

# 5CS022

## Report on Analysis and Maintenance of Distributed Chat Programs

David Pearson

1725412

## Contents

Part 1 Functionality (20 marks) .....	3
Correct the bug for sharing work between clients .....	3
Add functionality to specify text file to search .....	4
Add functionality to choose between searching for vowels and consonants .....	5
Add functionality to count specific words .....	6
Add functionality to count words over a specified length .....	8
Part 2 Comparison to single threaded (15 marks) .....	9
Sockets chat program .....	9
Threaded chat program .....	9
ChernoChat chat program .....	9
Similarities, differences and what next? .....	9
Part 3 General Commenting (15 marks) .....	10

## Part 1 Functionality (20 marks)

Before starting on the modification of the code provided, either the addition of comments as per Task 3 or the addition of functionality as per Task 1, the decision was made to track all changes via the use of the universities GITlab server. The finished project can be found at the address shown.

<https://fsegitlab.wlv.ac.uk/1725412/5cs022>

For the purposes of academic integrity, the project will remain private until the hand in date has passed, at which time it will be made public, to allow a further in-depth review of the work completed.

All requirements specified within the marking criteria have been met, from minimal through to excellent. Further details on how each element was achieved is shown in the relevant section below.

Please note for the purposes of testing all functionality requirements, each element was checked against three different text files;

- jungle-book.txt
- war-and-peace.txt
- LongfellowPoem.txt

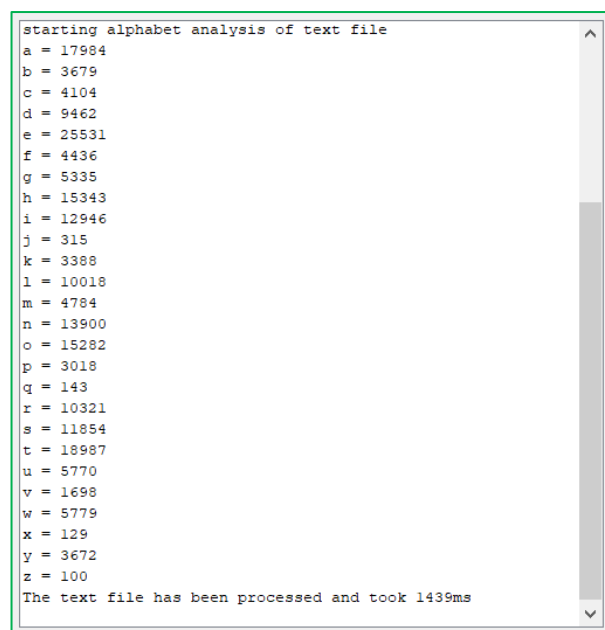
As well as the use of different text files, each test was also carried out with one, ten, fifty and one hundred clients. For the purposes of demonstration i.e. screen shots, within this report ten clients were used in conjunction with the jungle-book.txt file.

### Correct the bug for sharing work between clients

*Fix the bugs in the alphabet counting to ensure result only printed once and the data is merged correctly*

A variable to track the number of lines received was added and this was checked against the lines sent variable, only when these two were equal would the server print out the results. Another change was made as to when then server would update its count, this now takes place earlier in the 'if statement' that is called when a message starting with the specific tag.

The basic logic for the program after clicking start analysis is;



```
starting alphabet analysis of text file
a = 17984
b = 3679
c = 4104
d = 9462
e = 25531
f = 4436
g = 5335
h = 15343
i = 12946
j = 315
k = 3388
l = 10018
m = 4784
n = 13900
o = 15282
p = 3018
q = 143
r = 10321
s = 11854
t = 18987
u = 5770
v = 1698
w = 5779
x = 129
y = 3672
z = 100
The text file has been processed and took 1439ms
```

The server starting with the first line of the file and in sequence send a single line to all connected clients, incrementing a line sent counter. Then it listens.

The client receives a line and checks the start of the message for the specific analysis to be carried out. Analyse the line, use the results to build a message and send it back to the server. Then it listens.

