Mini-Project 1 – Simple Chat Service

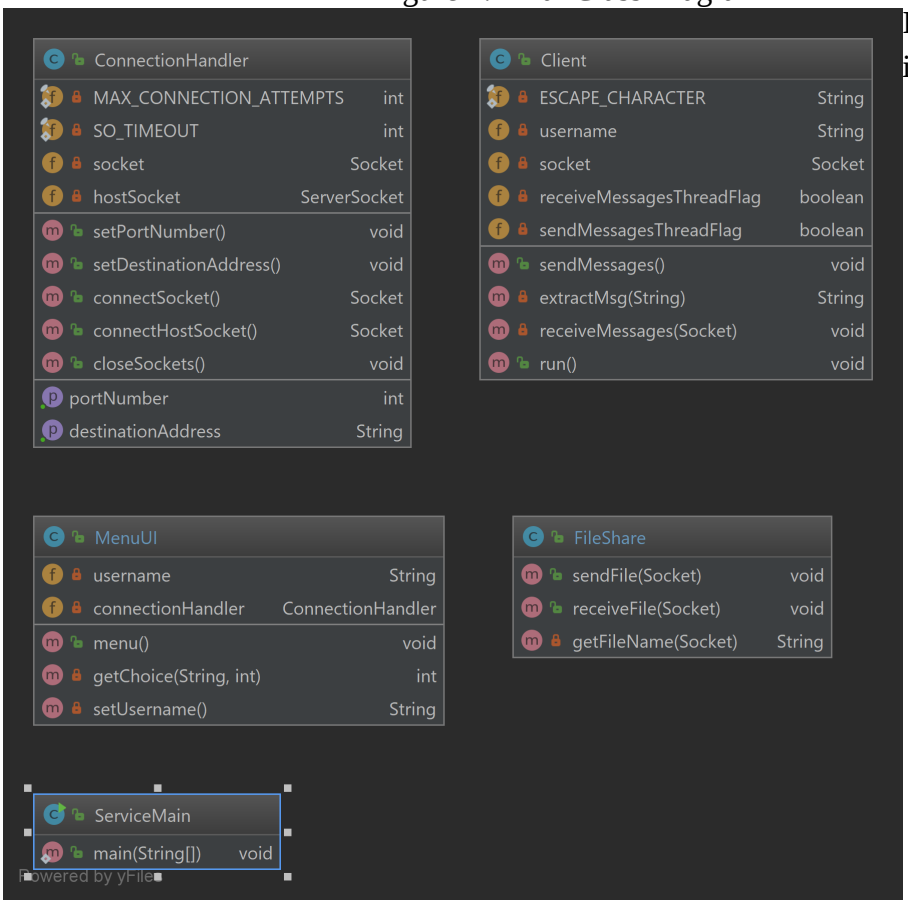
Overview

The aim of this practical was to create a program to exchange text messages between two computers using TCP. In my implementation of this program I completed the following features:

- The application can connect to another machine and exchange chat messages using TCP
- If a connection to a specified machine fails a number of times, it can host a connection.
- The chat can be closed as desired.
- The user can choose to host the connection instead.
- Provides a simple Text-based UI.
- The application can deal with abnormal connection terminations.
- It can send and receive files of arbitrary type.

