# **Report 1**

# **EECS 3214 : Programming Assignment 1**

## Part II: Writing a clicker program.

**Author: Natalie Sheinin**

**211507704**

**CSE13039**

**Table of Contents**

1 Introduction……………………………………………………………………3

2 Overall Program Design……………………………………………...……...3

2.1 How It Works……...………………………………………...……….3

2.2 Design Decisions…………………………………………...…….....4

3 Improvements and Extensions…….………………………………………..6

4 Program Issues……………………………………….……...……………....8

5 Grader Instructions…………………….………………………...…………..9

**1 Introduction**

This program consists of a server and client that implement a clicker program similar to the ones used at York to collect student responses to questions in class. Multiple clients should be able to connect to the server simultaneously, be authenticated through their student number, recieve number of choices for the question, and be able to “click” (input) their choice from the selection. The client-side of the clicker program ends there and the connection is closed.

**2 Overall Program Design**

**2.1 How It Works**

The server side program was designed with three layers in mind, one to manage the Server UI - serve the 3 instructions: START\_QUESTION, END\_QUESTION, and LIST.

The second layer, to listen to connections and create a separate socket for each incoming client connection. And the third layer, to serve the client and manage communications.

*i) MainServer*

When the MainServer program is run, gives the instructions of the commands available to the user (instructor). There are the 3 sections to the program:

1. public class MainServer implements *Runnable*

This class declaration contains the other 2 sections of the program, as well as the global functions and variables. Most importantly, it contains the HashMap that is used to record student answers.

The following functions are created to be used by other programs to work with the HashMap data:

public void recordAnswer(String sn, String answ)

public boolean checkStudent(String sn)

2. public static void main(String[] args) throws IOException

This method serves the instructions by matching user standard input to the three commands.

3. public void run()

This method serves to run the thread that gets created once, and solely for the purpose of handling the START\_QUESTION instruction. Within it, a volatile boolean variable is used as a flag to control when this thread should stop running (when END\_QUESTION instruction is issued).

This is where the ServerSocket is declared and initialized, and loops while listening to connections through the portnumber.

For every connection made, a new ServerThread is created.

*ii) ServerThread*

The serverThread class extends Thread and creates an instance of the MainServer class to have the public methods at its disposal:

public class ServerThread extends Thread

MainServer ms = new MainServer();

As in MainServer.java, ServerThread.java has a run() function, as well as a section to handle the file, it contains 2 methods, one to read in the file and store in a String list, and the other to authenticate the studentNumber based on the list:

public void readFile()

public boolean authenticate(String studentNumber)

The next portion of the program, is the public void run() and focuses on communication with the client. In sequential order, it reads in the file, authenticates the user, communicates number of choices, receives the selected choice, and records the choice.

The client-side is a very simple program with minimal functionality: it focuses on communication and authentication of user input for the choices.

It connects to the socket, reads in from the socket, prints to the standard output, reads from standard input, and sends it to the listening socket.

**2.2 Design Decisions**

1. Storing the student numbers in a text file.

It was decided to store the student numbers in a separate txt file because it seemed to be the best solution for the scope of this project. Since this project was not cross-platform and didn’t require scalability, no java classes or databases were made to support the student number object (especially since that there would only be one attribute to the object).

In the program, a String List was used to store the values with size of 100.

String[] numbers = new String[100];

Also, two supporting methods were created as mentioned above, one to read in the file, and another to authenticate incoming student numbers against the List.

It is clear that this technique for storing the student numbers is not the most secure and scalable (the list size is limited).

1. Implements Runnable versus extends Thread

When deciding how the design of the program would be, I settled to use “implements Runnable” for the START\_QUESTION port listening since it would minimize the number of java programs I would need to create in total. I also decided to use extends Thread for dealing with the communication-per-client. Using the Runnable interface allowed for me to have the run() method in the same class as my main main() method, meaning I could share global static variables between the two and synchronize the behaviour of the thread with the demand of the main() method. Them being in the same class made it simpler to share the resources (data structures, and variables) and update them when necessary.

1. HashMap

The reason for using a HashMap to store the recorded student answers is because it works with key-value pairs, which was the necessary data that needed to be stored for the specifications of this project. The key being the student number, and the value being the answer selected by the student.

static HashMap<String, String> answers = new HashMap<String, String>();

The main decision was between using a HashMap or a HashTable, and after doing some research, it seemed like HashTable is used more often for mutli-threading projects where synchronization is needed. Although this project was a multi-threading project, only one type of the threads implemented in this project - ServerThread was the one accessing this resource. Moreover, it was known that there could be multiple ServerThread instances running at the same, however, it was only known that the clients would be dealt with manually, so the resource would not need synchronized access.

Also, in MainServer.java, two public methods were provided for means of accessing and utilizing the HashMap:

public void recordAnswer(String sn, String answ)

{

answers.put(sn, answ);

}

public boolean checkStudent(String sn)

{

return answers.containsKey(sn);

}

1. Hardcoding

A lot of hard-coding was done to save time on developing more elaborate methods and to save on authentication methods and catching errors.

