COMP416: Computer Networks Project 1 Report
NFTNet: CoinGecko-API Based Application Layer Protocol

Çisem Özden, 69707

1. Description

The project aims the implementation of CoinGecko API-based application layer protocol based on the client/server model. In the scope of the project, there are basically 3 components which are the NFTNetServer, CoinGecko-API, and the client(s). With this project, information of a specific NFT or NFTs exchange is realized between the server and the client(s). The logic is the following: first, the NFTNetServer is initiated. After that, the client(s) connect to the server and make a request following the protocol. Then, the server connects to the CoinGecko-API and requests the desired data. Finally, the server redirects the obtained data to the client and the information exchange is completed. The implementation allows multiple clients to connect to the server, meaning multi-threading is achieved. Interactions between the server and the clients are occurred according to the protocol defined.

2. Protocol Design & Specifications

The protocol followed in the project is inspired by the HTTP protocol. After the connection is established between the server and a client, the client is asked to input a request with the prompt of "`Enter a request for the NFTNet server`". The client must follow the request format to be able to get a proper response. The request input must compose of 2 parts; first one specifying the request method (which is "GET" for this project, since there is no other methods), the second one specifying either the id of a particular NFT, or just the "list" keyword to obtain the whole list of NFTs. Examples of valid requests can be found below:

```
GET squiggly
GET list
```

After the client sends her request, the server gets it and acknowledges it by displaying a statement like below:

```
Client /127.0.0.1:53810 requested :  GET abc
```

The response from the server side varies according to the validity of the request, or the result of the request made to the API. However, the response always contains a line displaying the status of the request, and a body for the content only if a proper result is obtained from the API. After the server sends the response, the client acknowledges that and displays the response right after the following line: "`Response from NFTNetServer:`". The status codes are inspired by the HTTP protocol, which are "`200 OK`", "`404 Not Found`", "`405 Method Not Allowed`", and "`429 Too Many Requests`". All types of responses from possible scenarios can be found below:

a. If the request came from the client is valid, and in the form of "`GET list`": A response composed of "`200 OK`" in the first line, and the names and IDs of all NFTs listed line by line is sent to the client.
b. If the request came from the client is valid, and in the form of "`GET <id of an NFT>`":
b.1. If there is no NFT associated with the specified id in the CoinGecko: A response composed of "`404 Not Found`" is sent to the client.

b.2. If there is an NFT associated with the specified id in the CoinGecko: A response composed of "`200 OK`" in the first line, and the information of id, name, asset platform id, and floor price belonging to that NFT listed line by line is sent to the client.

c. If the request came from the client is not valid: If the request came from the client is not composed of 2 parts and/or if the first word is something other than "`GET`", then a response stating "`405 Method Not Allowed`" is sent to the client.

d. If the limit of API calls made to the CoinGecko is exceeded: A warning response message stating "`429 Too Many Requests`" is sent to the client.

The connection between a client and the server lasts until one of two scenarios occurs:

a. The client wants to stop the connection: Whenever the client types "`QUIT`", client-side socket of the connection is stopped, and the thread belonging to that client will no longer be active.

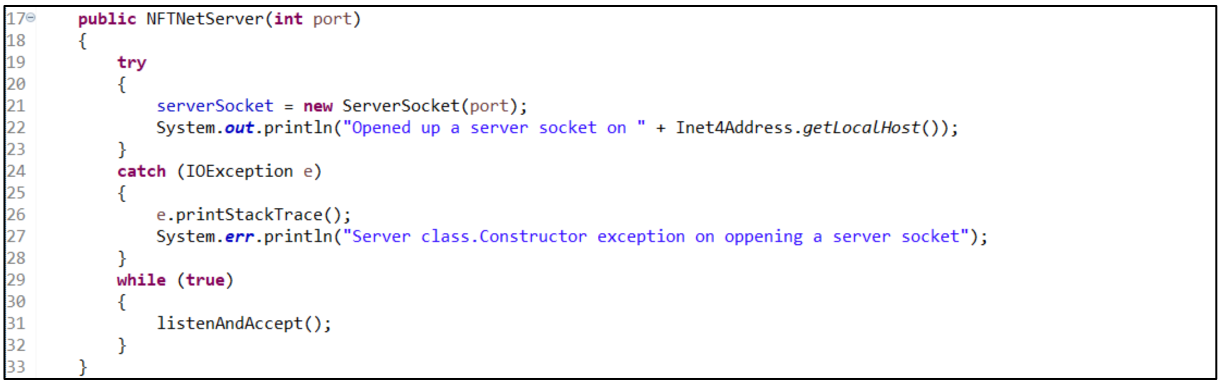
b. The server timeouts the socket of the client: The server timeouts the socket of a client if it does not receive any request within the specified time (10s). The server-side socket of the connection is stopped immediately. When the client tries to input a request after that, it gets a warning saying, "`Connection to NFTNet Server Closed`", and the connection from the client-side is also stopped.

In both scenarios, a statement indicating that the connection of a particular client is stopped along with the client's port number is displayed at the server-side.

Last but not least, thanks to the multi-threading in the code, the NFTNet server listens and accepts the connection coming from multiple servers by assigning thread for each client, and the whole procedure described above is valid for all the clients connected to the server.