Design

Figure 1. Final Class Diagram

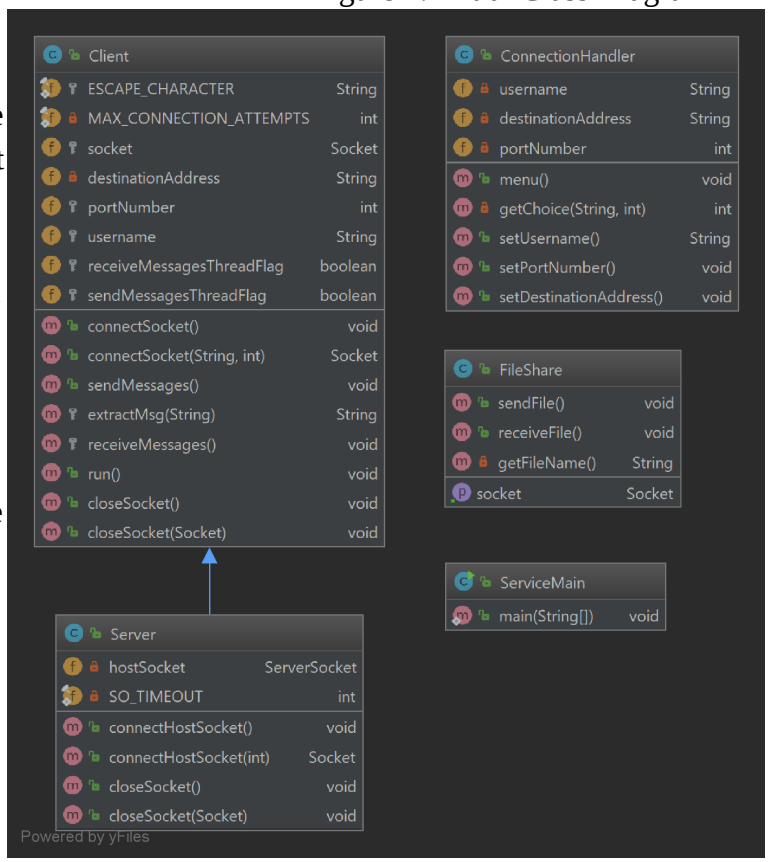


I decided to split my application into four primary classes:

- The MenuUI class is meant to be the point of contact for the user.
- The Client class provides the implementation for messaging services. It writes to, and receives messages from a given socket.
- The ConnectionHandler Class provides an object that connects to other users and gives that connection to the Client program in order for it to send and receive messages.

- The FileShare class represents the filesharing extension. It only requires a socket to send/receive files.

Figure 2. Initial Class Diagram



Initially I considered having the server be a subclass of the client. This was because the only difference between the two would be that the server would host a connection whereas the client would connect to a specified address. I ultimately changed my mind because in order for my FileShare class to work I would have to pass it a socket to use. This would require either adding extra methods to the Client and Server classes just for use by FileShare (Figure 2 – connectSocket(String,Int) and connectHostSocket(int)) or have it extend the Server class and gain many unnecessary methods and properies.

I ultimately chose Figure 1 as it involved the least duplication of data or inheritance of unnecessary methods or attributes.

In order to send and receive files I had a choice between synchronous or asynchronous communication. This meant either implementing multi-threading or using queues in order to send and receive messages.

Using queues would potentially be simpler, as there wouldn't wouldn't be as many bugs or difficulties caused by concurrency. On the other hand, a queue would have to block while waiting for input from the user. This would mean that the user doesn't get the messages in real time if multiple messages are received by the program while the user is typing. Because I valued messages being received as they were sent, I chose to use multi-threading in my application.

Implementation

In my implementation the ConnectionHandler class handles all access to and manipulation of Sockets (and ServerSockets). This was to make it easier to control when and were Sockets were opened and closed. However for the Client class I had to give it its own Socket attribute because in order to implement Runnable (and use multi-threading) I would have to make a zero argument method called run(). The run() method would contain the receiveMessages method which requires a socket to be passed into it, so the only way to this would be to have a global variable Socket which it could access. I do not think this should be an issue however as because the Socket object is passed by reference, when it is closed by the ConnectionHandler it should also be closed in the Client.

While it makes not practical difference, I initially chose to use the Runnable interface instead of extending thread because it would allow the Client class to still extend other classes if it needed to, and it could still implement more interfaces.

I made the InvalidSocketAddressException and ClientHasNotConnectionException so that when a user tries and fails to connect to a Socket, they would have better feed back on exactly what exactly was the problem preventing them from connecting.

If a SocketTimeoutException occurs when using ServerSocket.accept, the socket has to be closed afterwards otherwise the connectHostSocket method will always cause that exception, everytime it is called thereafter.

The messaging protocol I used was very simple. I would add the username and ‘:’ as a suffix to the message before it was sent. This was so that the receiving individual would be able to easily differentiate between messages they sent and messages they received as they scroll up the window.

