

Thien Tran

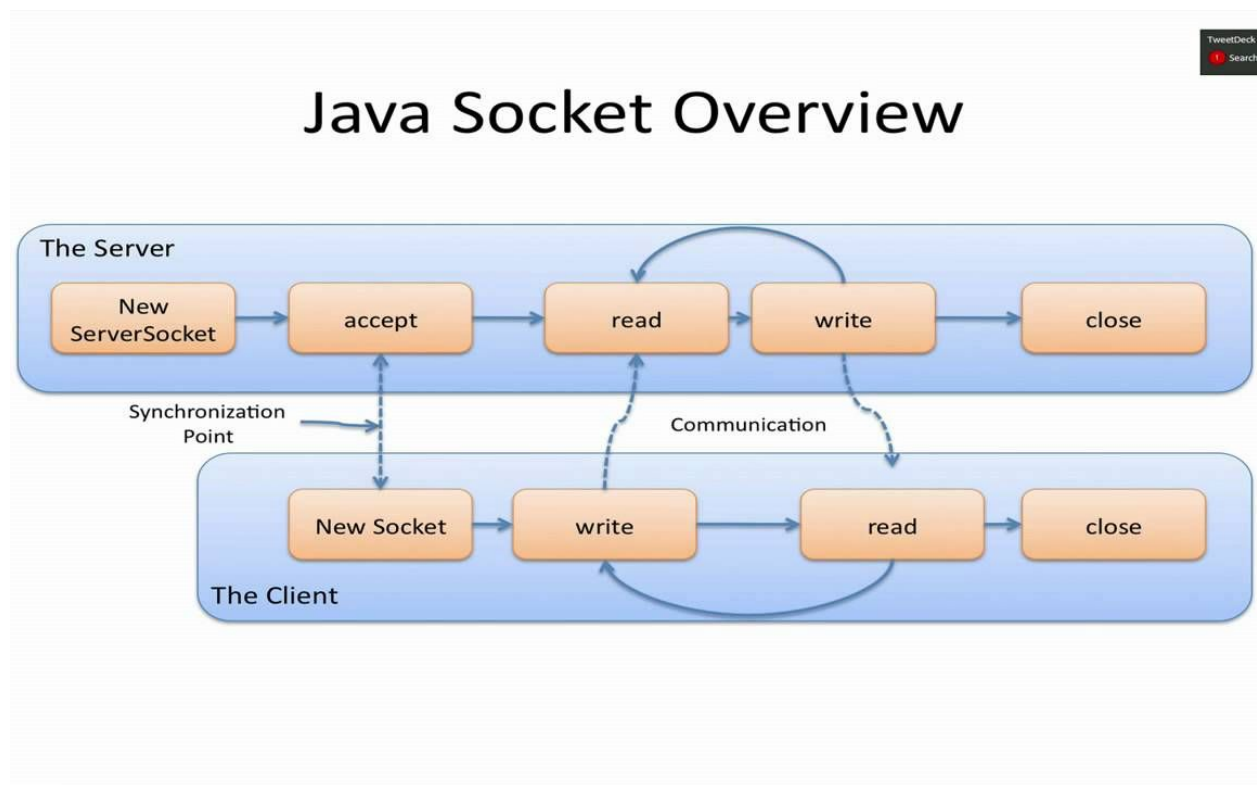
Professor Nur

Networking

3 March 2021

### Programming Assignment 1: Socket Programming

For the first assignment of the Computer Networking class, students must implement a simple group chat application to illustrate the basic foundations of the course. I personally spent about 10.5 hours total - admittedly likely a lot longer than most - attempting this assignment not including the report. I implemented my program using the Java programming language and TCP.



(image source: <https://www.youtube.com/watch?v=aEDV0WlwXTs>)

Although there were times when it was necessary to tackle both the client and server side of the code at the same time, the majority of the time I spent was fixing or implementing specific

portions of either the client or server side one at a time. I used the Apache template and some minor logic from the base code provided by Professor Nur. The first thing that I attempted to do was to implement the client's ability to process packets from the server - something that the base code lacked.

