Mini-Project 1 – Simple Chat Service

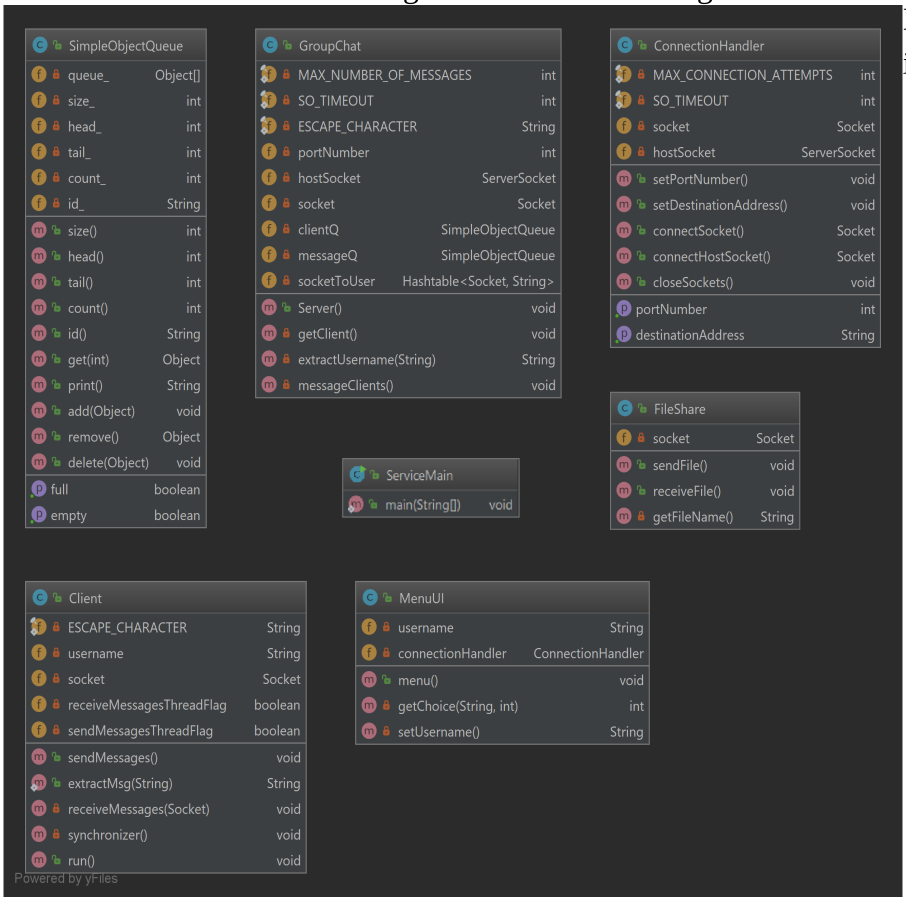
Overview

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.
- Extension: It can send and receive files of arbitrary type.
- Extension: It can host a group chat server.

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.
- As Groupchat class represented a more difficult task it was given it was treated as its own module that would use and close all its own attributes.

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.