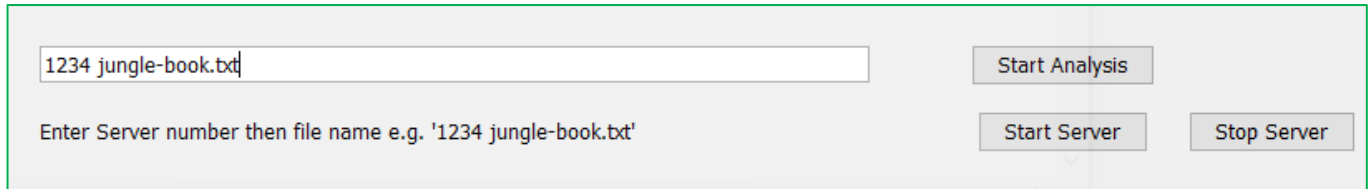
The server receives a line from a client and increments a line received counter. It then then processes the information. Checks to see if there are any more lines to send, if there are it sends the next line back to the client that it just received a line from. Then it listens.

When the server has received the final line and the lines received count matches the lines sent count it will print out the results.

## Add functionality to specify text file to search

Add functionality so that user can load different text files for analysis

The program required that the user enter a four-digit port number to initialise the server. This is done via the text input box in the server window. The change made to meet this requirement was to ask the user to also type the name of the file they wish to be analysed at the same time, separating them by a space character. In addition to many changes to the various class files to add the functionality a helpful prompt was also added to the GUI window.



1234 jungle-book.txt

Start Analysis

Enter Server number then file name e.g. '1234 jungle-book.txt'

Start Server

Stop Server

Here is a snippet of the code showing how the filename is split out of the text.

```
//Start server button, has an action listener
JButton btnStart = new JButton("Start Server");
btnStart.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //This behaves exactly as the above method for the enter key
        String textbox = txtMessage.getText();
        int port = Integer.parseInt(textbox.substring(0, 4));
        String filename = textbox.substring(5);
        System.out.println(port);
        System.out.println(filename);
        server = new Server(port, filename);
        server.output = history;
        txtMessage.setText("");
    }
});
```

Here is a snippet of the code showing how the server constructor now uses the filename variable

```
public Server(int port, String filename) {
    this.port = port;
    this.filename = filename;
    try {
        socket = new DatagramSocket(port);
    } catch (SocketException e) {
        e.printStackTrace();
        return;
    }

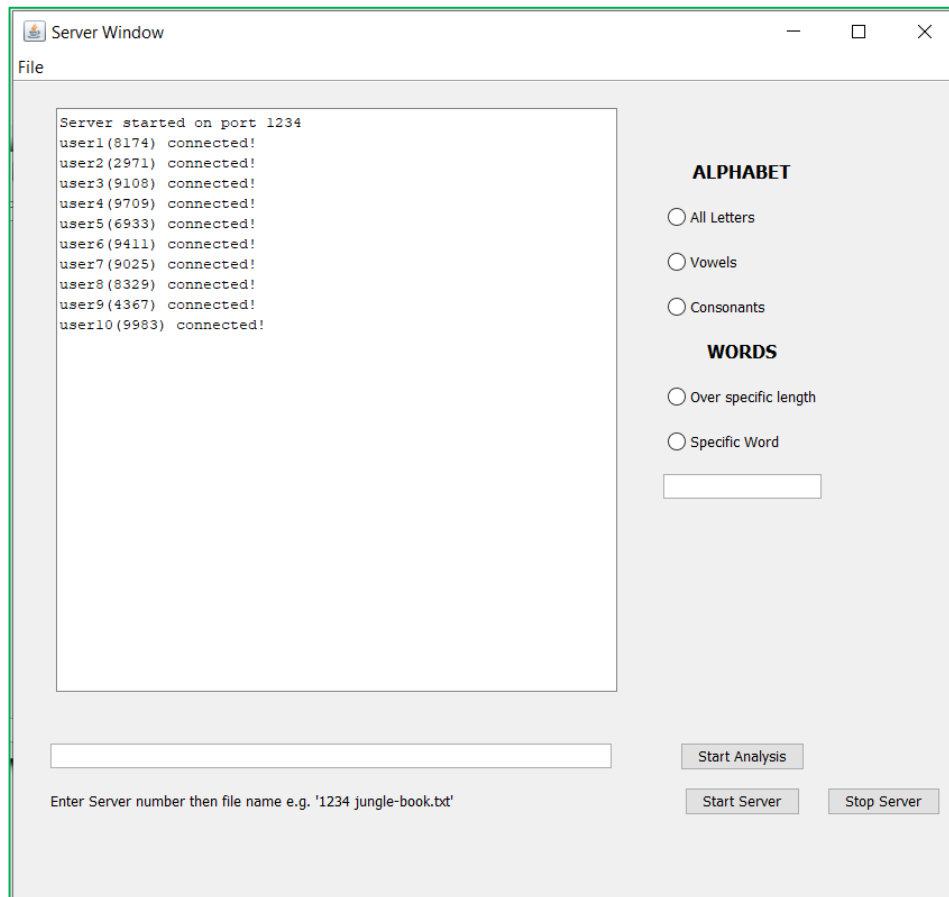
    try {
        in = new BufferedReader(new FileReader(filename));
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

## Add functionality to choose between searching for vowels and consonants

Add functionality choose between searching for all alphabetic characters or vowels (a,e,i,o,u) only

The first change made to the program that covers this and the following two requirements for analysis of a count of words over a specific length or a count of the occurrences of a specific word. Was to add the option for the user to specify the type of analysis then want to carry out.

The five radial buttons shown were all placed into a group so that only a single button could be selected at any time. The start analysis button would first check which button is pressed before carrying out the analysis.



In order to achieve this functionality no changes were made to the letter counting algorithm, instead the change was made to the printing of the results, simply omitting the letters that are not required based on the user's selection.

This approach was much more efficient than creating two new algorithms to be distributed amongst each client connected to the server.

Below you can see the results of each option, vowels on the left and consonants on the right.