When closing a connection, either due to the escape character being sent or abnormal loss of connection, it is necessary to ensure that both the sendMessages thread and receiveMessages thread have finished. To do this I created two flag attributes, one for each thread, so that each thread could check if the other had finished running and stop running if other had stopped.

I chose to use DataIOStreams instead of BufferedIOStreams to send files as I found them simpler to use. This was because before I sent the filedata over I would first send the length of the filename, filename and length of the file (in that order). Using DataIOStreams, it was easy to send each piece of information as a different data type (Int, Chars, Long, Bytes). This meant at the receiving end I could use a different read method to read each piece of information in turn instead of writing a method that would calculate the number of bytes a BufferedReader would have to read or creating a protocol where the host sends confirmation messages after each message was sent. However given more time I would have utilized a BufferedIOStreams as they are significantly faster at reading and writing, which would be especially useful if the users wish to send larger files.

Testing

Test 1 Command line Arguments

Test Brief: Program should start successfully, whether or not there is a commandline argument with a destination address.

Input: a. No Command Line Arguments

b. java ServiceMain js395.host.cs.st-andrews.ac.uk

Expected Results: a. Program starts successfully, destination not set

b. Program starts successfully, destination correctly set to my host address.

Results: Successful. a. Figure 3 b. Figure 4

Figure 3. Test 1a

Figure 4. Test 1b

Test 2 Change Destination Address and Port Number

Test Brief: The user should be able to change the current destination.

Input: Initial Destination/Port Number: Not set, Use the MenuUI to switch to js395.host.cs.st-andrews.ac.uk and 38516.

Expected Result: Current Destination should change from ‘Not Set’ to ‘js395.host.cs.st-andrews.ac.uk’

Result: Successful.

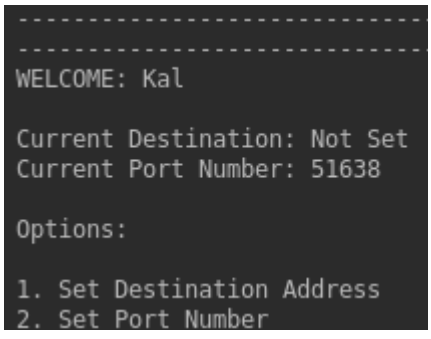


Figure 5a. Before

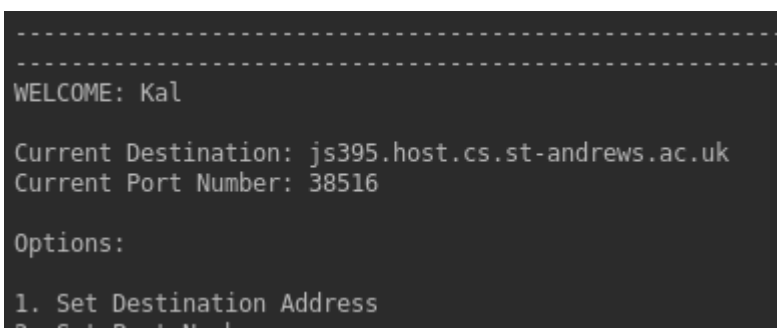


Figure 5b. After

Test 3 Connect to/ Host Another User

Test Brief: Two users should be able to connect to one another and exchange messages.

Input: One terminal will ssh onto js395.host.cs.st-andrews.ac.uk and will host the connection. They will then exchange a simple conversation.

Expected Result: Before Connection each terminal should have a waiting message. Then as messages are sent, Messages from the other terminal should have a username attached to the message.

Figure 6a. Host side

```
Waiting for client to connect...
Client found....
You may now enter your messages...
Client: Hello World!
The Swift kitsune jumped over the tortle
```

Figure 6b. Client side

```
Searching for server...
Server found...
You may now enter your messages...
Hello World!
Host: The Swift kitsune jumped over the tortle
```

Result: Successful. Figure 6a + 6b.

Test 4 – ServerSocket Timeout

Test Brief: After 10 seconds of waiting the host should return to menu, and be allowed to try to try to connect again afterwards.