FileReader fin = new FileReader("Students.txt");

String hostName = "localhost";

int portNumber = 7704;

1. Clearing the HashMap only when START\_QUESTION command is issued.

Since the HashMap is the only instance being used to store all recoreded answers, however it is possible for MainServer.java to run forever with the use of START\_QUESTION and END\_QUESTION countless times, it was decided to ensure that the HashMap is cleared before every new session started (meaning prior to opening a port for listening), meaning at the beginning of START\_QUESTION.

This was done using: answers.clear();

This could have been implemented differently to also store the question/session ID, and have the full list printed of every session, or even with choices given to instructor for which question/session ID to print the LIST for, this design decision seemed to fit the specifications of the assignment to the most basic functionality.

**3 Improvement and Extensions**

1. More secure

Security of the program was not considered, but given more time would have been given greater attention. All global objects would be made private, and public supporting methods would have been written to support the usage and modification of the objects (being variables or data structures).

e.g. The HashMap in MainServer.java

1. GUI

Given more time, a graphical user interface could have been implemented, either using the Swing library, or even using JSPs, HTML, CSS, and possibly JS. With buttons as the 3 separate commands, and a file opener to select the text file which should be used as the necessary student number database.

1. Stronger authentication.

The authentication throughout the program was either weak or lacking completely on both client and server-side. There was no authentication for when the user(instructor) enters the number of choices, them having to be between 2-5. A simple standard input reader, while loop, and String matcher could have been used to ensure valid inputs.

Also, authentication could have been implemented for ensuring that no two same students numbers submit answers for the same question. In a realistic scenario, once an answer has been recorded for a question, a student should not be able to go back and re-answer a question. This portion could have been implemented in the public boolean authenticate(String studentNumber)

function, using the code that was commented out on lines 60- 63 in ServerThread.java

Lastly, proper authentication could have been implemented for when accepting client input for their choice. Although the following was implemented in Client.java:

if (!(choice.equals("a") || choice.equals("b") || choice.equals("c") || choice.equals("d") || choice.equals("e")))

This didn’t limit the client from entering an “e” choice when there were only 4 choices available as per the instructor’s input. Thus, stronger authentication and cooperation with the server-side processes could have been implemented.

1. Provide more commands.

Given more time, I would implement more server-side commands and functionality, such as:

1. How many users are actively connected at a certain point in time.
2. GRAPH - to produce a bar graph of the answers
3. FILENAME - to allow user to upload / select / enter filename to be used as the database for the student numbers.
4. Read any file.

Currently implemented through method: public void readFile()with the filename hardcoded:

FileReader fin = new FileReader("Students.txt");.It would be changed to public void readFile(String fileName), reading in the filename from the user and passing it to the FileReader object.

**4 Program Issues**

1. Runs locally instead of over the network

1. Does not support a client-side UI.

Due to a lack of time on my behalf, I was not able to implement the client-side UI with the required commands: STUDENT\_NUMBER, CLASS\_INFO, and ENTER\_CHOICE(). However, throughout the process, in a sequential manner, the server requests and prompts the user for input for the student number, and the choice. Hence, the user is not able to prompt the server to be able to enter the information, but rather the server drives the communication.

1. Allows clients to connect to server after END\_QUESTION is entered server-side.

The volatile boolean variable, along with the while loop, wasn’t properly implemented to stop the thread in MainServer.java when END\_QUESTION instruction was issued. The boolean variable only modified the state of the outer while loop, while in the program the inner while loop is the one responsible for listening to connections, and should have been the one who’s “true/false” state was modified. This error was immediately fixed after the code review session.

while (!shutdown)

{

...

while (conn)

{

new ServerThread(s.accept()).start();

1. Client-side authentication of choice entered is not complete.

There is no proper mechanism to ensure that the client enters the correct choice,

there are some limitations but they are not real-world accurate:

if (!(choice.equals("a") || choice.equals("b") || choice.equals("c") || choice.equals("d") || choice.equals("e")))

This only limits the client from entering anything other than those letters, however the number of choices can be anywhere from 3-5, thus if the instructor selects to host a 3-choice question, “d”, and “e”, can still be accepted. The authentication can be done on the server-side as well.

1. Once the server receives the choice submitted by the user, and it updates the hashmap, it does not display this to the user (instructor).

This is a minor code requirement that was not implemented. As per the instructions, “Then it updates its table of choices received and displays it to the user (instructor).”. This could have easily been completed with a minor code adjustment, to listen for updates to answer (the HashMap which stores all the student answers) and display the table of choices received as required.

**5 Grader Instructions**

The following four files were submitted:

MainServer.java

ServerThread.java

Client.java

Students.txt

Please compile all the \*.java programs by using:

javac [filename].java

To run the server, please run the MainServer.java program by using command:

java MainServer

Once running, the Server will give clear instructions on how it can be used as per the specifications.

To run the client, please open a separate terminal for each Client and run the Client.java program by using command:

java Client

The Students.txt file specifies the student numbers that can be used separated by newlines. It can be modified to store any acceptable student numbers.