```
Server started on port 1234
user1(7965) connected!
user2(7482) connected!
user3(8440) connected!
user4(3306) connected!
user5(1131) connected!
user6(9588) connected!
user7(5367) connected!
user8(1080) connected!
user9(2957) connected!
user10(3042) connected!
starting alphabet analysis of text file
a = 17984
e = 25531
i = 12946
o = 15282
u = 5770
The text file has been processed and took 1476ms
```

```
user6(9883) connected!
user7(3229) connected!
user8(7179) connected!
user9(1233) connected!
user10(4424) connected!
starting alphabet analysis of text file
b = 3679
c = 4104
d = 9462
f = 4436
g = 5335
h = 15343
j = 315
k = 3388
l = 10018
m = 4784
n = 13900
p = 3018
q = 143
r = 10321
s = 11854
t = 18987
v = 1698
w = 5779
x = 129
y = 3672
z = 100
The text file has been processed and took 1441ms
```

## Add functionality to count specific words

Add functionality to count the occurrences of a specific word

When adding the ability to search for the occurrences of a specific word within the text file passed in, the original alphabet analysis algorithm was used as a template.

When the 'start analysis' button is pressed, the program looks to see which type of analysis is to be completed by checking the status of the radial buttons. Below is the code snippet that shows this.

```
// Start Analysis button with action listener
JButton btnStartAnalysis = new JButton("Start Analysis");
btnStartAnalysis.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        /**
         * The method that is called when the button is pressed
         * It first checks to see which type of analysis has been selected
         * based upon the user selection the following steps are all similar
         * a line stating that it has started to analyse the file is
         * appended to the JTextArea
         * the server set start time variable is set by calling the
         * system time
         * then the respective Analysis method is called
         */
        if((rdbtnAllLetters.isSelected() || rdbtnVowels.isSelected() || rdbtnConstantant.isSelected())) {
            history.append("starting alphabet analysis of text file\n");
            server.setStartTime(System.nanoTime());
            StartAnalysis();
        } else if (rdbtnLength.isSelected()) {
            history.append("starting word length analysis of text file\n");
            server.setStartTime(System.nanoTime());
            LengthAnalysis();
        } else if (rdbtnSpecific.isSelected()) {
            history.append("starting specific word analysis of text file\n");
            server.setStartTime(System.nanoTime());
            WordAnalysis();
        }
    }
});
```

Where this method differs from the alphabet analysis, is that it picks up the word it is looking to count from the text entry box directly below this analysis option. This method appends the message to be sent with a '/w/' as well as the key search word. Below is the code snippet that shows this.

```
private void WordAnalysis() {
    server.ResetAnalysis();
    List<ServerClient> clients = server.getClients();
    for(int i=0;i<clients.size();i++) {
        ServerClient c = clients.get(i);
        String data = "/w/"+txtBoxAnalysis.getText()+"/"+server.getList().get(server.getLineCount());
        server.setLineCount(server.getLineCount()+1);
        server.send(data.getBytes(), c.address, c.port);
    }
}
```

A count variable was added to the server and client objects. This was incremented when an occurrence of the specific word occurred. The client end algorithm can be seen below