Input: Attempt to Host a user, wait for 10 seconds, after timeout, try again.

Expected Result: Application should wait 10 seconds before returning to menu. It should allow the user to try and host again after.

Result: Successful. Socket timeout took 10 seconds and waited again on second attempt.

Figure 7. SocketTimeout

```
Waiting for client to connect...
Server Socket Timed out...
Returning to menu...

--------------------------------
--------------------------------
WELCOME: Syl

Current Destination: Not Set
Current Port Number: 51638

Options:

1. Set Destination Address
2. Set Port Number
3. Connect to another user
4. Host another user
5. Send a file.
6. Receive a file
7. Quit


Choose an Option: 4

--------------------------------
--------------------------------

Waiting for client to connect...
Server Socket Timed out...
```

Test 5 Repeat Connect

Test Brief: Application should repeatedly try to connect a specified number of times if (initial connection doesn't work) before returning to menu.

Input: Destination set to j395 (not an address)

Expected Result: Application retries a 4 times before returning to menu.

Result: Successful.

Figure 8. Repeated connection attempts

```
Searching for server...
Server not found... Retrying...
Searching for server...
Server not found... Retrying...
Searching for server...
Server not found... Retrying...
Searching for server...
Server not found... Retrying...
Invalid Address, please check your destination before trying again...

------------------------------------------------------------------
------------------------------------------------------------------
WELCOME: Dalinar

Current Destination: js395
```

Test 6 Normal Connection Termination

Test Brief: When the escape character is sent both Host and Client should close and return to menu.

Input: a. “q” sent from Host.

b. “q” sent from Client.

Expected Result: In both cases, both applications should return to menu.

Result: Successful. Test 6a – Figure 7a + b, Test 6b – Figure 7c + d.

