# Comp416 Project 1-2 Report

Burak Yıldırım
72849

Coingecko API is used to get the list of the coins and the prices of specific coins. Timeout is set to 30 seconds.

**Protocol Design:**

[4 bytes, 4 bytes, 4 bytes, 8 bytes]

First 4 bytes are allocated for the messageType which can be 1,2, or 3 where 1 denotes for listing the coins, 2 denotes for getting price of coins, and 3 denotes for termination operation.

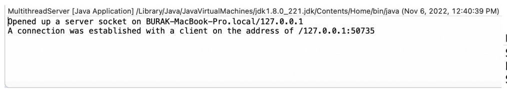
Second 4 bytes are allocted for the N variable which is the length of the data.

Third 4 bytes are allocated for the dataReceived variable which indicates the acknowledge reception of data.

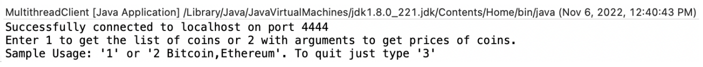
Last 8 bytes are allocated for the data which is the resulting string where it is coin names from client to server and resulting string from server to client. For listing the coins, it becomes the string that shows the list of all coins. For termination operation, it's basically set to empty string.

**Procedure:**

After taking input from the client side, I convert the string into a corresponding protocol as byte array. Then, I send the byte array to the server side. In the server side, I parse the byte array and generate the necessary response for the client. I convert that response to the protocol format and send that to the client. Finally, I parse the protocol in the client side and output the result. All API requests are made within the server side.

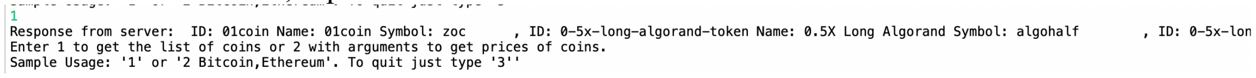


```
MultithreadServer [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java (Nov 6, 2022, 12:40:39 PM)
Opened up a server socket on BURAK-MacBook-Pro.local/127.0.0.1
A connection was established with a client on the address of /127.0.0.1:50735
```



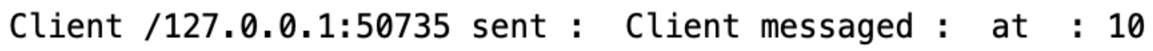
```
MultithreadClient [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java (Nov 6, 2022, 12:40:43 PM)
Successfully connected to localhost on port 4444
Enter 1 to get the list of coins or 2 with arguments to get prices of coins.
Sample Usage: '1' or '2 Bitcoin,Ethereum'. To quit just type '3'
```

Left hand side shows the server side and the right hand side shows the client side. After a successful connection, it prints out the connected socket information.



```
1
Response from server:  ID: 01coin Name: 01coin Symbol: zoc      , ID: 0-5x-long-algorand-token Name: 0.5X Long Algorand Symbol: algohalf       , ID: 0-5x-lon
Enter 1 to get the list of coins or 2 with arguments to get prices of coins.
Sample Usage: '1' or '2 Bitcoin,Ethereum'. To quit just type '3''
```

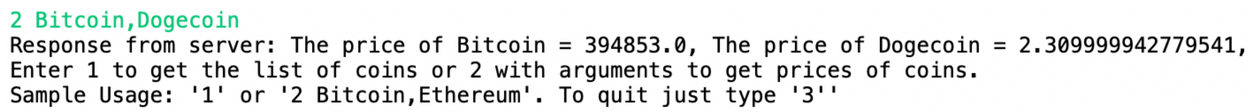
Client-side



```
Client /127.0.0.1:50735 sent :  Client messaged :  at  : 10
```