```
public void WordAnalysis(String msg) {
    String s = msg.replace("/w/", "").trim();
    String[] step1 = s.split("/");
    String word = step1[0];
    word = word.toLowerCase();
    String[] words = s.split("\\W+");
    for (int x = 1; x < words.length; x++) {
        words[x] = words[x].toLowerCase();
    }
    for (int x=1; x<words.length; x++) {
        if (words[x].contains(word)) {
            if(words[x].length()==word.length()) {
                wordCount++;
            }
        }
    }
    String counts = String.valueOf(wordCount);
    sendWordCounts(counts);
    wordCount=0;
}
```

Finally, the total is tallied by the server and the results displayed in the server window text area.

Server Window

File

Server started on port 1234  
user1(6268) connected!  
user2(547) connected!  
user3(5851) connected!  
user4(407) connected!  
user5(1309) connected!  
user6(4935) connected!  
user7(3211) connected!  
user8(1033) connected!  
user9(6113) connected!  
user10(2465) connected!  
starting specific word analysis of text file  
There were 15 instances of the word bear  
The text file has been processed and took 1359ms

**ALPHABET**

☐ All Letters  
☐ Vowels  
☐ Consonants

**WORDS**

☐ Over specific length  
☒ Specific Word

bear

Start Analysis

Start Server Stop Server

Enter Server number then file name e.g. '1234 jungle-book.txt'

## Add functionality to count words over a specified length

Add functionality to count the occurrences of words over a specified length

When implementing this functionality, the same count variable and text box were both used from the specific word analysis along with the same basic structure from the alphabet analysis methods.

This is the method called by the start analysis button:

```
private void LengthAnalysis() {
    server.ResetAnalysis();
    List<ServerClient> clients = server.getClients();
    for(int i=0;i<clients.size();i++) {
        ServerClient c = clients.get(i);
        String data = "/1/"+txtBoxAnalysis.getText()+"/"+server.getList().get(server.getLineCount());
        server.setLineCount(server.getLineCount()+1);
        server.send(data.getBytes(), c.address, c.port);
    }
}
```

This is the method run at the client when it receives a line of text from the server:

```
public void LengthAnalysis(String msg) {
    //System.out.println("client"+client.getName()+" just got "+msg);
    String s = msg.replace("/1/", "").trim();
    String[] step1 = s.split("/");
    int length = Integer.parseInt(step1[0]);
    //System.out.println("the Target Length is " +length);
    String[] words = s.split("\\W+");
    //System.out.println("There are " + words.length + " many words in the scentence");
    for (int x=0 ; x<words.length ; x++) {
        // need to add something that checks its just a word
        if (words[x].length() > length) {
            wordCount++;
        }
    }
    String counts = String.valueOf(wordCount);
    sendLengthCounts(counts);
    wordCount=0;
}
```

This is the method run at the server to receive messages, update the count, send additional lines and print the results:

```
} else if (string.startsWith("/1/")) {
    lineRecievedCount++;
    String msg = string;
    msg = msg.replace("/1/", "").trim();
    String[] temp = msg.split("/");
    String name = temp[0];
    int tempCount = Integer.parseInt(temp[1]);
    wordCount += tempCount;
    if(lineCount < list.size()) {
        SendClientAnotherLengthLine(name.trim());
    } else {
        if ( lineRecievedCount==lineCount) {
            LogData("There were " + wordCount + " words found with a length of " +
                ServerWindow.txtBoxAnalysis.getText());
            long endTime = System.nanoTime();
            long duration = (endTime - startTime);
            LogData("The text file has been processed and took "+(duration/1000000)+"ms");
        }
    }
}
```

The results are printed in the server window text area.

