CS2101: Mini – Project – 1:
MATRICULATION ID: 200007645
Tutor: Alex Konovalov.

Simple File Sharing (SFS)

Introduction:
My task was to build a Simple File Sharing (SFS) application in Java which permits you to share files with other people using my application.

The file sharing application will allow the user to share a specified directory on a specified port of the machine. When another machine that's running the same program connects to the previously mentioned machine, it will have access to all the files in the directory, which can choose a specific file and request it to download it to its user specified local directory.

Requirements:
The application should permit users:

1. To share files with other users and to download files shared by other connected users.
2. To specify shared and saved directories from which files are shared and to which downloaded files are saved.
3. To access the features of the client part of the function using a simple text-based interface.
4. The server part of the application should respond to incoming requests from other application instances by returning the appropriate data over the network.
5. The application should provide reliable data delivery, so it should use TCP for data transfer.
6. The application should deal with abnormal client-side and server-side connection termination.

Extensions:
Application should:

7. Share and download any type of file.
8. Provide the user with the list of computers that are sharing the files.
9. Permit the user to start an automated search (over all participating computers on the network) for files by specifying suitable search criteria as well as list all the file from the shared directories that are share.
10. Provide a GUI in addition to the simple text-based requirements listed above.

To share files with other users and to download files shared by other connected users:

- This was one of the core requirements of my application. This feature allows a single machine to share a specified directory on a specific port which can be connected from other machines.
- The other part of the program is to connect to other machines that are sharing Directories and download shared files to a specified local directory. These 2 parts run concurrently on the machine by running the server part of the program on another thread.

- This also allows a single machine to share multiple directories on multiple ports whilst running the same program. Whilst, connecting to other machines to download files.

To specify shared and saved directories from which files are shared and to which downloaded files are saved:

- This was also core requirement of the application. This feature will allow the user to specify the shared and save directory. This is essential for the program as this provides some security in the fact that the client can only download files from the specified directory.
- When sharing the user will have to specify a specific directory to path for the program to start a server.
- On the client side, the user can use the set command to set a download directory path.
- The directory path for the download can be changed by the set command unlike the share directory which cannot be changed.

To access the features of the client part of the function using a simple text-based interface:

- On the client part of the application, the user has multiple commands, they can use to access the features of the application. They are:
    - "set <pathForDirectory>" to specify a download directory.
    - "list files" to list all the files that the user can download from all the shared directories. This will also tell the user on which machine i.e. shared directory those files belong as well
    - "list servers" to list all the machines that the user can download files from.
    - "find servers" to list all the machines in the local network that are sharing.
    - "search <filename>" to search for a specific file in the list of shared files.
    - "search <.format>" to filter out the files that are shared based on its extension and
    - "get <filename.format> from directory <Dir no.>" this command will go to the shared directory specified by <Dir no.> then search if there is a file <filename.format>, if it exists it will request the server to send it through the output stream so that it can be saved on the download directory.
    - "connect to <ip> at <port>" this command allows the user to connect to multiple servers i.e. access multiple shared directories.
    - "share <pathForDirectory> at <port>" this command allows the user to share a directory on a port, which will run on an alternate thread all the whilst being connected to other machines.
- This set of commands allows the user to have a better understanding of which files are being shared and how they can download them to their local download directive.
- Since the user interface which requests for command is surrounded within a for loop, the user can give any number of commands.
- The receiver will only stop and close all the connection with the machine if the user gives the command "exit".

The server part of the application should respond to incoming requests from other application instances by returning the appropriate data over the network:

- The client part of the application has many different commands which the server would have to respond correctly:
    - list files
    - get a specific file
- When the user tells the client to list all the files in the shared directory, the client will send to each server a specific command, in this application, "list files" through the sockets output stream. Which the server will recognise, then get the list of files inside the directory and then write them to the client through the servers output streams.
- When the user requests a specific file from server, the client will send the command "get <filename.format>" through the sockets output stream. Which the server will recognise, then confirms that the file exists. Once confirmed, a buffered input reader will read the file and then send the file through the output stream to the client which will then be read and saved in the local download directory.
- The communication between the server and client is handled by input and output streams, which only sends bytes, in order to handle the communication I have utilised a write method within the server and the receiver class which converts a string into bytes and sends the length of the string of the before the bytes so the other side can make sense of the data that's being sent.