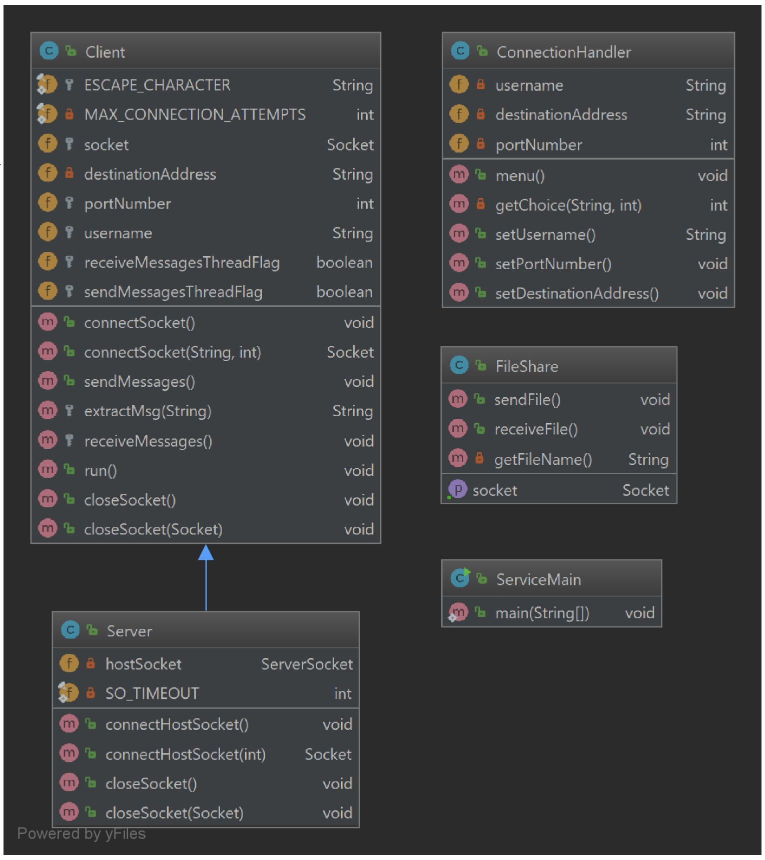


Figure 2. Initial Class Diagram

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.

I chose to use queues for the group chat server as it would be significantly simpler than coming up with and implementing a multi-threading solution to the problem.

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.

The MenuUI class acts as the interface for the user with the application. It allows them to set the destination address and portnumber, and select which function of the application they want to run.

ConnectionHandler Class

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.

The ClientHasNotConnectionException is for when a user fails to connect to another user, both if they are hosting or the one trying to connect. The InvalidSocketAddressException informs the user that the address they have entered is not currently hosting a connection.

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

Client Class

The messaging protocol I used was very simple. When connecting to a a single user, each application would send their username to the other in order to establish identity. For each subsequent message provided by the user, 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, and as a final measure the synchronizer method ensures that the main thread of the application doesn’t continue until the receiveMessages thread is finished.

FileShare class

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.

I chose to use queues for the GroupChat class as it allowed me to receive messages from the various clients in the order that they were sent. I used Prof Bhatti's implementation of an object queue over a standard object array because when adding and removing clients it can already delete a specific element out of the queue and reshuffle it so there are no gaps. This makes implementation of the GroupChat class much simpler. I decided to use a hashtable for mapping sockets to users because it would give me a constant time lookup and since users could not share a socket there would no chance of a socket being overwritten.

GroupChat

The GroupChat algorithm works because each cycle of the primary part of the Server method happens extremely fast. The only time the server is blocking for input is when it checks for new connections for 10 ms. This allows it to sends messages it receives to its users with very little delay. The short amount of time a client has to connect is not an issue because the implementation of the client tries multiple times.

In the GroupChat class at line 140. I set the socket's TcpNoDelay to true. This was because this disables Nagle's algorithm, which is typically used when sending large amounts of data over a socket to limit congestion on a network (http://www.faqs.org/rfcs/rfc896.html). However this is unnecessary for a chat app as only small amounts of data are being sent to across the network so disabling it could offer marginal increases in speed.