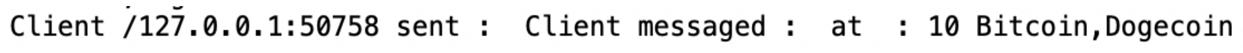
Server-side

In the example above, we can see that client side enters 1 and gets the list of coins as response. Meanwhile, server side displays the message sent by which particular client.



```
2 Bitcoin,Dogecoin
Response from server: The price of Bitcoin = 394853.0, The price of Dogecoin = 2.309999942779541,
Enter 1 to get the list of coins or 2 with arguments to get prices of coins.
Sample Usage: '1' or '2 Bitcoin,Ethereum'. To quit just type '3''
```

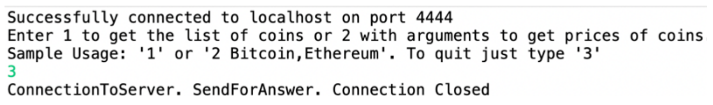
Client-side



```
Client /127.0.0.1:50758 sent :  Client messaged :  at  : 10 Bitcoin,Dogecoin
```

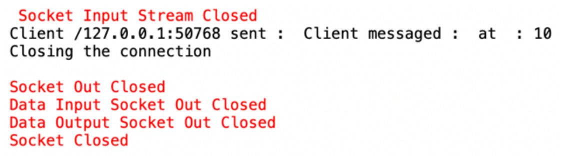
Server-side

In the example above, client-side enters 2 and Bitcoin,Dogecoin to get their prices. The client gets the prices of them as response from the client. Meanwhile, the server displays which client sent that request and the names of the coins.



```
Successfully connected to localhost on port 4444
Enter 1 to get the list of coins or 2 with arguments to get prices of coins
Sample Usage: '1' or '2 Bitcoin,Ethereum'. To quit just type '3'
3
ConnectionToServer. SendForAnswer. Connection Closed
```

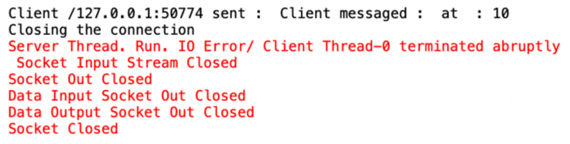
Client-side



```
 Socket Input Stream Closed
Client /127.0.0.1:50768 sent :  Client messaged :  at  : 10
Closing the connection

Socket Out Closed
Data Input Socket Out Closed
Data Output Socket Out Closed
Socket Closed
```

Server-Side

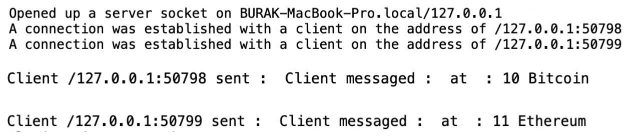
In the example above, client wanted to terminate the connection by entering 3. Then, connections and sockets are closed for both sides.



```
Client /127.0.0.1:50774 sent :  Client messaged :  at  : 10
Closing the connection
Server Thread. Run. IO Error/ Client Thread-0 terminated abruptly
 Socket Input Stream Closed
Socket Out Closed
Data Input Socket Out Closed
Data Output Socket Out Closed
Socket Closed
```