This portion of the code successfully enabled the client to persistently read and print messages from the server. The 'endphrase' in this case is 'Bye'.

```
String line = "";
while (!line.equals(endphrase)) {
    try {
        line = in.readUTF();
        System.out.println(line);
    } catch (IOException e) {
        System.out.println("Error " + e.getMessage());
    }
}
```

Next, I implemented the ability for the client to send data to the server.

```
String inputLine = "";
while (!inputLine.equals(endphrase)) {
    try {
        inputLine = userInput.readLine();
        messageToServer(inputLine);
    } catch (IOException e) {
        System.out.println("Error: " + e);
    }
}
```

```
private void messageToServer(String message) {
    if (!message.equals(endphrase)) {
        //Send outs from Client
        try {
            out.writeUTF(username + ": " + message);
        } catch (IOException e) {
            System.out.println("Error: " + e);
        }
    } else {
        endConnection();
    }
}
```

The data input and output parts of the codes worked fine separately, but there was a problem I encountered where only one portion would be active at a time. For example, the client wouldn't be able to write to the server if it was processing items from the server, and vice-versa the client couldn't read from the server if it was writing items to the server. I fixed this by using multithreading to run both portions of the code in parallel. As shown below.

```

Thread getandsendMessages = new Thread(){
    public void run(){
        String inputLine = "";
        while (!inputLine.equals(endphrase)) {
            try{
                inputLine = userInput.readLine();
                messageToServer(inputLine);
            } catch (IOException e) {
                System.out.println("Error: " + e);
            }
        }
    }
};

Thread receiveandprintMessages = new Thread(){
    public void run(){
        String line = "";
        while (!line.equals(endphrase)) {
            try {
                line = in.readUTF();
                System.out.println(line);
            } catch (IOException e) {
                System.out.println("Error " + e.getMessage());
            }
        }
    }
};

getandsendMessages.start();
receiveandprintMessages.start();

```

---

The bulk of the client-side of the program was completed at this point, so I then started on the server. Similarly to my implementation of the client, I first worked on the ability for the server to persistently receive and process data from the client.

```

String line = "";
while (!line.equals("Bye")) {
    try {
        line = in.readUTF();
        messages.push(line);
        System.out.println(line);
    } catch (IOException e) {
        System.out.println("Error " + e.getMessage());
    }
}

```

Next, I implemented the ability for the server to send messages to the client. There were many strange hitches during this stage of implementation. Some of these hitches included an inability to generally send any kind of data to the client, and an inability to send data unless the message was sent quickly and repeatedly inside of an infinite loop. I fixed this problem respectively by adding a Linked-List to store the messages when the client receives data (see code from previous paragraph) and by adding a 10-millisecond delay to the loop.

```
line = in.readUTF();  
messages.push(line);
```

```
LinkedList<String> messages = new LinkedList<String>();
```

```
while(true){  
    try{TimeUnit.MILLISECONDS.sleep(10);}catch(InterruptedException e){}  
    if(messages.size() > 0){  
        try{  
            out.writeUTF(messages.pop());  
        }catch(IOException e){  
            System.out.println("Error: " + e);  
        }  
    }  
}
```

With the input and output abilities completed, I encountered the problem that I faced earlier with the inability for both portions of the code to run together. Learning from my past mistake, I quickly fixed the problem by making each portion run as a thread.

```

Thread getandsendMessages = new Thread(){
    public void run(){
        String inputLine = "";
        while(!inputLine.equals(endphrase)){
            try{
                inputLine = userInput.readLine();
                messageToServer(inputLine);
            }catch(IOException e){
                System.out.println("Error: " + e);
            }
        }
    }
};

Thread receiveandprintMessages = new Thread(){
    public void run(){
        String line = "";
        while (!line.equals(endphrase)) {
            try {
                line = in.readUTF();
                System.out.println(line);
            } catch (IOException e) {
                System.out.println("Error " + e.getMessage());
            }
        }
    }
};

getandsendMessages.start();
receiveandprintMessages.start();

```

---

With the bulk amount of the significant things done, the program was ready for use!

Attached below are some of the outputs.