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

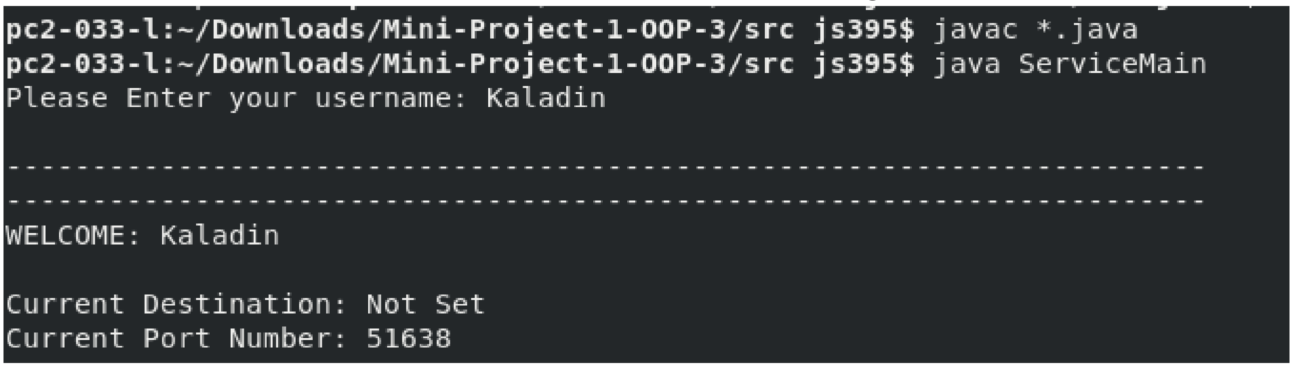
b. java ServiceMain <My Host Domain>

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

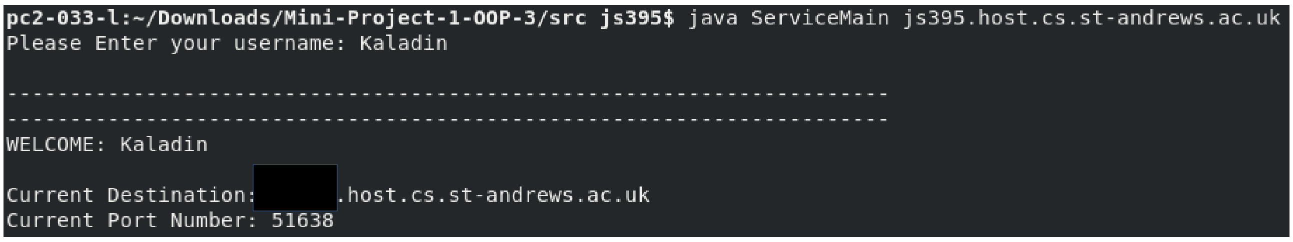


```
pc2-033-l:~/Downloads/Mini-Project-1-OOP-3/src js395$ javac *.java
pc2-033-l:~/Downloads/Mini-Project-1-OOP-3/src js395$ java ServiceMain
Please Enter your username: Kaladin

-----------------------------------------------------------------
-----------------------------------------------------------------
WELCOME: Kaladin

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

Figure 4. Test 1b



```
pc2-033-l:~/Downloads/Mini-Project-1-OOP-3/src js395$ java ServiceMain js395.host.cs.st-andrews.ac.uk
Please Enter your username: Kaladin

----------------------------------------------------------------
----------------------------------------------------------------
WELCOME: Kaladin

Current Destination:[redacted].host.cs.st-andrews.ac.uk
Current Port Number: 51638
```

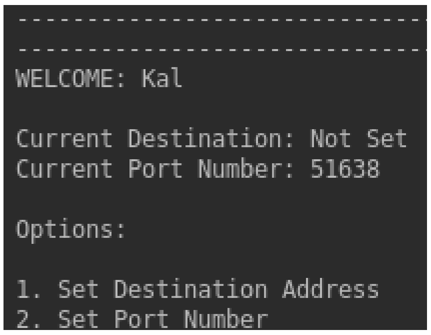
Test 2 Change Destination Address and Port Number

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 <My Host Domain>and 38516.

Expected Result: Current Destination should change from 'Not Set' to '<My Host Domain>'

Result: Successful.



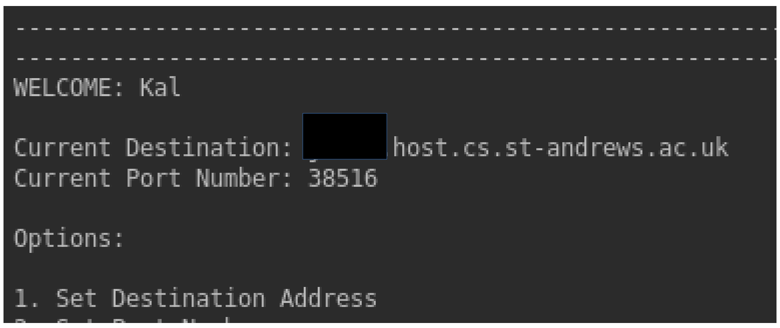
```
------------------------------
------------------------------
WELCOME: Kal

Current Destination: Not Set
Current Port Number: 51638

Options:

1. Set Destination Address
2. Set Port Number
```

Figure 5a. Before



```
-------------------------------------------------------
-------------------------------------------------------
WELCOME: Kal

Current Destination: [redacted]host.cs.st-andrews.ac.uk
Current Port Number: 38516

Options:

1. Set Destination Address
```

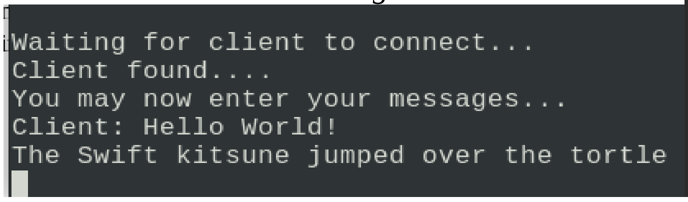
Figure 5b. After

Test 3 Connect to/ Host Another User

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

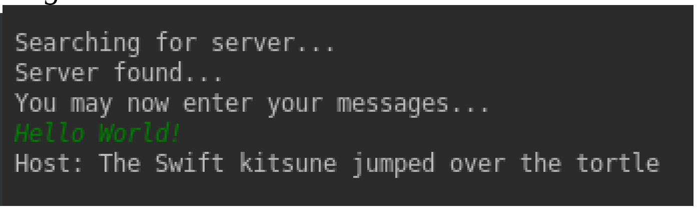
Input: One terminal will ssh onto <My Host Domain> and will host the connection. They will then exchange a simple conversation.

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.



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

Figure 6a. Host side



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

Figure 6b. Client side

Result: Successful. Figure 6a + 6b.

Test 4 – ServerSocket Timeout

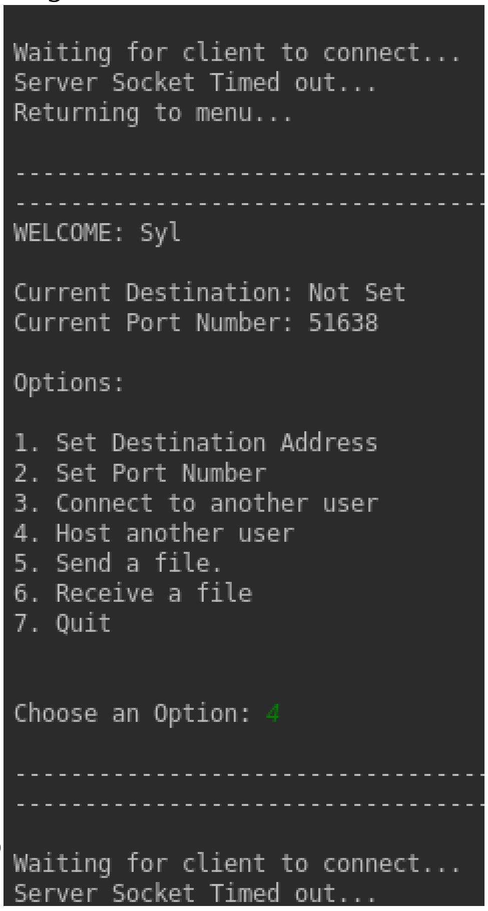
Test Brief: After 10 seconds of waiting the host should return to menu, and be allowed to try to try to connect again afterwards.

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...
```

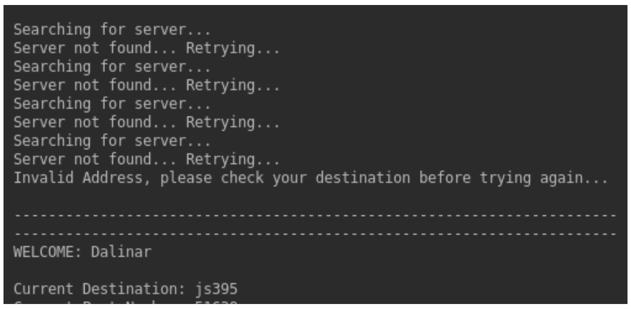
Test 5 Repeat Connect

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 (not an address)

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

Result: Successful.



```
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
```

Figure 8. Repeated connection attempts

Test 6 Normal Connection Termination

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

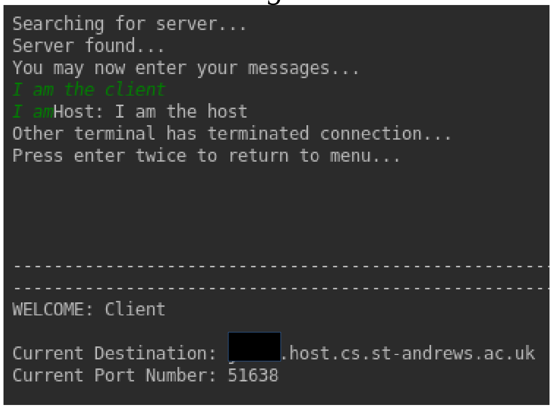
Input: a. “q” sent from Host.

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



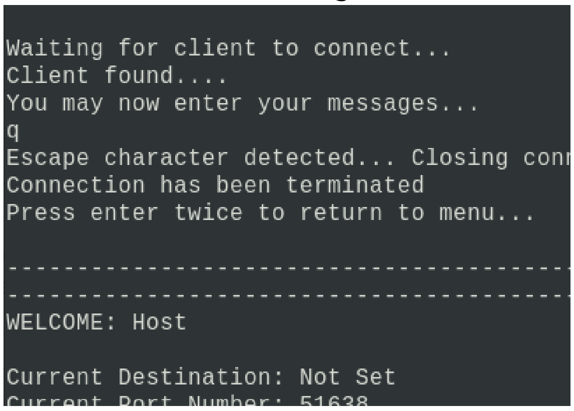
```
Searching for server...
Server found...
You may now enter your messages...
I am the client
I amHost: I am the host
Other terminal has terminated connection...
Press enter twice to return to menu...



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

Current Destination: [redacted].host.cs.st-andrews.ac.uk
Current Port Number: 51638
```