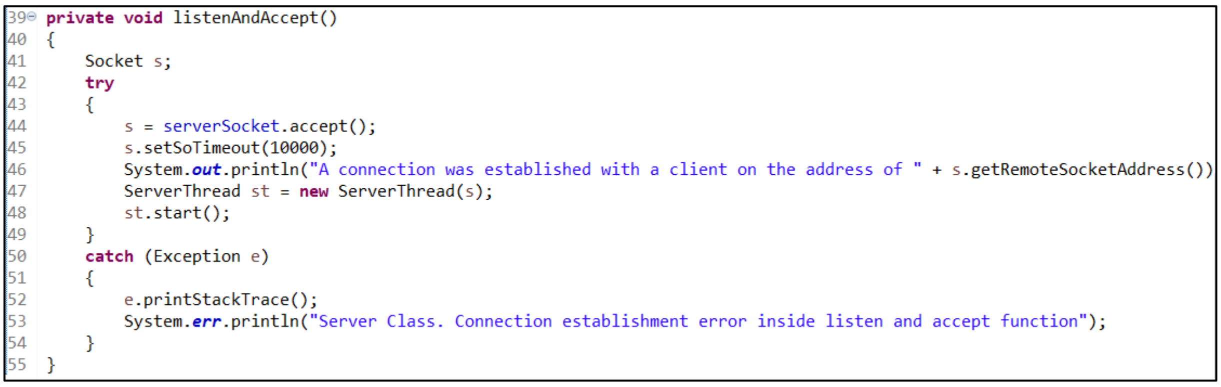
3. Overview of the Server Implementation

The NFTNetServer project is composed of 3 java files which are MultithreadServer.java, NFTNetServer.java, and ServerThread.java. In MultithreadServer, an instance of NFTNetServer is created. Inside NFTNetServer, through the constructor, the welcoming socket is created and the listenAndAccept() method is called constantly as shown below.



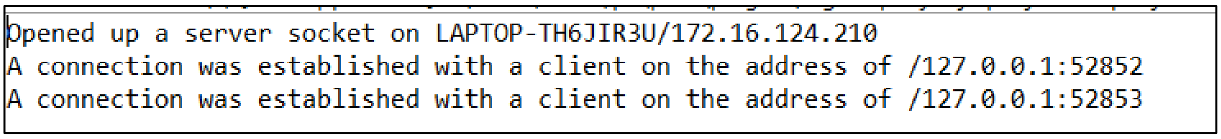
```java
    public NFTNetServer(int port)
    {
        try
        {
            serverSocket = new ServerSocket(port);
            System.out.println("Opened up a server socket on " + Inet4Address.getLocalHost());
        }
        catch (IOException e)
        {
            e.printStackTrace();
            System.err.println("Server class.Constructor exception on oppening a server socket");
        }
        while (true)
        {
            listenAndAccept();
        }
    }
```

With the help of listenAndAccept method, the server can accept the connection requests coming from the clients, as well as creating new threads and assigns them to each client as shown in figure 2. Moreover, timeout time for the sockets is set here for each thread associated with clients. With that, when the time is up, the server-side socket of the connection associated with a certain client stops without touching the welcoming socket.



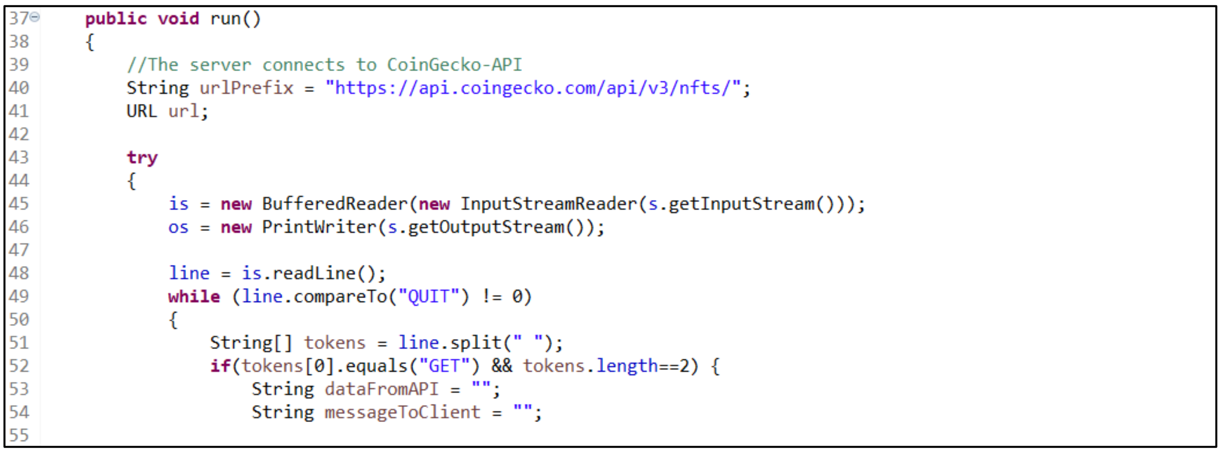
```java
private void listenAndAccept()
{
    Socket s;
    try
    {
        s = serverSocket.accept();
        s.setSoTimeout(10000);
        System.out.println("A connection was established with a client on the address of " + s.getRemoteSocketAddress())
        ServerThread st = new ServerThread(s);
        st.start();
    }
    catch (Exception e)
    {
        e.printStackTrace();
        System.err.println("Server Class. Connection establishment error inside listen and accept function");
    }
}
```