Server-Side

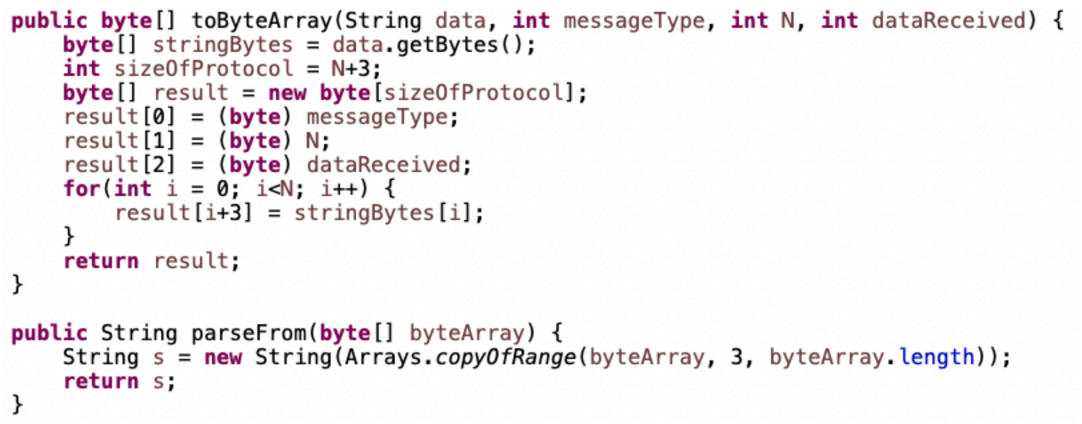
In this example, if none of the clients enters a request after 30 seconds which is the timeout limit that I set server sides closes the connections and sockets.



```
Opened up a server socket on BURAK-MacBook-Pro.local/127.0.0.1
A connection was established with a client on the address of /127.0.0.1:50798
A connection was established with a client on the address of /127.0.0.1:50799

Client /127.0.0.1:50798 sent :  Client messaged :  at  : 10 Bitcoin

Client /127.0.0.1:50799 sent :  Client messaged :  at  : 11 Ethereum
```

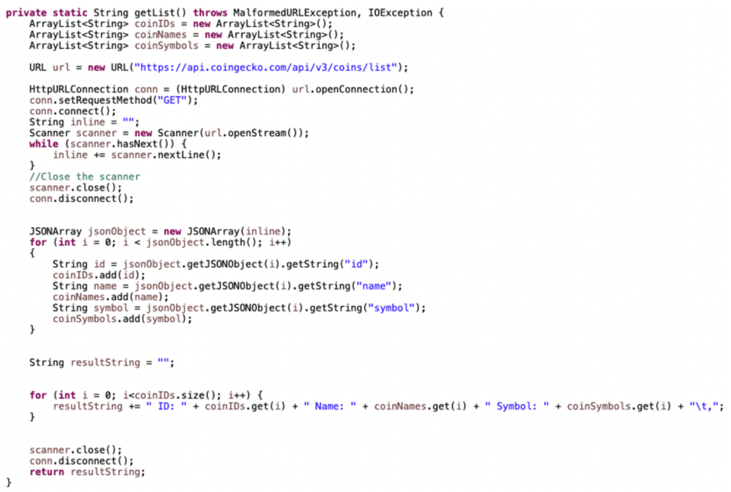
Server - side

Multiple clients can connect to the server. Server will display their addresses and their requests as shown above.



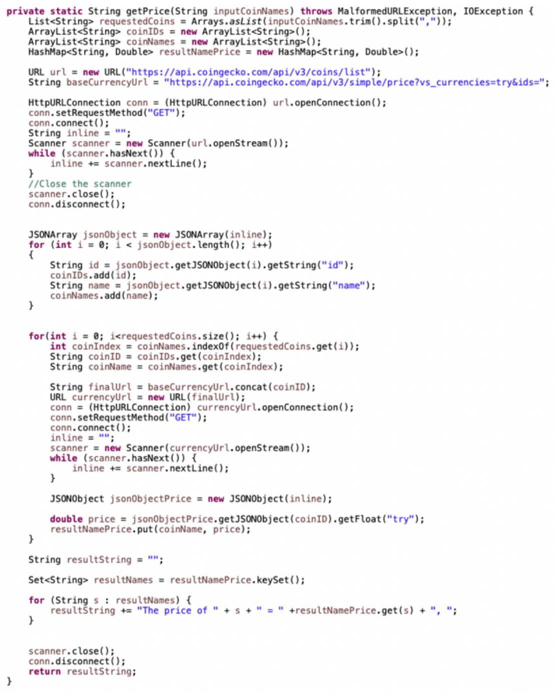
```java
public byte[] toByteArray(String data, int messageType, int N, int dataReceived) {
    byte[] stringBytes = data.getBytes();
    int sizeOfProtocol = N+3;
    byte[] result = new byte[sizeOfProtocol];
    result[0] = (byte) messageType;
    result[1] = (byte) N;
    result[2] = (byte) dataReceived;
    for(int i = 0; i<N; i++) {
        result[i+3] = stringBytes[i];
    }
    return result;
}

public String parseFrom(byte[] byteArray) {
    String s = new String(Arrays.copyOfRange(byteArray, 3, byteArray.length));
    return s;
}
```

Both server and client sides have the toByteArray and parseFrom methods to convert the message into the protocol format and retrieve necessary information from the protocol by using parseFrom.