Figure 7a. Client

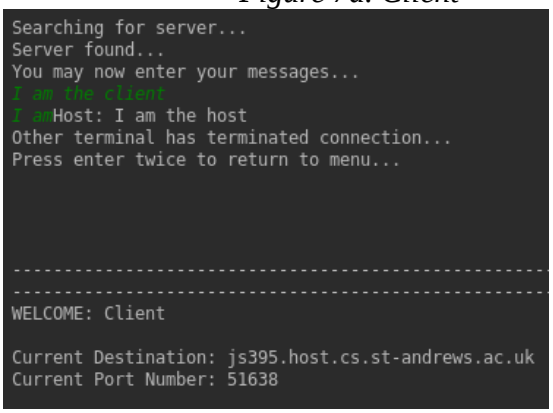


Figure 7b. Host

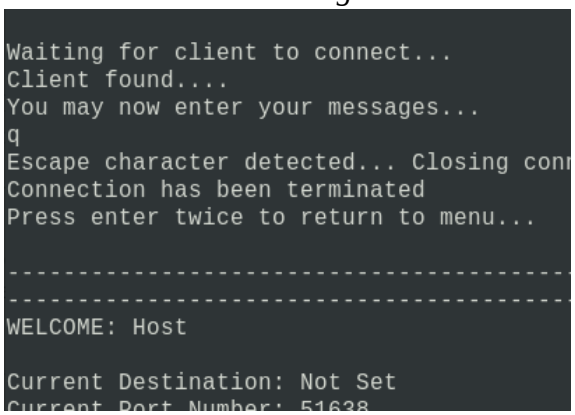


Figure 7c. Client

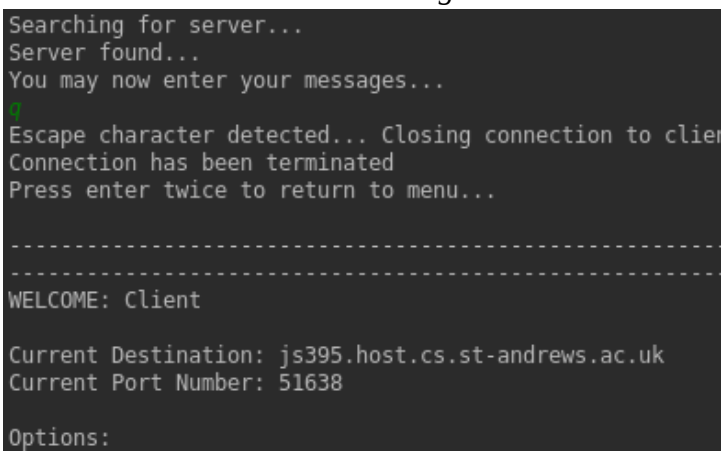
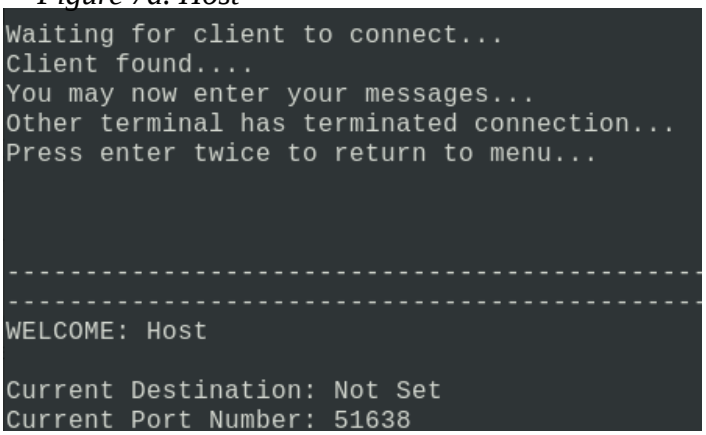


Figure 7d. Host



Test 7 Abnormal Connection Termination

Test Brief: Application should gracefully return to menu up sudden termination from connected user.

Input: a. Host Disconnects

b. Client Disconnects

Expected Results: Both should return to menu.

Results: Successful. Figure 8a+b

```
----------------------------------------

Searching for server...
Server found...
You may now enter your messages...
Connection has been terminated
Press enter twice to return to menu...



----------------------------------------
----------------------------------------
WELCOME: Client
```

Figure 8a. Host Disconnects.

Test 8 Send File

Test Brief: Application should be able to send and receive files of arbitrary type.

Input: a. blob.txt in project folder

b. stick.jpeg in project folder

Expected Result: Both files should be copied successfully to src folder.

Result: Successful.

```
Waiting for client to connect...
Client found....
You may now enter your messages...
Connection has been terminated
Press enter twice to return to menu...



----------------------------------------
----------------------------------------
WELCOME: Host
```

Figure 8b. Client Disconnects

Test 9 Invalid Filename

Test Brief: When given an invalid filename the sender should be prompted to try again an arbitrary number of times, at which point the application will close.

Input: Try to send the file blab.fake

Expected Results: Sender should close, receiver should return to menu.

Results: Successful. Figure 9a + b

Figure 9a. Client-Side

```
Searching for server...
Server found...
Enter the directory + name of the file you want to transfer: blab.fake
File does not exist, please try again...
Enter the directory + name of the file you want to transfer: blab.fake
File does not exist, please try again...
Enter the directory + name of the file you want to transfer: blab.fake
File does not exist, please try again...
Enter the directory + name of the file you want to transfer: blab.fake
File does not exist, please try again...
Maximum attempts reached..
Check the file exists and try again..
Program closing..

Process finished with exit code 255
```

Figure 9b. Host-Side

```
Choose an Option: 6

----------------------------------------
----------------------------------------

Waiting for client to connect...
Client found....
Connection Lost returning to menu...

----------------------------------------
----------------------------------------
WELCOME: Host

Current Destination: Not Set
Current Port Number: 51638
```

Difficulties

The biggest issue when designing this program was designing it in such I way that it utilized the concepts of Object-Oriented programming(OOP). My very first model had all the methods and attributes in one class, that while easy to read and understand didn't properly utilize the features OOP offers.

Using multi-threading also proved to be somewhat of a challenge. I had to try several different methods to exit a thread when the connection was being closed before I decided to try using flags.

Bibliography

Source: *https://gist.github.com/CarlEkerot/2693246* – *Last Accessed 02/11/18*

Use: The methods described above were adapted slightly in order to make the FileShare class and its methods.