Corresponding result can be found below:



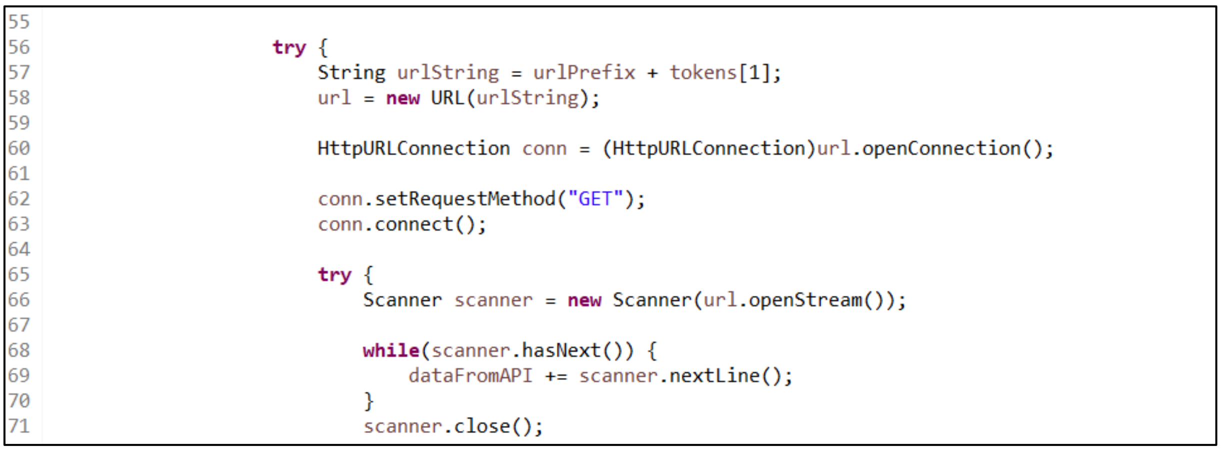
```
Opened up a server socket on LAPTOP-TH6JIR3U/172.16.124.210
A connection was established with a client on the address of /127.0.0.1:52852
A connection was established with a client on the address of /127.0.0.1:52853
```

The whole logic for the server-side runs inside the ServerThread.java. After the reader and the writer for the socket are created, input from the client is read as seen in line 48. As long as the client does not want to end the connection (or it is ended by the set timeout), the communication continues. The if statement seen is responsible for checking the validity of the request as described before in the protocol.



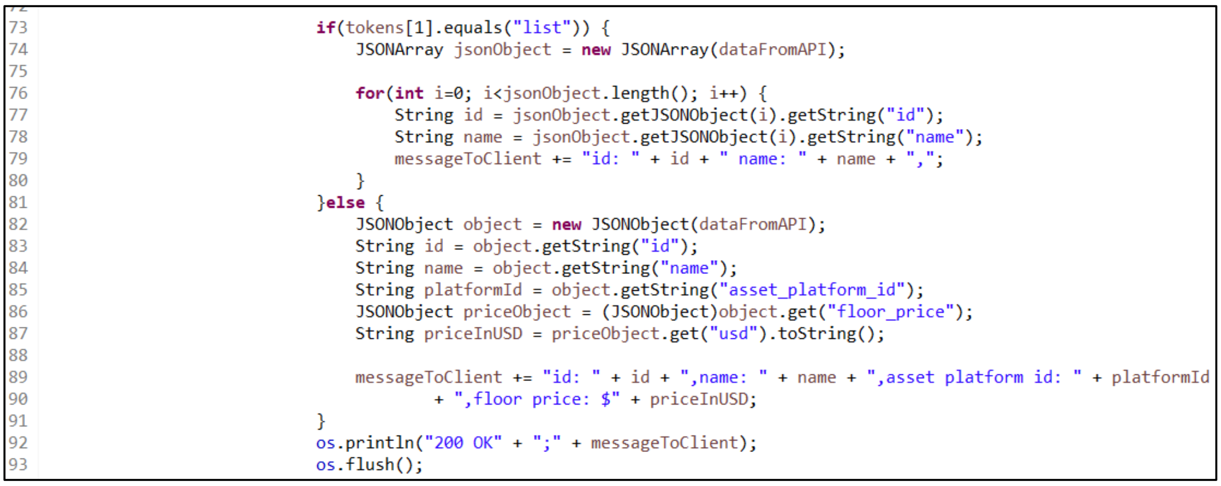
```java
37      public void run()
38      {
39          //The server connects to CoinGecko-API
40          String urlPrefix = "https://api.coingecko.com/api/v3/nfts/";
41          URL url;
42
43          try
44          {
45              is = new BufferedReader(new InputStreamReader(s.getInputStream()));
46              os = new PrintWriter(s.getOutputStream());
47
48              line = is.readLine();
49              while (line.compareTo("QUIT") != 0)
50              {
51                  String[] tokens = line.split(" ");
52                  if(tokens[0].equals("GET") && tokens.length==2) {
53                      String dataFromAPI = "";
54                      String messageToClient = "";
55
```

Next, the server connects to the API by using the url formed using the necessary input taken from the client (lines 57-63). After the connection is achieved, the process of taking the line-by-line result from the API is started as seen in lines 65-71.