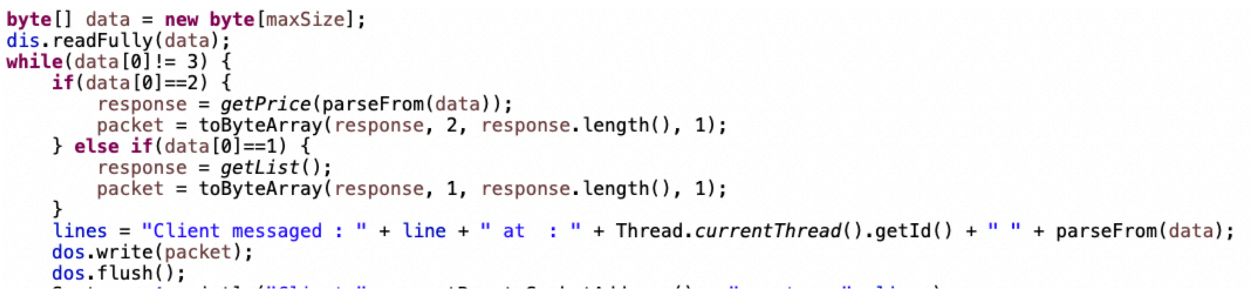
```java
private static String getList() throws MalformedURLException, IOException {
    ArrayList<String> coinIDs = new ArrayList<String>();
    ArrayList<String> coinNames = new ArrayList<String>();
    ArrayList<String> coinSymbols = new ArrayList<String>();

    URL url = new URL("https://api.coingecko.com/api/v3/coins/list");

    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("GET");
    conn.connect();
    String inline = "";
    Scanner scanner = new Scanner(url.openStream());
    while (scanner.hasNext()) {
        inline += scanner.nextLine();
    }
    //Close the scanner
    scanner.close();
    conn.disconnect();


    JSONArray jsonObject = new JSONArray(inline);
    for (int i = 0; i < jsonObject.length(); i++)
    {
        String id = jsonObject.getJSONObject(i).getString("id");
        coinIDs.add(id);
        String name = jsonObject.getJSONObject(i).getString("name");
        coinNames.add(name);
        String symbol = jsonObject.getJSONObject(i).getString("symbol");
        coinSymbols.add(symbol);
    }


    String resultString = "";


    for (int i = 0; i<coinIDs.size(); i++) {
        resultString += " ID: " + coinIDs.get(i) + " Name: " + coinNames.get(i) + " Symbol: " + coinSymbols.get(i) + "\t,";
    }


    scanner.close();
    conn.disconnect();
    return resultString;
}
```

Server have getList() and getPrice() methods for getting list of coins and prices of certain coins from the coingecko API. A JSON array returns from the API. I turned it into a string format to display it to the client.