Figure 7b. Host

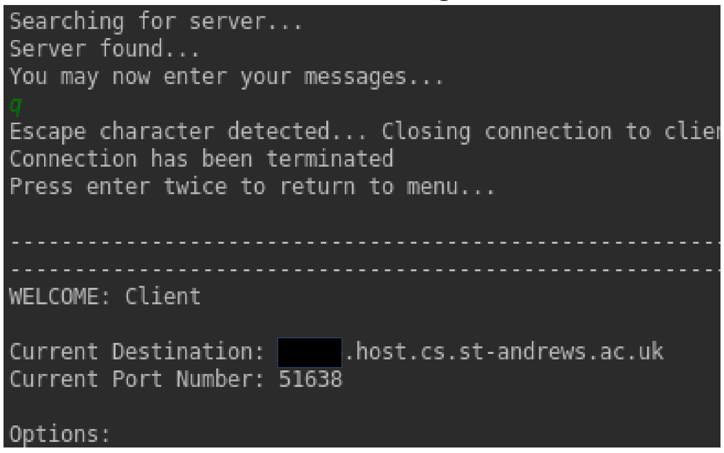


```
Waiting for client to connect...
Client found....
You may now enter your messages...
q
Escape character detected... Closing conn
Connection has been terminated
Press enter twice to return to menu...

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

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

Figure 7c. Client



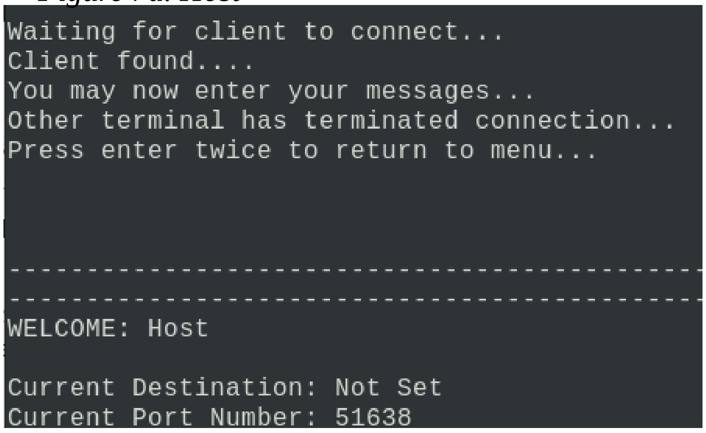
```
Searching for server...
Server found...
You may now enter your messages...
q
Escape character detected... Closing connection to clie
Connection has been terminated
Press enter twice to return to menu...

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

Current Destination: [redacted].host.cs.st-andrews.ac.uk
Current Port Number: 51638

Options:
```

Figure 7d. Host



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



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

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

Test 7 Abnormal Connection Termination

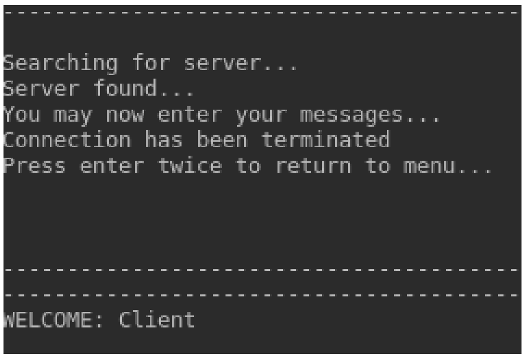
Test Brief: Application should gracefully return to menu up sudden termination from connected user.

Input: a. Host Disconnects

b. Client Disconnects

Expected Results: Both should return to menu.

Results: Successful. Figure 8a+b



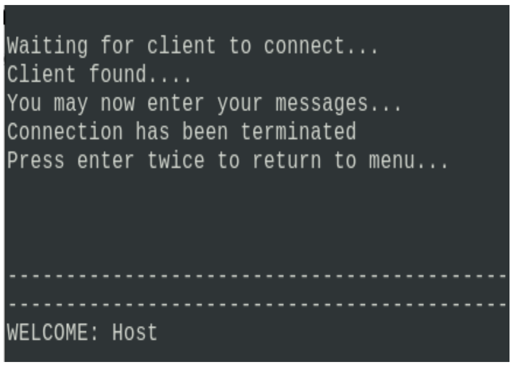
```
----------------------------------------