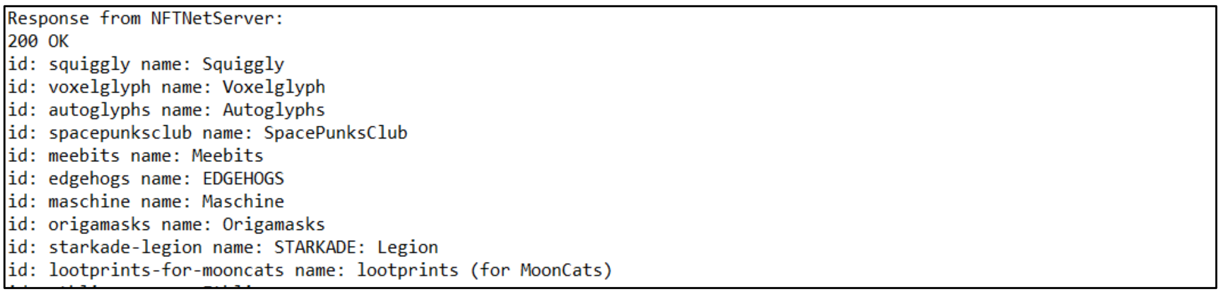
```java
55
56                      try {
57                          String urlString = urlPrefix + tokens[1];
58                          url = new URL(urlString);
59
60                          HttpURLConnection conn = (HttpURLConnection)url.openConnection();
61
62                          conn.setRequestMethod("GET");
63                          conn.connect();
64
65                          try {
66                              Scanner scanner = new Scanner(url.openStream());
67
68                              while(scanner.hasNext()) {
69                                  dataFromAPI += scanner.nextLine();
70                              }
71                              scanner.close();
```

In the below code, if the request is made for the list, then a response composing of the id and the name information belonging to all the NFTs is extracted. Otherwise, i.e., if a specific NFT is requested, then id, name, asset platform id, and floor price fields are extracted only for the NFT specified by the second argument of the request. Finally, the created message is sent to the client along with the success code (line 92).

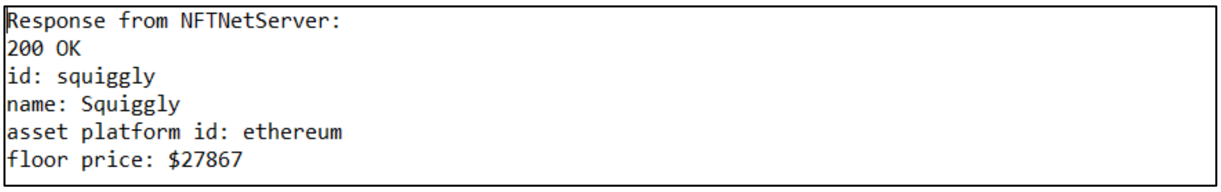


```
73                              if(tokens[1].equals("list")) {
74                                  JSONArray jsonObject = new JSONArray(dataFromAPI);
75
76                                  for(int i=0; i<jsonObject.length(); i++) {
77                                      String id = jsonObject.getJSONObject(i).getString("id");
78                                      String name = jsonObject.getJSONObject(i).getString("name");
79                                      messageToClient += "id: " + id + " name: " + name + ",";
80                                  }
81                              }else {
82                                  JSONObject object = new JSONObject(dataFromAPI);
83                                  String id = object.getString("id");
84                                  String name = object.getString("name");
85                                  String platformId = object.getString("asset_platform_id");
86                                  JSONObject priceObject = (JSONObject)object.get("floor_price");
87                                  String priceInUSD = priceObject.get("usd").toString();
88
89                                  messageToClient += "id: " + id + ",name: " + name + ",asset platform id: " + platformId
90                                          + ",floor price: $" + priceInUSD;
91                              }
92                              os.println("200 OK" + ";" + messageToClient);
93                              os.flush();
```

Example responses created as described can be found below (which is displayed on the client-side):