The application should provide reliable data delivery, so it should use TCP for data transfer:

- TCP is connection oriented, such that an exclusive connection between must be 1st established between the server and client for communication to take place.
- In my application, for each server the program is connected to access the files there is a dedicated socket. Whenever there is a need to communicate with a specific server, there specific output and input streams are used.
- My application only uses sockets to provide the communication mechanism between the two machines, which uses TCP.
- Since, TCP is a two-way communication protocol, which my application uses to communicate between the server and the client.

The application should deal with abnormal client-side and server-side connection termination:

- on the server side, the program handles any abnormal termination as well broken pipes by catching the exception and safely closing the socket connection as well as any buffered readers.
- On the client side, if there was an unexpected termination, that specific socket is called and removed from the list of sockets.

The application should share and download any type of file:

- When the client gives a command to get specific file from a specific server, the respective input stream and output stream is wrapped in buffered output/input streams. Then get "filename" command is written through the socket output stream, which the server interprets and then begins to read that specific file input stream

wrapped inside a buffered input stream, each byte that's being read is then written to the socket using the servers output stream.

- The client which was waiting for the reply from the server gets the byte from the sockets input stream and write them to a new file of the same name and type in the local download directory.
- Since all the communication between the client and the server is wrapped inside a buffered input stream which writes and reads bytes, any type of file can be shared.

The application should provide the user with the list of computers that are sharing the files:

- When the user is connected to one or more machines, they can use the command "list servers" to get a list of computers that are sharing directories.
- This feature lists for each server it lists the IP address and the port its connected to.
- When the user uses the command "find servers" either in the start-up interface or the receiver interface the program will display to the user which machines are sharing i.e. IP address and the ports they are sharing on.
- This feature is implemented using multicast socket. This is UDP based communication.
- Once a MulticastSocket object is created, the joinGroup() method is invoked to make it one of the members to receive a multicast message. When the user commands the application to find servers, the application will send a request to the group, in response, the machines that are sharing will send a message with their IP and ports they are sharing on.

The application should Permit the user to start an automated search:

- When the user is connected to one or more machines, they can use the command "list files" to get a list of files that are being shared by the servers that the user is connected to.
- The user can also use the "search <filename>" or "search <.ext>" to search for all the files that are being shared that matches the filename or if the file name at least resembles the word that's being specified by the user.
- When the user searches for a file by the extension, the application will retrieve all the files that are of the file type requested by the user.
- This feature only allows the user to search for files in the machines that the receiver is connected to.

