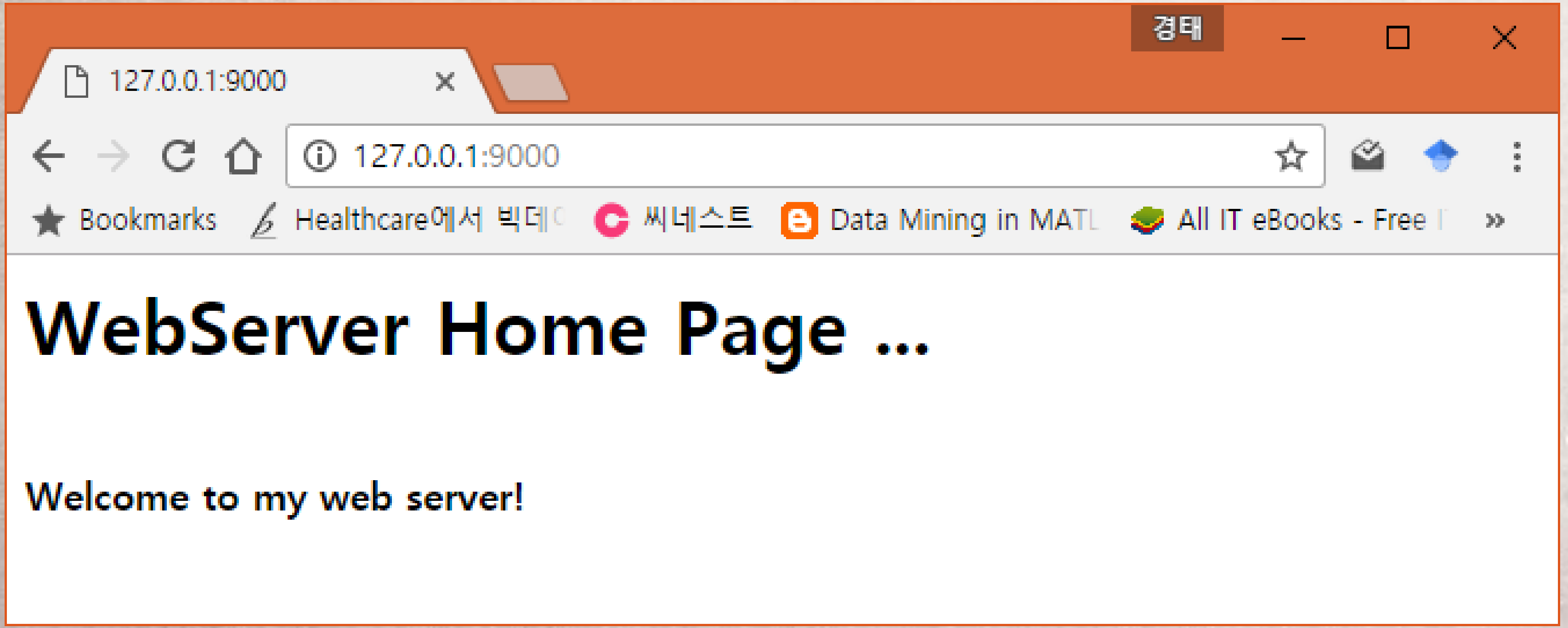
클라이언트가 서버에 접속 후…

Client started



<terminated> HTTPClient [Java Application] C:₩Program Files₩Java₩jre1.8.0_60₩bin₩javaw.exe (2016. 11. 29. 오전 11:22:29)

```
HTTP Client Started
HTTP/1.0 200 OK
Server: Webserver
Content-Type: text/html
Content-Length: 88

<html><h1>WebServer Home Page ... </h1></br><b>Welcome to my web server!</b></br></html>
```



WebServer Home Page ...

Welcome to my web server!