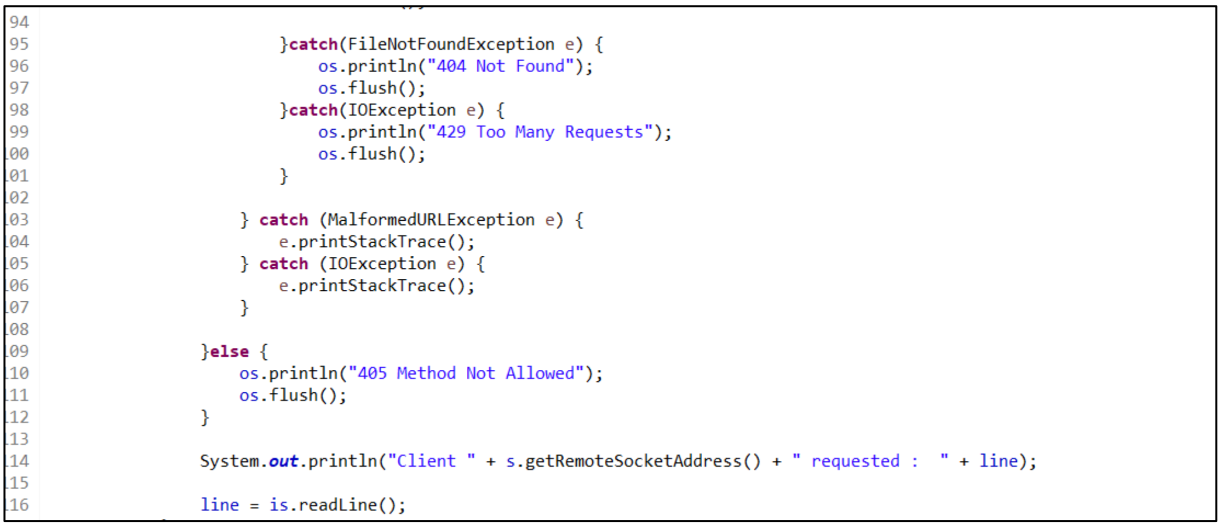


```
Response from NFTNetServer:
200 OK
id: squiggly name: Squiggly
id: voxelglyph name: Voxelglyph
id: autoglyphs name: Autoglyphs
id: spacepunksclub name: SpacePunksClub
id: meebits name: Meebits
id: edgehogs name: EDGEHOGS
id: maschine name: Maschine
id: origamasks name: Origamasks
id: starkade-legion name: STARKADE: Legion
id: lootprints-for-mooncats name: lootprints (for MoonCats)
```



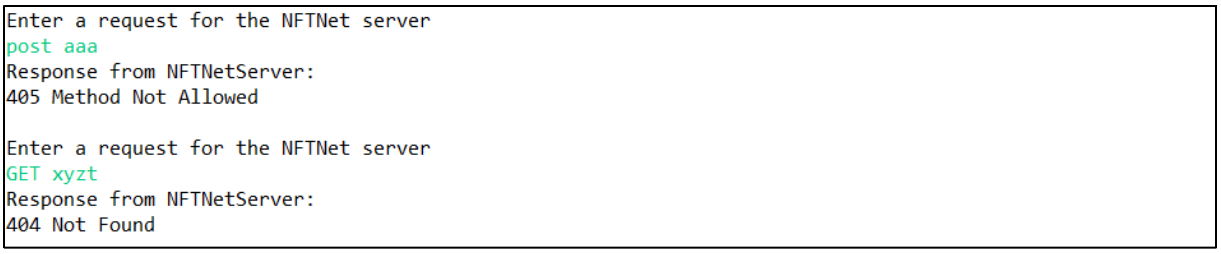
```
Response from NFTNetServer:
200 OK
id: squiggly
name: Squiggly
asset platform id: ethereum
floor price: $27867
```

Necessary exception handling is also shown below in lines 95-112. Corresponding status codes are sent to the client if the message extraction from the API fails for some reason (as explained in protocol design part in detail).

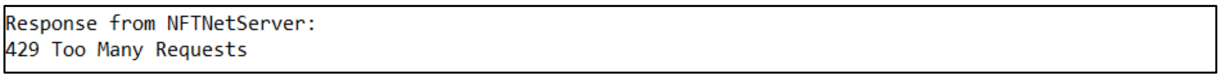


```
                    }catch(FileNotFoundException e) {
                        os.println("404 Not Found");
                        os.flush();
                    }catch(IOException e) {
                        os.println("429 Too Many Requests");
                        os.flush();
                    }

                } catch (MalformedURLException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }

            }else {
                os.println("405 Method Not Allowed");
                os.flush();
            }

            System.out.println("Client " + s.getRemoteSocketAddress() + " requested :  " + line);

            line = is.readLine();
```

Corresponding results can be found below (screenshots are taken from the client-side) (Note that the warning with the status code 429 is obtained from sending the same request from one client to the server more than 50 times in a for loop. This part is commented out in the source code and is only for the testing purpose):