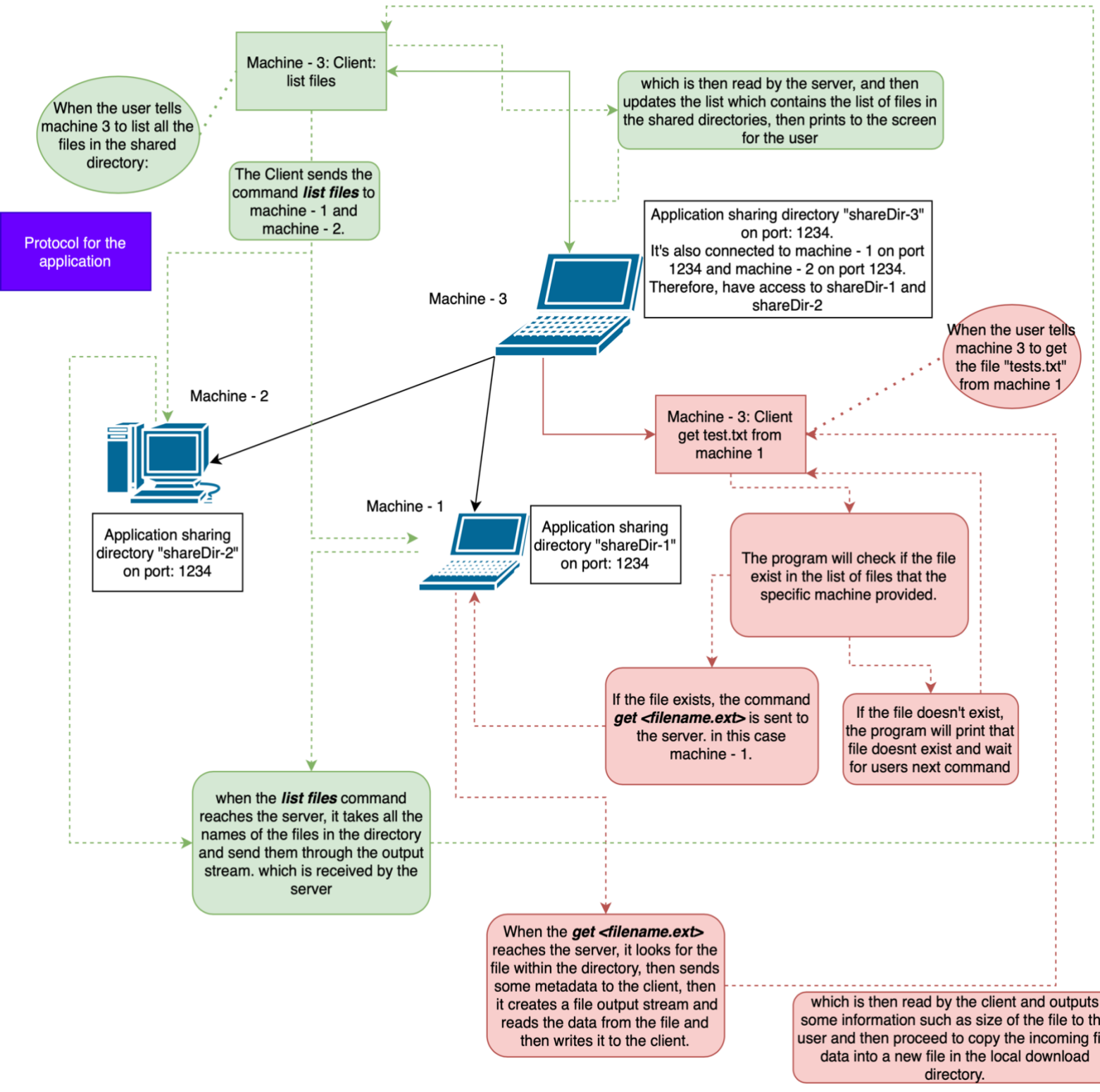
Provide a GUI in addition to the simple text-based requirements listed above

- Due to time constraints as well as how I implemented my application, I was unsuccessful in implementing an GUI interface in addition to the text-based interface.
- It would have also been very difficult and nearly impossible for me to test them if I was connected to my cs remote host.