```
Server started on port 1234
user1(6620) connected!
user2(270) connected!
user3(7731) connected!
user4(8988) connected!
user5(6757) connected!
user6(6483) connected!
user7(9107) connected!
user8(866) connected!
user9(2227) connected!
user10(458) connected!
starting word length analysis of text file
There were 16441 words found with a length of 4
The text file has been processed and took 1335ms
```

### ALPHABET

☐ All Letters

☐ Vowels

☐ Consonants

### WORDS

☒ Over specific length

☐ Specific Word

4



## Part 2 Comparison to single threaded (15 marks)

Before discussing the differences and similarities between the three programs, firstly each program will be individually described.

### Sockets chat program

This program consists of 393 lines of code spread across four class files, though two of the class files simply call the other two. Essentially, it is a simple GUI based chat program with a single server and single client, each with their own window. Each window is made up of very similar elements, a text box to take input, a send button to send the text contained in the input box and a text area to display the messages.

The program only allows a single client to connect to the server. The TCP connection is made using `ServerSocket` and `Socket` objects called from their respective classes. Messages are transferred by calling the `getOutputStream` and `getInputStream` methods on the socket object. As soon as a message is typed in either window, and the send button is pressed, then it will be displayed in the log of both server and client windows.

There is no threading present within this program.

### Threaded chat program

This program consists of 336 lines of code spread across four class files. The program allows for multiple clients to connect to a single server. The TCP connection is made using `ServerSocket` and `Socket` objects called from their respective classes.

When a client object is created the user is prompted to enter a name, there is some checking to ensure the name is unique and not already in use. The client object then attempts to connect with the server. When the server receives a connection request, it accepts the connection and calls the `ClientThread` class file creating a thread for that specific client. The code within `ClientThread` sends a message to all other connected clients updating the list as well as the server.

Notably there is no 'chat' based functionality, nor is there any GUI within this program.

### ChernoChat chat program

This program consists of 1253 lines of code spread across 9 class files. The program creates a single server and allows for multiple clients to connect. The UDP connection is made using a `DatagramSocket` object called from its respective class.

There are several GUI windows used in the program for the server, client and login windows. The program allows for messages to be sent from each connected party to all other connected parties. Within the server and client there are 'listening' threads. When a message is received then a 'receive' thread is created. In the case of the server object a 'process' thread is then spawned. Much of the program is laid out in this way, threads created for sending and receiving information.

### Similarities, differences and what next?

All three programs were created in the java language. Each of the three programs involve creating a connection between a server and client/s. The first two chat programs use the 'Socket' class to create a TCP type connection, whereas the last chat program uses the 'DatagramSocket' to create a UDP type connection. The complexity increases as you move from the first program to the last, as does the inclusion of threading. There is no use of threads at all in the first program, the use of a single thread (*or single thread class/call*) in the second and prolific use of threads in the third.

Focusing on the third chat program, it uses threads almost as actors in that they are used on small individual tasks such as receiving a message or processing a message or sending a message. The key difference being there is no communication like you would get with actors. Implementation of some form of actor-based model would be the next step, perhaps executors, further information on their use can be found here:

<https://winterbe.com/posts/2015/04/07/java8-concurrency-tutorial-thread-executor-examples/>

## Part 3 General Commenting (15 marks)

The code has been commented, in depth and repeatedly. Again, the use of the universities GIT server assisted in the process. Following the routine of, adding comments to a file or section, commit that to the repository. In total approximately 800 lines of comments have been added to the program. Typically, in the case of methods, the comment will break down the logic being implemented, as shown in the examples below;

```
* This method ConcatLines it takes in an int for the start and end
* @param start is a number
* @param end is a number
* @return
* logic:
* 1. Set a String variable called temp to "/a/"
* 2. Check if the end variable is larger than the size of the
*    value returned by the getList method called from the server object
* 3. Using a for loop starting at start and incrementing until you hit end
*    add the result of the getList.get(i) and new line character to the string
* 3. Return the string
* this method does not look like it will be used???
*/
String concatLines(int start, int end) {
    String temp = "/a/";
    if(end > server.getList().size()) {
        end = server.getList().size();
    }
    for(int i=start;i<end;i++) {
        temp+=server.getList().get(i)+"\n";
    }
    return temp;
}
```

```
/**
 * Start Analysis method logic:
 * 1. Call the ResetAnalysis method from the server
 * 2. Create a list of ServerClient objects
 * 3. populate the list by calling the getClients method
 * 4. Drop into a for loop starting at 0 and incrementing
 *    by 1 while it is less than the size of the list just
 *    created
 * 5. Create a ServerClient object called 'c' and assign
 *    the ServerClient object in the list at that index
 * 6. create a String called data, set it to a concanation of
 *    (i)      "/a/"
 *    (ii)     server.getList().get(server.getLineCount())
 * 7. Set the line count variable to line Count +1, by calling the get method
 *    then set method, something not right here
 * 8. Call the send method on the Server passing in the client address
 *    client port and message
 */
private void StartAnalysis() {
    server.ResetAnalysis();
    List<ServerClient> clients = server.getClients();

    for(int i=0;i<clients.size();i++) {
        ServerClient c = clients.get(i);

        String data = "/a/"+server.getList().get(server.getLineCount());
        //System.out.println("Sending the line: "+data);
        server.setLineCount(server.getLineCount()+1);
        server.send(data.getBytes(), c.address, c.port);
    }
}
```