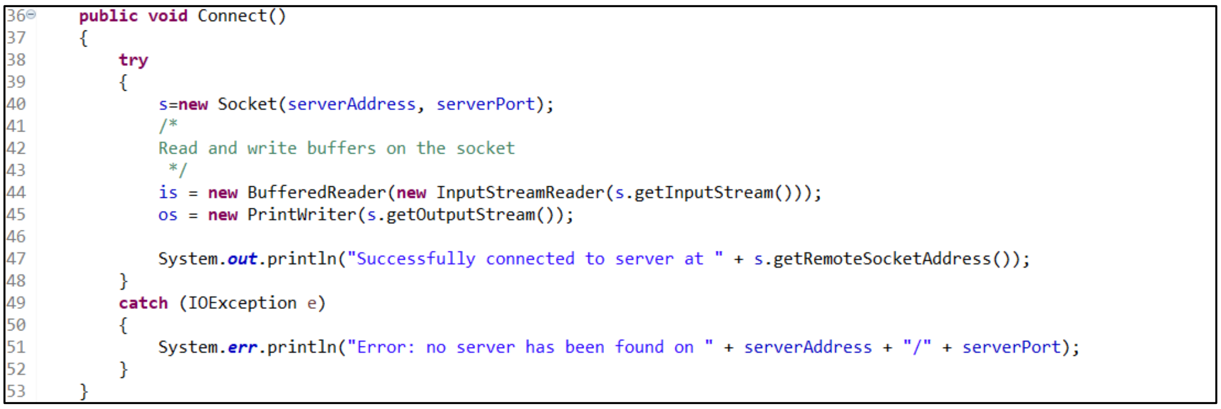
```
Enter a request for the NFTNet server
post aaa
Response from NFTNetServer:
405 Method Not Allowed

Enter a request for the NFTNet server
GET xyzt
Response from NFTNetServer:
404 Not Found
```



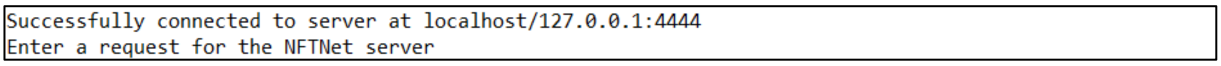
```
Response from NFTNetServer:
429 Too Many Requests
```

4. Overview of the Client Implementation

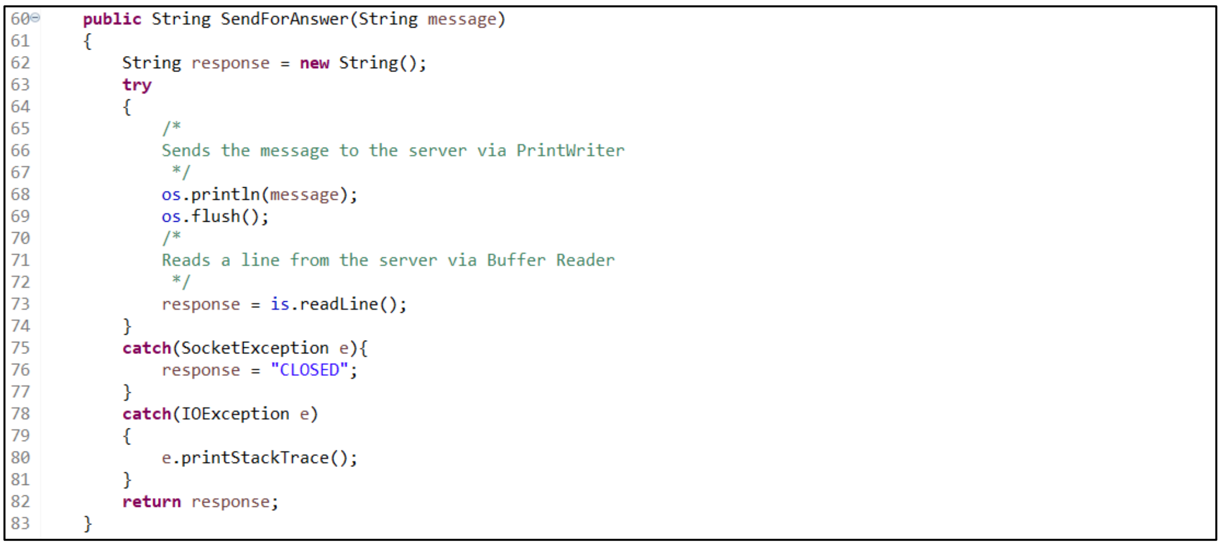
The Client is composed of 2 java files which are ConnectionToServer.java and MultithreadClient.java. ConnectionToServer is used to handle the connection process between the server and the client. In the Connect() method of it, the client creates a socket and connects to the server in line 40, and the relevant information is displayed as shown right below the related code.



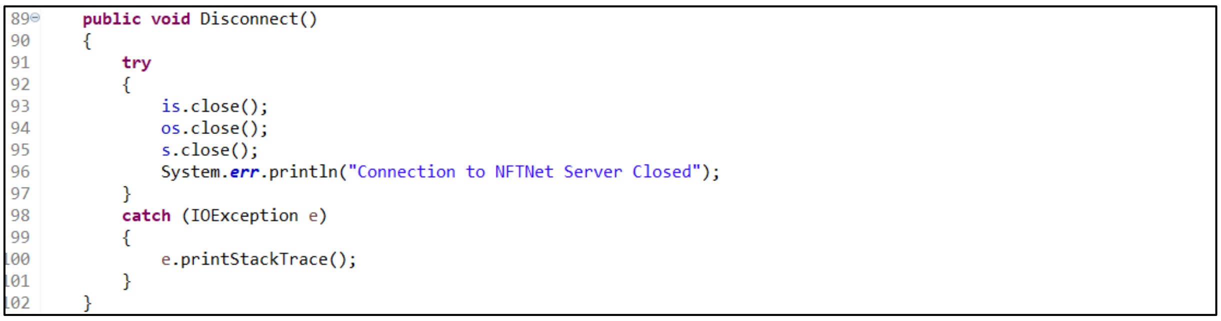
```
36      public void Connect()
37      {
38          try
39          {
40              s=new Socket(serverAddress, serverPort);
41              /*
42              Read and write buffers on the socket
43               */
44              is = new BufferedReader(new InputStreamReader(s.getInputStream()));
45              os = new PrintWriter(s.getOutputStream());
46
47              System.out.println("Successfully connected to server at " + s.getRemoteSocketAddress());
48          }
49          catch (IOException e)
50          {
51              System.err.println("Error: no server has been found on " + serverAddress + "/" + serverPort);
52          }
53      }
```


```
Successfully connected to server at localhost/127.0.0.1:4444
Enter a request for the NFTNet server
```