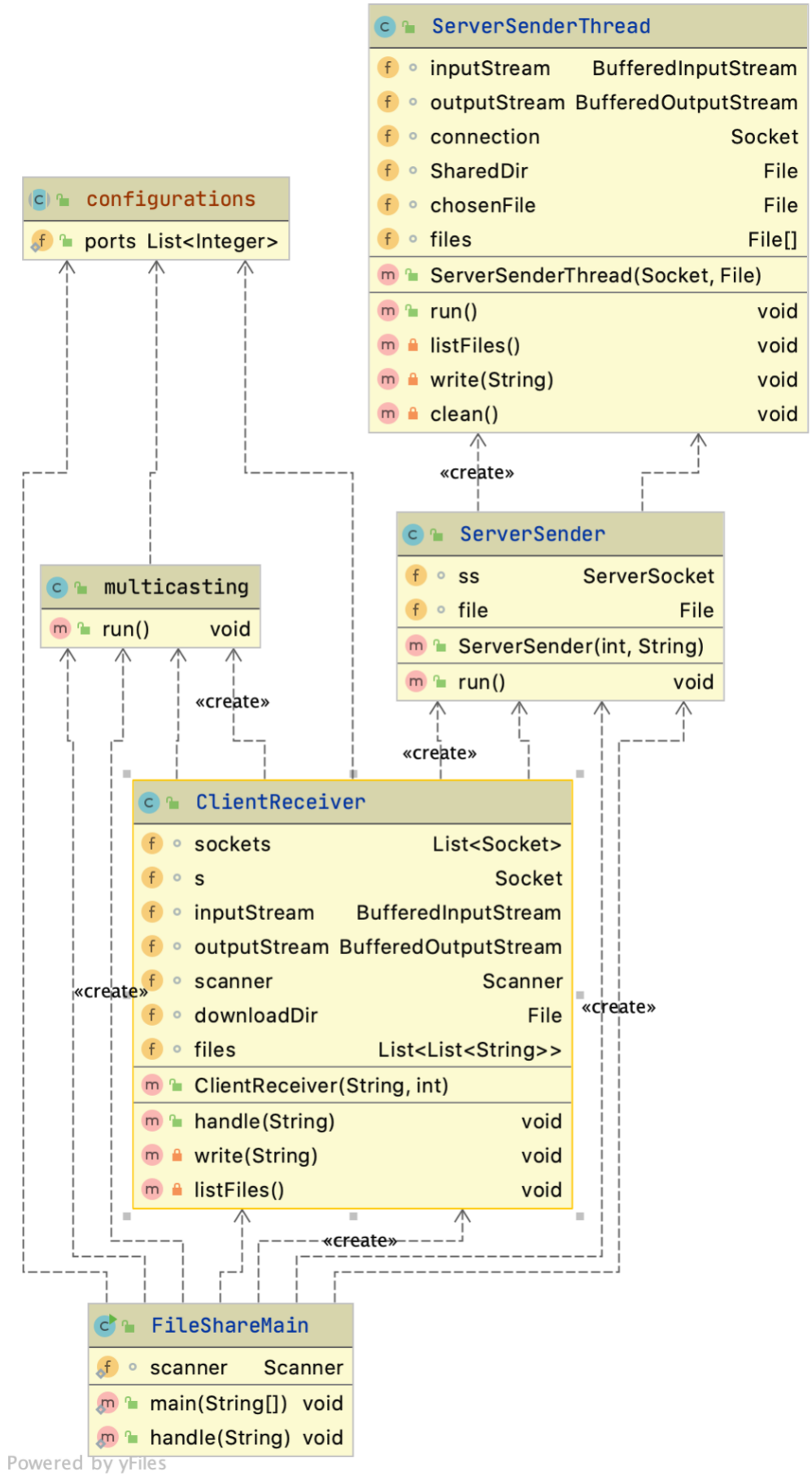
Protocol for the application:

The communication that takes place between the server and the client.

When the client, sends a command to the server, the string is converted into bytes and sent the length of the string and the bytes through the output stream which is then converted back into a string to interpret the command and proceed accordingly.



UML class diagram:


ServerSenderThread
inputStream BufferedInputStream
outputStream BufferedOutputStream
connection Socket
SharedDir File
chosenFile File
files File[]
ServerSenderThread(Socket, File)
run() void
listFiles() void
write(String) void
clean() void
configurations
ports List<Integer>
«create»
ServerSender
ss ServerSocket
file File
ServerSender(int, String)
run() void
multicasting
run() void
«create»
«create»
ClientReceiver
sockets List<Socket>
s Socket
inputStream BufferedInputStream
outputStream BufferedOutputStream
scanner Scanner
downloadDir File
files List<List<String>>
ClientReceiver(String, int)
handle(String) void
write(String) void
listFiles() void
«create»
«create»
«create»
FileShareMain
scanner Scanner
main(String[]) void
handle(String) void
Powered by yFiles

Overview of how the application operates:
When the application is run, the main method in the FileShareMain class is where the program starts. The user will have 2 options, either they can share a directory at a specific folder or connect to another machine.