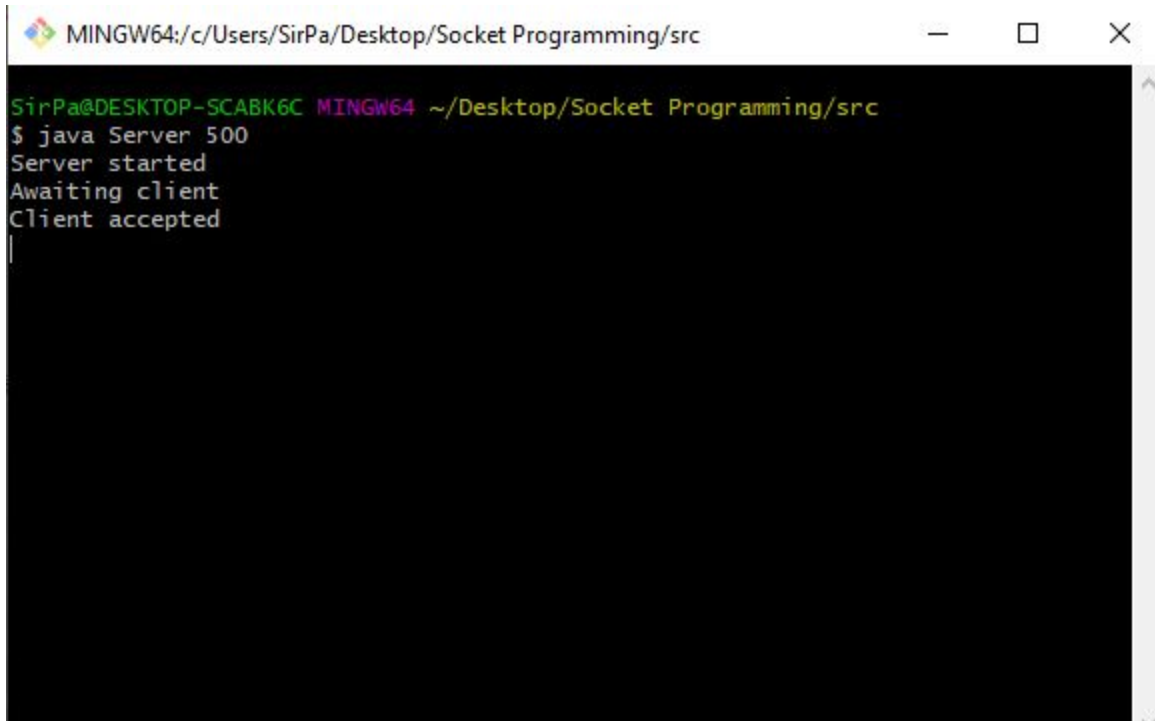
```

SirPa@DESKTOP-SCABK6C MINGW64 ~/Desktop/Socket Programming/src
$ java Server 500
Server started
Awaiting client

```

A terminal window titled "MINGW64:/c/Users/SirPa/Desktop/Socket Programming/src" with standard window controls. The prompt is "SirPa@DESKTOP-SCABK6C MINGW64 ~/Desktop/Socket Programming/src". The user enters "\$ java Client 500", and the output is "Connected" followed by "Enter your username: Thien:".

```
MINGW64:/c/Users/SirPa/Desktop/Socket Programming/src
SirPa@DESKTOP-SCABK6C MINGW64 ~/Desktop/Socket Programming/src
$ java Client 500
Connected
Enter your username: Thien:
```

A terminal window titled "MINGW64:/c/Users/SirPa/Desktop/Socket Programming/src" with standard window controls. The prompt is "SirPa@DESKTOP-SCABK6C MINGW64 ~/Desktop/Socket Programming/src". The user enters "\$ java Server 500", and the output is "Server started", "Awaiting client", and "Client accepted" on separate lines.

```
MINGW64:/c/Users/SirPa/Desktop/Socket Programming/src
SirPa@DESKTOP-SCABK6C MINGW64 ~/Desktop/Socket Programming/src
$ java Server 500
Server started
Awaiting client
Client accepted
|
```

MINGW64:/c/Users/SirPa/Desktop/Socket Programming/src

```
SirPa@DESKTOP-SCABK6C MINGW64 ~/Desktop/Socket Programming/src
$ java Client 500
Connected
Enter your username: Thien
Thien: Server: Welcome Thien
Hello
Thien: Hello
I'm Thien! Hope you're doing good
Thien: I'm Thien! Hope you're doing good
I'm doing fantastic
Thien: I'm doing fantastic
ggs!
Thien: ggs!
|
```

MINGW64:/c/Users/SirPa/Desktop/Socket Programming/src

```
SirPa@DESKTOP-SCABK6C MINGW64 ~/Desktop/Socket Programming/src
$ java Server 500
Server started
Awaiting client
Client accepted

Thien: Server: Welcome Thien
Thien: Hello
Thien: I'm Thien! Hope you're doing good
Thien: I'm doing fantastic
Thien: ggs!
...
```