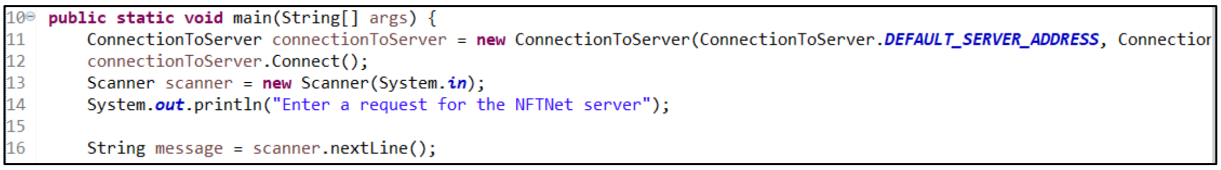
Also, with the help of SendForAnswer() method of ConnectionToServer, the request taken from the user is sent to the server and the response is read write after that as shown below. This method will be used later in the relevant part of the MultithreadClient.java. Here, the exception caught in line 75 is for letting the client know that the server-side socket is closed (which occurs due to timeout set in the project).


```java
60      public String SendForAnswer(String message)
61      {
62          String response = new String();
63          try
64          {
65              /*
66              Sends the message to the server via PrintWriter
67               */
68              os.println(message);
69              os.flush();
70              /*
71              Reads a line from the server via Buffer Reader
72               */
73              response = is.readLine();
74          }
75          catch(SocketException e){
76              response = "CLOSED";
77          }
78          catch(IOException e)
79          {
80              e.printStackTrace();
81          }
82          return response;
83      }
```

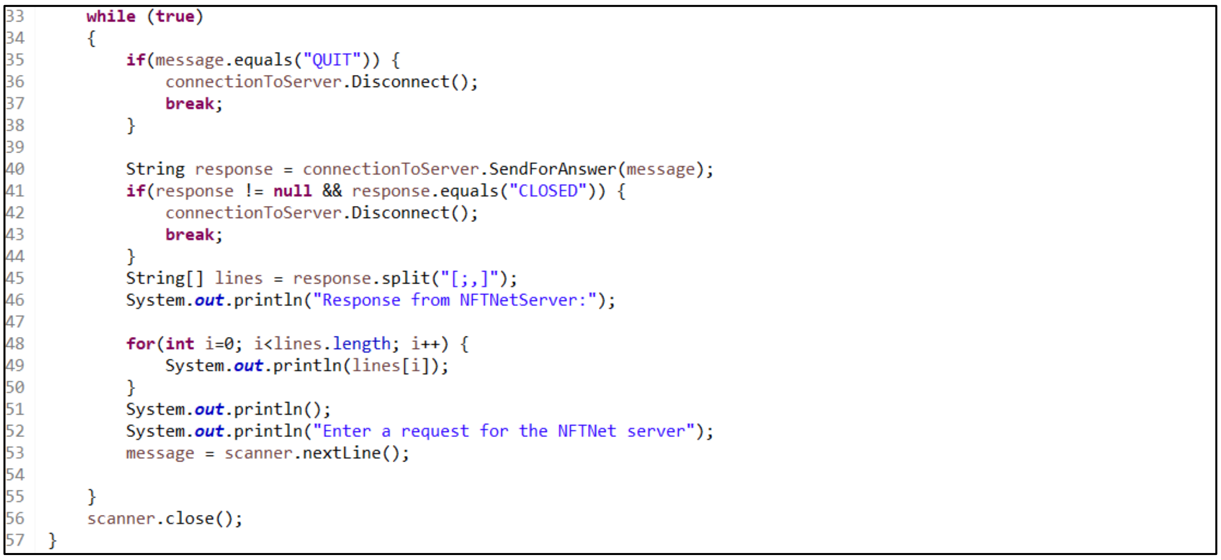
Finally, Disconnect() method shown below is responsible for closing the socket and the streaming operators when it is called.


```java
    public void Disconnect()
    {
        try
        {
            is.close();
            os.close();
            s.close();
            System.err.println("Connection to NFTNet Server Closed");
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
```

Inside the main method of MultithreadClient.java, connection to server is accomplished by creating an instance of ConnectionToServer class, and an input is waited from the user to be sent to the server later on as seen below.