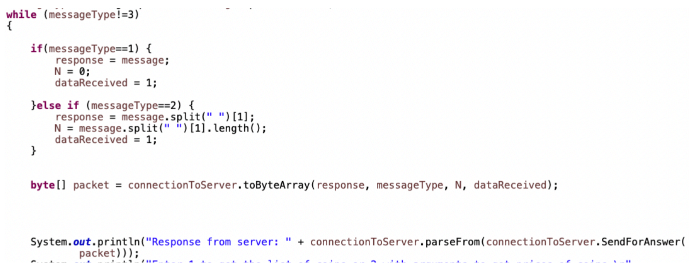
```java
private static String getPrice(String inputCoinNames) throws MalformedURLException, IOException {
    List<String> requestedCoins = Arrays.asList(inputCoinNames.trim().split(","));
    ArrayList<String> coinIDs = new ArrayList<String>();
    ArrayList<String> coinNames = new ArrayList<String>();
    HashMap<String, Double> resultNamePrice = new HashMap<String, Double>();

    URL url = new URL("https://api.coingecko.com/api/v3/coins/list");
    String baseCurrencyUrl = "https://api.coingecko.com/api/v3/simple/price?vs_currencies=try&ids=";

    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("GET");
    conn.connect();
    String inline = "";
    Scanner scanner = new Scanner(url.openStream());
    while (scanner.hasNext()) {
        inline += scanner.nextLine();
    }
    //Close the scanner
    scanner.close();
    conn.disconnect();


    JSONArray jsonObject = new JSONArray(inline);
    for (int i = 0; i < jsonObject.length(); i++)
    {
        String id = jsonObject.getJSONObject(i).getString("id");
        coinIDs.add(id);
        String name = jsonObject.getJSONObject(i).getString("name");
        coinNames.add(name);
    }


    for(int i = 0; i<requestedCoins.size(); i++) {
        int coinIndex = coinNames.indexOf(requestedCoins.get(i));
        String coinID = coinIDs.get(coinIndex);
        String coinName = coinNames.get(coinIndex);

        String finalUrl = baseCurrencyUrl.concat(coinID);
        URL currencyUrl = new URL(finalUrl);
        conn = (HttpURLConnection) currencyUrl.openConnection();
        conn.setRequestMethod("GET");
        conn.connect();
        inline = "";
        scanner = new Scanner(currencyUrl.openStream());
        while (scanner.hasNext()) {
            inline += scanner.nextLine();
        }

        JSONObject jsonObjectPrice = new JSONObject(inline);

        double price = jsonObjectPrice.getJSONObject(coinID).getFloat("try");
        resultNamePrice.put(coinName, price);
    }

    String resultString = "";

    Set<String> resultNames = resultNamePrice.keySet();

    for (String s : resultNames) {
        resultString += "The price of " + s + " = " +resultNamePrice.get(s) + ", ";
    }


    scanner.close();
    conn.disconnect();
    return resultString;
}
```

getPrice() method takes inputCoinNames as argument such as Bitcoin,Ethereum then it stores comma-seperated coin names as requestedCoins. Then, it finds their id by looking at the all coins with names and sends another request to the API to get the price of the coins.



```java
byte[] data = new byte[maxSize];
dis.readFully(data);
while(data[0]!= 3) {
    if(data[0]==2) {
        response = getPrice(parseFrom(data));
        packet = toByteArray(response, 2, response.length(), 1);
    } else if(data[0]==1) {
        response = getList();
        packet = toByteArray(response, 1, response.length(), 1);
    }
    lines = "Client messaged : " + line + " at  : " + Thread.currentThread().getId() + " " + parseFrom(data);
    dos.write(packet);
    dos.flush();
```

In the server side, if the messageType equals to 2 which is used to get the price, then corresponding method is called. If it's 3 then getList() method is called. If it equals to 3 which is the termination type, it starts to terminate by closing connections and sockets. Protocol comes from the client, it is processed, it converted into a response protocol and sent to the client.



```java
while (messageType!=3)
{

    if(messageType==1) {
        response = message;
        N = 0;
        dataReceived = 1;

    }else if (messageType==2) {
        response = message.split(" ")[1];
        N = message.split(" ")[1].length();
        dataReceived = 1;
    }


    byte[] packet = connectionToServer.toByteArray(response, messageType, N, dataReceived);



    System.out.println("Response from server: " + connectionToServer.parseFrom(connectionToServer.SendForAnswer(
            packet)));
```

Similar logic applies for the client side as well. Messages are prepared regarding to the messageType. It sends the protocol to the server and gets a response from the server. Then, it parses the retrieved protocol and printed out to the client.