## Appendix

### Client.java

```
1  import java.net.*;
2  import java.io.*;
3  import java.util.LinkedList;
4
5  public class Client{
6      private final String endphrase = "Bye";
7      private Socket socket = null;
8      private DataInputStream userInput = null;
9      private DataInputStream in = null;
10     private DataOutputStream out = null;
11     private String username;
12
13     public Client(String address, int port){
14         connectToServer(address, port);
15         initializeUsername();
16         startChat();
17     }
18
19     private void startChat(){
20         Thread getandsendMessages = new Thread(){
21             public void run(){
22                 String inputLine = "";
23                 while(!inputLine.equals(endphrase)){
24                     try{
25                         inputLine = userInput.readLine();
26                         messageToServer(inputLine);
27                     } catch (IOException e) {
28                         System.out.println("Error: " + e);
29                     }
30                 }
31             }
32         };
33
34         Thread receiveandprintMessages = new Thread(){
35             public void run(){
36                 String line = "";
37                 while (!line.equals(endphrase)) {
38                     try {
39                         line = in.readUTF();
40                         System.out.println(line);
41                     } catch (IOException e) {
42                         System.out.println("Error " + e.getMessage());
43                     }
44                 }
45             }
46         };
47
48         getandsendMessages.start();
49         receiveandprintMessages.start();
50     }
51 }
```



```

50     }
51
52     private void messageToServer(String message){
53         if (!message.equals(endphrase)) {
54
55             //Send outs from Client
56             try {
57                 out.writeUTF(username + ": " + message);
58             }catch(IOException e){
59                 System.out.println("Error: " + e);
60             }
61         }
62         else{
63             endConnection();
64         }
65     }
66
67     private void connectToServer(String address, int port){
68         try{
69             socket = new Socket(address, port);
70             System.out.println("Connected");
71             userInput = new DataInputStream(System.in);
72             in = new DataInputStream(socket.getInputStream());
73             out = new DataOutputStream(socket.getOutputStream());
74         }catch (Exception e){
75             System.out.println("error " + e.getMessage());
76         }
77     }
78
79     private void initializeUsername(){
80         System.out.printf("Enter your username: ");
81         try{
82             username = userInput.readLine();
83             messageToServer("Server: Welcome " + username);
84         }catch(IOException e){
85             System.out.println("Error: " + e);
86         }
87     }
88
89     private void endConnection(){
90         try {
91             userInput.close();
92             in.close();
93             out.close();
94             socket.close();
95         }catch (IOException i){
96             System.out.println(i);
97         }
98     }
99
100     public static void main(String args[]){
101         int port = Integer.parseInt(args[0]);
102         Client client = new Client("localhost", port);
103     }
104 }

```

## Server.java

```
1  import java.net.*;
2  import java.io.*;
3  import java.util.*;
4  import java.util.concurrent.TimeUnit;
5
6
7  public class Server{
8      public Server(int port) {
9          startChat(port);
10     }
11
12     public void startChat(int port){
13         LinkedList<String> messages = new LinkedList<String>();
14
15
16         try {
17             final ServerSocket server = new ServerSocket(port);
18             System.out.println("Server started");
19             System.out.println("Awaiting client");
20             final Socket socket = server.accept();
21             System.out.println("Client accepted");
22
23             DataInputStream in = new DataInputStream(
24                 socket.getInputStream());
25             DataOutputStream out = new DataOutputStream(
26                 socket.getOutputStream());
27
28             Thread receiveandprintMessages = new Thread(){
29                 public void run(){
30                     String line = "";
31                     while (!line.equals("Bye")) {
32                         try {
33                             line = in.readUTF();
34                             messages.push(line);
35                             System.out.println(line);
36                         } catch (IOException e) {
37                             System.out.println("Error " + e.getMessage());
38                         }
39                     }
40                     System.out.println("Goodbye!");
41                 }
42             };
43
44             Thread getandsendMessages = new Thread(){
45                 public void run(){
46                     while(true){
47                         try{TimeUnit.MILLISECONDS.sleep(10);}catch(InterruptedException e){}
48                         if(messages.size() > 0){
49                             try{
```

```
50         out.writeUTF(messages.pop());
51     } catch (IOException e) {
52         System.out.println("Error: " + e);
53     }
54 }
55 }
56 }
57 };
58
59
60     receiveandprintMessages.start();
61     getandsendMessages.start();
62 } catch (IOException e) {
63     System.out.println("Error " + e.getMessage());
64 }
65 }
66
67 public static void main(String args[]) {
68     int port = Integer.parseInt(args[0]);
69     Server server = new Server(port);
70
71 }
72 }
```