Server.java | HttpServer.java

```java
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;
import java.util.logging.*;

public class Server{
    //static final File ROOT = new File(".");
    static final int PORT = 8080;
    //static final String DEFAULT_FILE = "index.html";
    //static final String FILE_404 = "404.html";



    public static void main(String[] args) throws IOException{
        //Set up the loggers
        Logger actLog = Logger.getLogger("activity");
        Logger errLog = Logger.getLogger("errors");
        actLog.setLevel(Level.FINER);
        errLog.setLevel(Level.FINER);

        //Handler files for log messages
        Handler act = new FileHandler("activity.log");
        Handler err = new FileHandler("error.log");
        act.setLevel(Level.FINER);
        err.setLevel(Level.FINER);

        actLog.addHandler(act);
        errLog.addHandler(err);


        ServerSocket serverSock = new ServerSocket(PORT);
        actLog.finer("Server running on IP and Port: " + serverSock.toString());

        Executor service = Executors.newCachedThreadPool();

        while(true){
            try{
                Socket client = serverSock.accept();
                HttpServer temp = new HttpServer(client,actLog,errLog);
                service.execute(temp);
            }
            catch(SocketTimeoutException x){
                errLog.finer("Socket timed out: "+x);
            }
        }
    }
}
```

CIS 389
Term Project

Nick Mohan
Isaiah Plummer
Raymond Zhu
Nick Landry

Option One

- Building a server with Java
  - Required HTTP, logging and multi-threading
  - Required two optional features
    - BlackList/Authorization
    - HTTP METHODS
- We also constructed multiple clients to test the server's functionality and to demonstrate how it functions with proper and improper use
- For this project we used JAVA as it was versatile enough to handle our needs

First implementation(Authorization/Authentication)

- The first function implemented adds certain IP's to a whitelist/blacklist
  - This is to prevent constant spam from a given IP and provides safe authorization to authorized IPs.
  - The blacklist will constantly be updated while the whitelist remains constant.

- Authentication methods were added to communicate with the server.
  - This includes user/password log in.
  - The server will store the usernames and passwords authorized to access the server.

Second Implementation (Put, Delete, Trace)

- Aside from POST and HEAD methods, we have also decided to implement the OPTIONS method.
  - This can be used respond to the user so they know what HTTP requests are supported by our server
- We will also implement a GET method where we will supply a file requested by the user such as index.html

How it Works

- We start the server by initiating a connection with a client socket. This initiates a new thread from the cache pool.
- The server then sets up all of the I/O streams between the server and the client.

```
ServerSocket serverSock = new ServerSocket(PORT);
actLog.finer("Server running on IP and Port: " + serverSock.toString());

Executor service = Executors.newCachedThreadPool();

while(true){
    try{
        Socket client = serverSock.accept();
        HttpServer temp = new HttpServer(client,actLog,errLog);
        service.execute(temp);
    }
    catch(SocketTimeoutException x){
        errLog.finer("Socket timed out: "+x);
    }
```

Client Connection To Server

```
ublic static void httpserver(Socket connect){
    actLog.finer("Connected to Client: "+connect.toString());
    BufferedReader in = null;
    PrintWriter out = null;
    BufferedOutputStream fileOut = null;

    try{
        in = new BufferedReader(new InputStreamReader(connect.getInputStream()));
        out = new PrintWriter(connect.getOutputStream());
        fileOut = new BufferedOutputStream(connect.getOutputStream());
        String fileRequested = null;
        String httpRequestType = null;
        actLog.finer("Finish I/O Connections");
```

Establishing In and Out Streams

How it Works (Cont.)

- Next the request line and headers are parsed to find out information about the request
- We are specifically looking at the request line, connection, and then content type and length for POST requests.

```
//Request Line Parsing
String requestLine = in.readLine();
System.out.println("Request Line: "+ requestLine);
StringTokenizer requestLineTokenizer = new StringTokenizer(requestLine);
httpRequestType = requestLineTokenizer.nextToken().toUpperCase();
fileRequested = requestLineTokenizer.nextToken().toLowerCase();
actLog.finer("Request Parsed");

//This parses for the keep alive header and ends if needs to
String temp = ".";

//Loops through all the headers extracting key information we are lookin
while(!(temp.equals(""))){
    temp = in.readLine();
    System.out.println(temp);
    StringTokenizer headParse;
    if(temp.startsWith("Connection:")){
        headParse = new StringTokenizer(temp);
        headParse.nextToken();
        connectKeepAlive = headParse.nextToken();
    }
    if(temp.startsWith("Content-Type:")){
        headParse = new StringTokenizer(temp);
        headParse.nextToken();
        contentType = headParse.nextToken();
    }
    if(temp.startsWith("Content-Length:")){
        headParse = new StringTokenizer(temp);
        headParse.nextToken();
        contentLength = Integer.parseInt(headParse.nextToken());
    }
}
```

How it Works (Cont.)

- Depending on what is sent, the server responds appropriately either sending HTML back or running a script and returning the input or whatever is requested.

- Afterwards the client's connection is either shut off or continued based off what was stated in the connection header.

```
out.println("HTTP/1.1 200 OK");
out.println("Server: TEST");
out.println("Date: "+new Date());
out.println("Content-type:" + fileType);
out.println("Content-length: "+fileLength);
out.print("\r\n\r\n");
//out.println();
out.flush();

if(httpRequestType.equals("GET")){
    fileOut.write(fileData,0,fileLength);
    fileOut.flush();
    actLog.finer("GET Request Returned");
}
else{ actLog.finer("HEAD Request Returned"); }
```

```
if(httpRequestType.equalsIgnoreCase("GET") || httpRequestType.equalsIgnoreCase("HEAD")){
```

Fin