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.



```
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 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.

Test 9 Invalid Filename

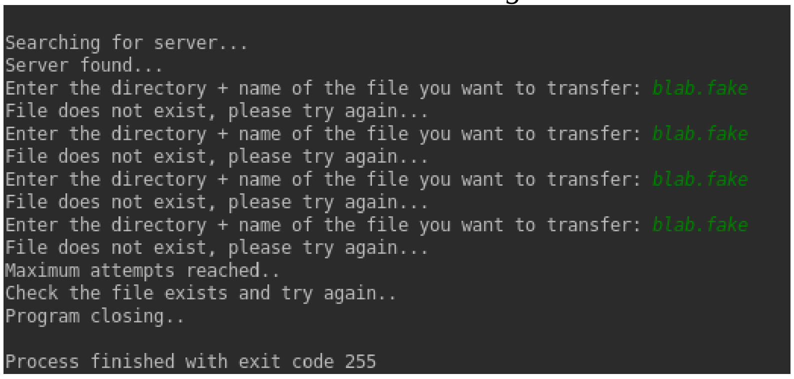
Test Brief: When given an invalid filename the sender should be prompted to try again an arbitrary number of times, at which point the application will close.

Input: Try to send the file blab.fake

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

Results: Successful. Figure 9a + b

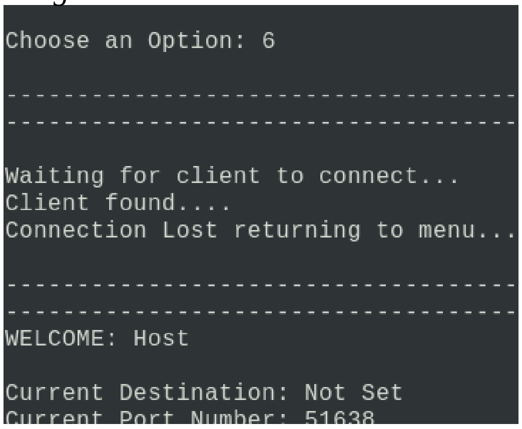
Figure 9a. Client-Side



```
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
```

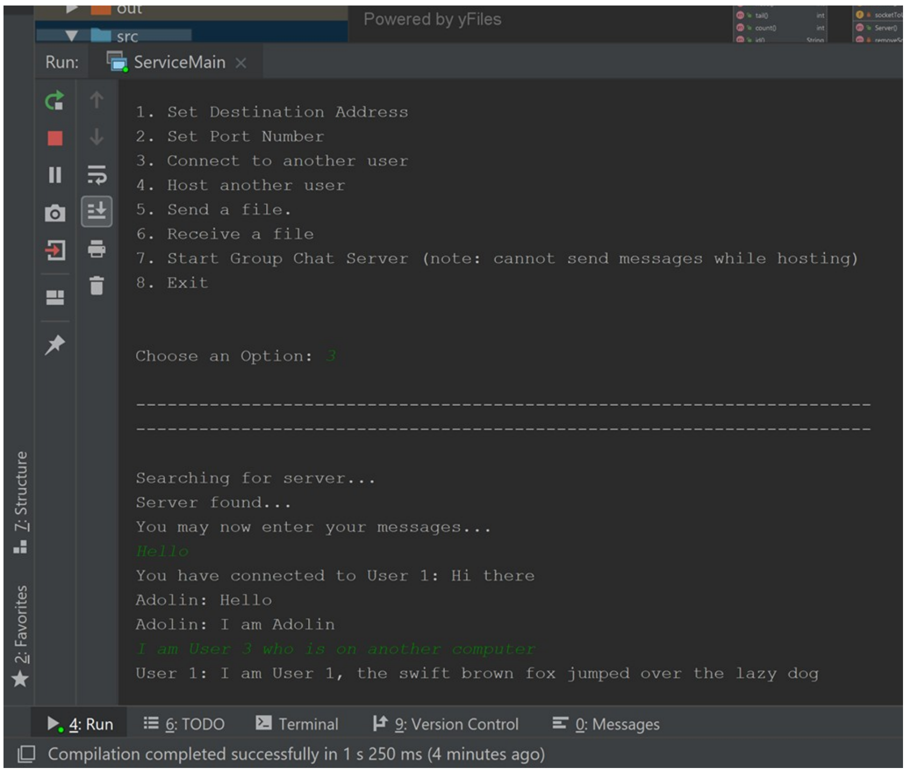
Test 10 Group chat

Test Brief: When a group chat server is started at least 3 users should be able to connect to it and exchange messages.

Input: Terminal 1 (Host) should start a group chat on my host domain. Terminals 2-4 will connect to it and exchange messages.

Expected Results: Messages should be sent to each client with the appropriate username. The user that sent the message should not receive a copy of the message they sent.

Results: Mostly Successful. Currently the first two users to connect give a small 'you have connected to' message before their username and message.