If the user chooses to share a directory, the application will then create a new instance of ServerSender class. Within this class, a new ServerSocket is then instantiated at the specified port. Then an infinite loop is run on an alternate thread, waiting for socket connections, if there was a connection, a ServerSenderThread object is instantiated, which handles the connection between the server and a client on another thread. This allows the application to have multiple servers and each server can have multiple client connected to them simultaneously.

After the process of starting up a server is completed, an instance of the multicast class is created. There is one object of this class for each machine. This class is run on another as well. On this thread, the program waits for multicast request from another machine in the local network to list all the servers its running. Which will sent be sent to that machine.

Then, the user is back at the start position where he has 2 options - share or connect.
If the user chooses to connect to a server, they would have to provide the IP address of the machine and the port the server is running on as the arguments. Using those arguments, a ClientReceiver class is instantiated.

Within the ClientReceiver class, a new socket is created and added to the list of sockets. Then application outputs the set of commands available to the user, the user then could choose to connect to other machines using the connect command, where that socket will also be added to the list, the user at this point can also choose to share another directory on a port as well. They can also choose to do the above-mentioned features: such as list files, list servers, get a specific file from a directory. Before the user can download a file from other machines, he must set up a download directory, which can be done using the set command. The user can also choose to close all the connection with the servers using the exit command.

When the user tells the application to get a specific file, first the server gets an updated list of files in the shared directories, if the file exists in the specified directory, it writes to the respective server to “get <filename>”. Which the server, responds to by sending some information about the file then progress to read the file then writes the data through the output stream. Once the data has been received, the input stream which read the file in the server is closed and goes back to waiting for command from the client, on the client side the data is saved and the file output stream is closed and the client goes back to waiting for users command.

Interesting features:

Sharing multiple directories on different ports: due to running a server on another thread, the application allows the user to have multiple servers sharing different directories on alternate ports. This feature will be useful for instance when sharing specific files that only certain machines should have access to not all the machines in the network.

Testing:
Sharing multiple directories on different ports on the same machine:

```
almh1@pc3-003-l:~/Documents/CS2101/mp-1/src $ java FileShareMain
P2P file sharing!
Available Commands:
+ for sharing a directory: share <pathForDirectory> at <port> eg:- share user/dir1/dir2/Sha
ring/ at 1234
+ for connecting to another Machine: connect to <ip> at <port> eg:- connect to localhost at
 1234
+ for seeing the list of servers available to connect to: find servers
+ for exit: exit
Your command: share /home/almh1/Documents/CS2101/mp-1/Share/ at 1234
Sharing the Directory: /home/almh1/Documents/CS2101/mp-1/Share/ at port: 1234
Your command: share /home/almh1/Documents/CS2101/mp-1/Share-1/ at 12345
Sharing the Directory: /home/almh1/Documents/CS2101/mp-1/Share-1/ at port: 12345
Your command:
```