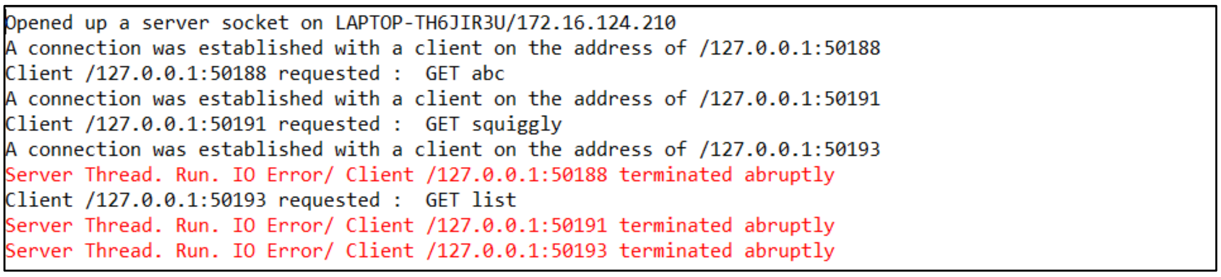

```java
public static void main(String[] args) {
    ConnectionToServer connectionToServer = new ConnectionToServer(ConnectionToServer.DEFAULT_SERVER_ADDRESS, Connectior
    connectionToServer.Connect();
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter a request for the NFTNet server");

    String message = scanner.nextLine();
```

After that, as long as the user does not type QUIT (or the connection is stopped by the timeout set), the response is taken from the server with the help of the SendForAnswer method of ConnectionToServer instance constantly. Inside this method, the request is delivered to the server and the response is obtained as described before. Note that the response coming from the server is just one line, composed of multiple lines separated by the ";" symbol. This method is just for simplifying the process of sending and receiving multiple lines between the server and the client. After the response taken from the server is divided into lines, it is immediately displayed to the user as shown in the overview part of the client-side before. Also, the if statement is for closing the client-side socket when the server-side socket is closed due to timeout.

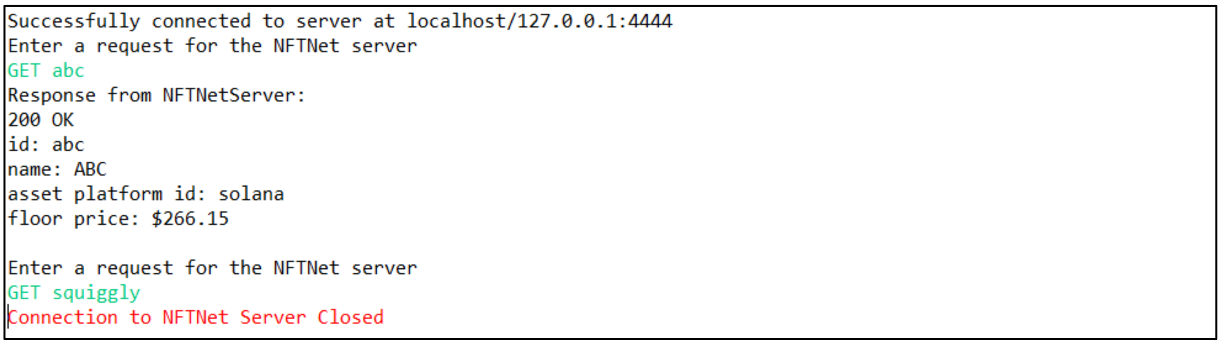


```java
    while (true)
    {
        if(message.equals("QUIT")) {
            connectionToServer.Disconnect();
            break;
        }

        String response = connectionToServer.SendForAnswer(message);
        if(response != null && response.equals("CLOSED")) {
            connectionToServer.Disconnect();
            break;
        }
        String[] lines = response.split("[;,]");
        System.out.println("Response from NFTNetServer:");

        for(int i=0; i<lines.length; i++) {
            System.out.println(lines[i]);
        }
        System.out.println();
        System.out.println("Enter a request for the NFTNet server");
        message = scanner.nextLine();

    }
    scanner.close();
}
```

As final points, the demonstration of multiple clients connected to the server and the result of timeout set can be seen below. First one belongs to the server-side, second one belongs to the client-side.



```
Opened up a server socket on LAPTOP-TH6JIR3U/172.16.124.210
A connection was established with a client on the address of /127.0.0.1:50188
Client /127.0.0.1:50188 requested :  GET abc
A connection was established with a client on the address of /127.0.0.1:50191
Client /127.0.0.1:50191 requested :  GET squiggly
A connection was established with a client on the address of /127.0.0.1:50193
Server Thread. Run. IO Error/ Client /127.0.0.1:50188 terminated abruptly
Client /127.0.0.1:50193 requested :  GET list
Server Thread. Run. IO Error/ Client /127.0.0.1:50191 terminated abruptly
Server Thread. Run. IO Error/ Client /127.0.0.1:50193 terminated abruptly
```



```
Successfully connected to server at localhost/127.0.0.1:4444
Enter a request for the NFTNet server
GET abc
Response from NFTNetServer:
200 OK
id: abc
name: ABC
asset platform id: solana
floor price: $266.15

Enter a request for the NFTNet server
GET squiggly
Connection to NFTNet Server Closed
```