```
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. Start Group Chat Server (note: cannot send messages while hosting)
8. Exit


Choose an Option: 3

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


Searching for server...
Server found...
You may now enter your messages...
Hello
You have connected to User 1: Hi there
Adolin: Hello
Adolin: I am Adolin
I am User 3 who is on another computer
User 1: I am User 1, the swift brown fox jumped over the lazy dog
```

```
Compilation completed successfully in 1 s 250 ms (4 minutes ago)
```

Figure 10. Group Chat

Test 11 Exit Group Chat

Test Brief: When a user exits the group chat there shouldn't be any change in service for the other users. Once all the users have left the server should close automatically.

Input: three users connected to a server hosted at my Domain. a. One user disconnects using an escape character.

b. The other user closes their program.

c. The last should leave with an escape character.

Expected result: After the first user the leaves the others should be informed. After the second user leaves the remaining user should be informed. Once the last user has leaved the server should return to menu.

Result: a. Pass, then from b fail, the server notices the last user leaving but doesn't return to menu. The server doesn't recognise that a user has disconnected in test b. This is likely because it only checks the BufferedReader (the input stream)to see if stream won't block (line. 86). A socket which disconnects suddenly will send 'null' on its input stream which I suspect the code treats it as though it will block (Figure 11). If all users exit normally the server returns to menu successfully (Figure 12)

Difficulties

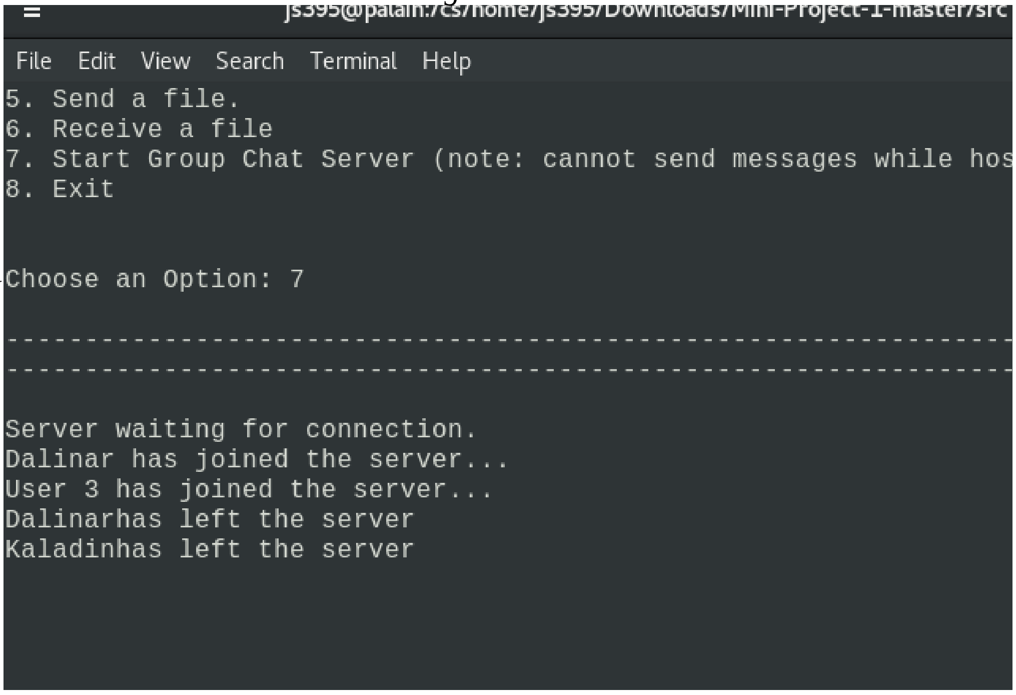
The biggest issue when designing this program was designing it in such I way that it utilized the concepts of Object-Oriented programming(OOP). My very first model had all the methods and attributes in one class, that while easy to read and understand didn't properly utilize the features OOP offers.

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.

For the group chat application I have not been able to determine why the first to messages between the first 2 connect users will be prefixed by 'you have been connected to <user> : <message>'.

In addition I cannot think of a solution to Test 11 whereby an abnormal disconnection from the stream won't be recognised by the group chat server.

Figure 11. Abnormal Disconnect


js395@palain:/cs/home/js395/Downloads/Mini-Project-1-master/src
File Edit View Search Terminal Help

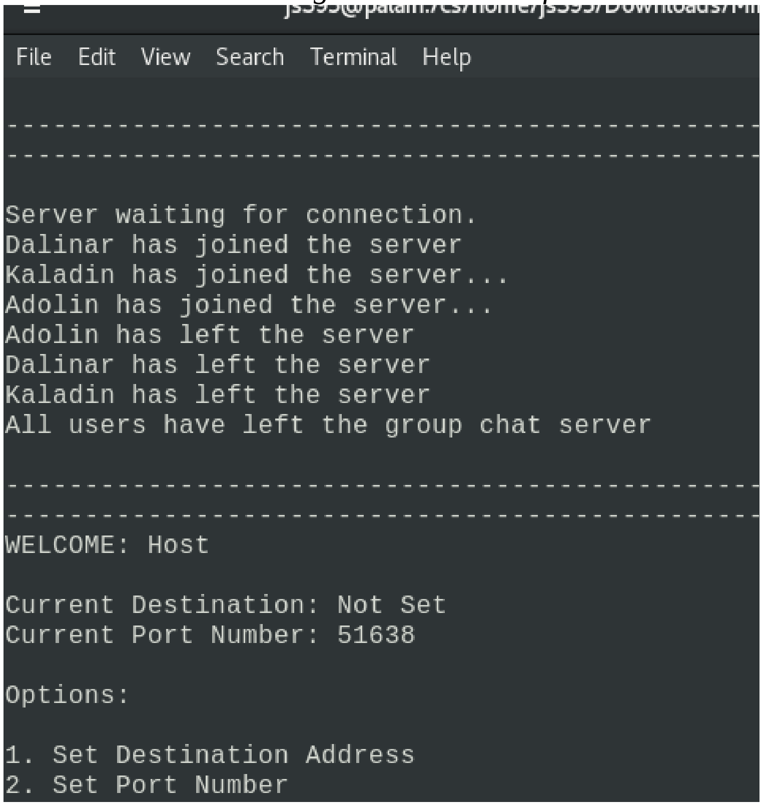
```
5. Send a file.
6. Receive a file
7. Start Group Chat Server (note: cannot send messages while hos
8. Exit