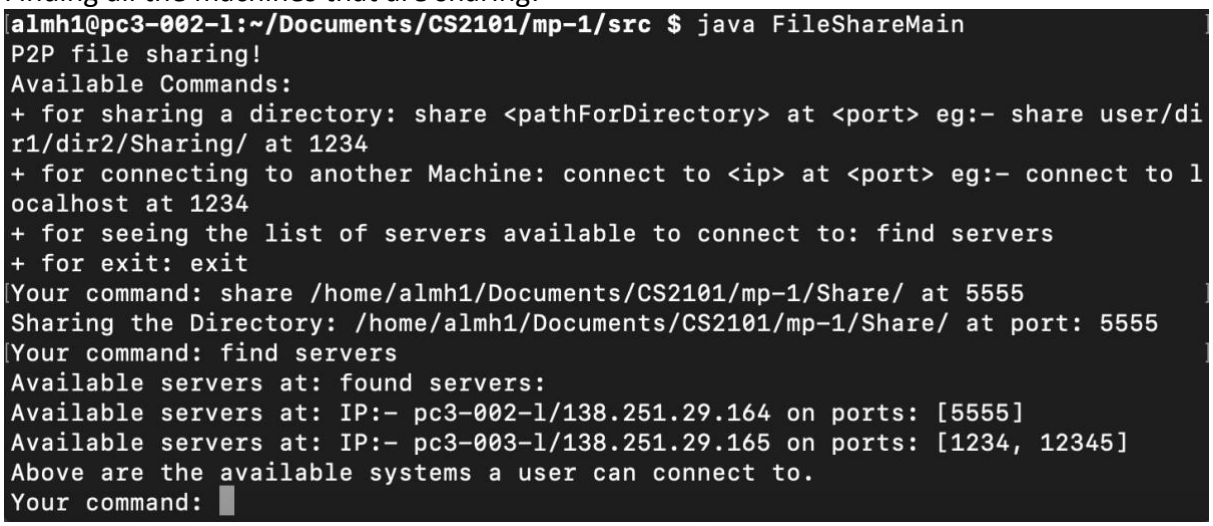
I have connected to a remote client and rung the application. At the start of the application, the user was given 2 options. I chose to share 2 different directories on different ports.

Connecting to other machines:

```
Your command: connect to 138.251.29.165 at 1234
Connected to 138.251.29.165 at port: 1234
--------------------------------------------------------------------------------
Set of Commands for server:
+ Set your download Dir: set <pathForDirectory> eg:- set /Users/user/dir/dir1/downloadDir/
+ list shared files: list files
+ list the computers that are sharing: list servers
+ for seeing the list of servers available to connect to: find servers
+ search for specific file: search <filename>
+ search for specific file by extension: search <.format>
+ download a specific file: get <filename.format> from machine <machine no.> eg:- get test.
txt from directory 2
+ connect to another machine: connect to <ip> at <port>
+ for sharing a directory: share <pathForDirectory> at <port>
+ close your connection: exit
Your command for server: connect to 138.251.29.165 at 12345
Connected to 138.251.29.165 at port: 12345
Your command for server: list servers
The Machines that we are accessing the shared directories:
Machine 1: IP: /138.251.29.165 at port: 1234
Machine 2: IP: /138.251.29.165 at port: 12345
Your command for server:
```

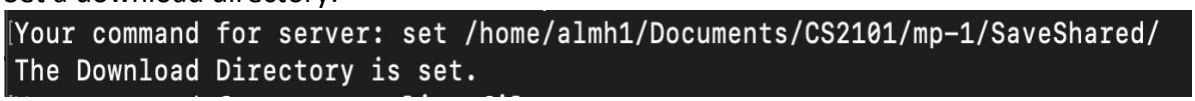
In this image, I used a terminal to connect to another lab client to run my application and connected to the servers that's sharing the directories, here as you can see, I have a set of commands I could use. When the user tells the application to list the servers its connected to, which outputs the IP address and the port.

Finding all the machines that are sharing:



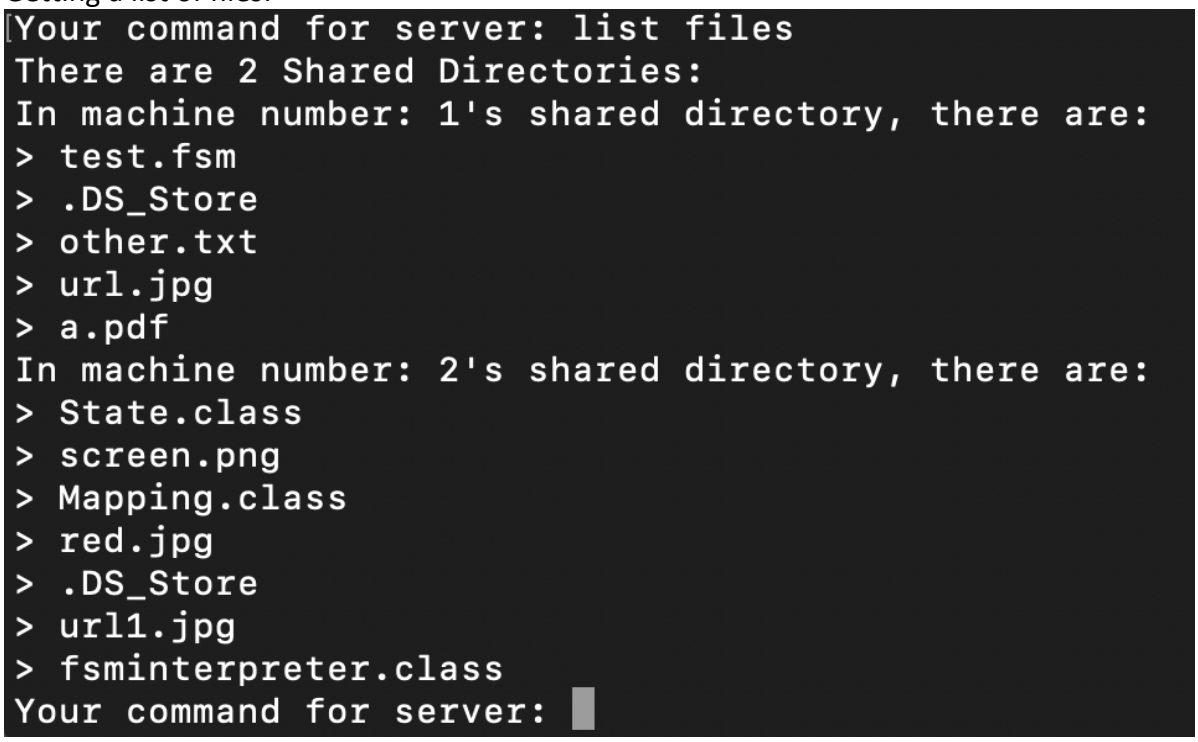
In this image, I'm ssh-ing into a remote client running my application. Here I'm sharing a directory at the port on that machine, then I use the command list servers to get the list of all the server that are sharing in the local network, which means it will return the IP address and the ports, the server is running on. From this point the user can use the information to establish a connection to the servers to access the directories.

Set a download directory:

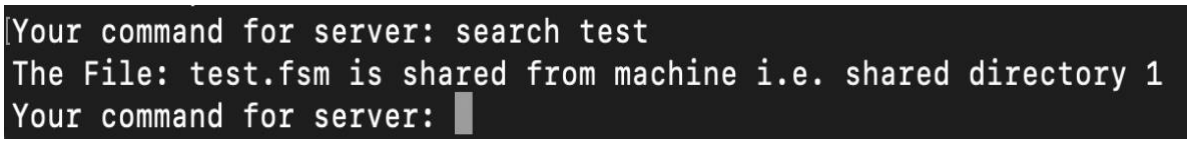


Setting up a download directory for the receiver to download the files.

Getting a list of files:

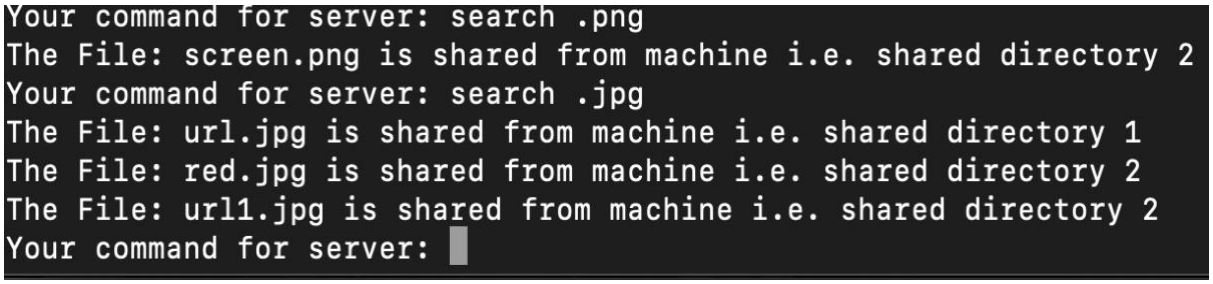


In this image, I used the command “list files” to get the list of files that are being shared by the servers the receiver is connected to.