Choose an Option: 7

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

Server waiting for connection.
Dalinar has joined the server...
User 3 has joined the server...
Dalinarhas left the server
Kaladinhas left the server
```

Figure 12 – Successful Exit to menu


js395@palain:/cs/home/js395/Downloads/Mi
File Edit View Search Terminal Help

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

Server waiting for connection.
Dalinar has joined the server
Kaladin has joined the server...
Adolin has joined the server...
Adolin has left the server
Dalinar has left the server
Kaladin has left the server
All users have left the group chat server

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

Current Destination: Not Set
Current Port Number: 51638

Options:

1. Set Destination Address
2. Set Port Number
```

Bibliography

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

Use: The methods described above were adapted in order to make the FileShare class and its methods. I adapted the above by making a messaging protocol so that the filename and and file length would be sent with the file. Simplifying matters for both users.

Source: http://www.jguru.com/faq/view.jsp?EID=42242 - *Last Accessed 02/11/18*

Source:http://www.faqs.org/rfcs/rfc896.html - *Last Accessed 02/11/18*

Use: The above sources were used to understand why I should have set a socket's TcpNoDelay attribute to true.

Source: https://studres.cs.st-andrews.ac.uk/CS2003/Examples/wk03/MultiChat/MultiChatServer.java - *Last Accessed 02/11/18*

Author: Prof Saleem Bhatti

Use: Much of the code for the GroupChat class utilized code already implemented from the above link. A summary of adaptations I made from the above code:

- Our messaging protocols were different so I had to change how messages were added to the queue
- I added a hashtable of Sockets to users so that a user would not receive a copy of a message they sent to the server.
- I decomposed the main method from the source code into several smaller methods that should be easier to read.

Source: https://studres.cs.st-andrews.ac.uk/CS2003/Examples/wk03/MultiChat/QueueEmptyException.java - *Last Accessed 02/11/18*

Source: https://studres.cs.st-andrews.ac.uk/CS2003/Examples/wk03/MultiChat/QueueFullException.java - *Last Accessed 02/11/18*

Source: https://studres.cs.st-andrews.ac.uk/CS2003/Examples/wk03/MultiChat/SimpleObjectQueue.java - *Last Accessed 02/11/18*

Use: The above code was utilized in the GroupChat class. It required queues to function and the exceptions were required for the queue to work.