Search for a specific file:

This command returns the full file name and which directory it belongs to.



Search for a file using its extension:

By searching for a file by the extension, the application will output the full name and which machine it belongs to as well.

Getting a specific file from the server:
In the previous images, I used the “set” command to set the download directory. The next step is to use the “get” command to get the file

```
Your command for server: get url.jpg from machine 1
The File url.jpg is ready to be downloaded. The size of the file is: 381727
file transfer progress: 50%
File has been successfully shared.
File received.
Your command for server:
```

In this image I get the “url.jpg” from machine 1. Where the application 1st confirms if the file exists in machine 1 and then proceed to request the file from machine 1. In response to that request, machine 1 will send “The File url.jpg is ready to be downloaded. The size of the file is: 381727", then proceed to send the contents of the file. The user is updated about the progress of the file. Once the entire file has been transferred, a confirmation is also sent from the server.

```
almh1@lyrane:~/Documents/CS2101/mp-1/SaveShared $ ls
almh1@lyrane:~/Documents/CS2101/mp-1/SaveShared $ ls
url.jpg
```

The above image confirms that the file has been successfully saved to the download directory. Before the file transfer the directory was empty and after the transfer a file exists with the same name and same content the user requested.

Getting binary files:

```
Your command for server: search .class
The File: State.class is shared from machine i.e. shared directory 2
The File: Mapping.class is shared from machine i.e. shared directory 2
The File: fsminterpreter.class is shared from machine i.e. shared directory 2
Your command for server: get State.class from machine 2
The File State.class is ready to be downloaded. The size of the file is: 927
file transfer progress: 50%
File has been successfully shared.
File received.
Your command for server:
```

In this test, the user searched for files with .class extensions. There were 3 files of that file type in machine 2. Then the user used the “get “command to get the file.

```
almh1@lyrane:~/Documents/CS2101/mp-1/SaveShared $ ls
State.class  url.jpg
```

Safely closing the connection between the machines and exiting the program:

```
Your command for server: exit
connection closed with the machine at IP: /138.251.29.165 at port: 1234
connection closed with the machine at IP: /138.251.29.165 at port: 12345
Your command: exit
almh1@pc3-002-l:~/Documents/CS2101/mp-1/src $
```

When the user uses the command “exit” whilst connected to multiple servers, the application will close all the connection to the server safely and informs the user. The user then returns to the beginning. Then, if they use the command exit, the program stops.

Conclusion:

This is the 1st time I used servers and sockets on a project. At the start, it was difficult for me to get started with this practical. Firstly, I started looking at the example provided on stud res - “ClientServerExample”. This example helped me understand how to make a program run on another thread. This was especially useful for the ServerSender and ServerSenderThread Class, since they both run on another thread. I also used the example on studres to understand how to copy a file.

For the 2nd extension, I used examples from the internet to understand how multicast should be coded. I have attached the links to the examples below. My code on how the multicast works is different to the examples provided since, those examples just gave me an idea how a basic multicast application works. Which I then improved and modified to fit my application.

Despite having some challenging aspects, I found writing this project really fun and valuable. I have satisfied all the requirements, except the GUI extension. Next time, I would implement a loosely coupled design of the application so making a GUI for the application would be easier.

Links:

https://stackoverflow.com/questions/3258959/network-discovery-in-java-using-multicasting

https://www.developer.com/java/data/how-to-multicast-using-java-sockets.html accessed at 29th oct at 2 am.