CMPS 284 Project1 Report

Group 5

October 2, 2016

Contents

1 Project Design

The project was developed using Java.

An abstract Server has basic methods such as `run()` and `stopServer()` that each of the actual Servers implements differently. For example, TCP-Server uses `ServerSocket` data type from `java.net` framework, whereas UDP-Server uses `DatagramSocket`. Moreover, this abstract class contains common methods such as `handleFatalException(Exception ex)` and global constants for the communication protocol commands such as *PLAY, STOP* and others, and these can be called from the Client.

`ClientGame` is an abstract class that handles common functions for both TCP-Client and UDP-Client. These functions are basically: asking for user input using a `System.in BufferedReader`, and validating it on the Client side, as well as handling exceptions that might occur in case of send/receive failure.

`ServerClientConnection` is an abstract instance class that handles a Client. There's one public method which is `dealWithClient()` meaning handle all the requests and respond accordingly to this Client. When the Client sends PLAY then this `ServerClientConnection` instance starts to interactively send an encoded word and wait for a guess character from this Client until the current game is over…

The TCP-Server uses threads from the `java.util.concurrent.Executors` package to handle up to 6 simultaneous Clients. However, the current implementation of UDP-Server doesn't support this.

Each Server contains an implementation inner class of `ServerClientConnection`. For example, the TCP-Server requires 3 objects to handle a Client interaction: a Socket `connectionSocket` where the accepted connection is now delegated to this Socket, a `BufferedReader inFromClient` which reads String data incoming from the Client to the Server, and finally a `DataOutputStream outToClient` which sends data (in the form of String objects) to the Client. However, the UDP-Server requires `DatagramPacket`: One to read packets from the Client and one to send data to the Client.

`FileUtils` is a utility class which reads from a file the list of Hangman games and writes to a file when the Client wins or loses a game. A Hangman game is basically the original word and its given number of tries, and also the encoded word and the remaining number of tries.

2 Installing and Running the Application

The project was compiled on JDK 1.6; therefore, please compile it with JDK1.6 or above! And run it with JRE 6 and above! Also, make sure there is an `in.dat` file for the Hangman games or pass the file name as argument. We provide here 3 different ways to install/run our app:

2.1 Easy Run on Windows: Dont Compile!

The application is shipped in jars! Therefore, in order to run on Windows, drag the Server and drop on `run.bat`. Then, do the same for the Client! You may try to use the `run.sh` for Linux. Keep in mind that console jars cannot be run with double-click start. You have to type the following command:

```
> java -jar <jar-file>.jar
```

2.2 Make and Run on Linux!

Linux makes life easy with its `makefile` utility. Say you want to run TCP, then type the following:

```
> make                       # this compiles the Java source!
> make runTCP F=in.dat       # starts the Server and a Client
# play here...
# and maybe from another terminal, run another Client as follows:
> java ClientTCP             # starts the game for another Client!
```

2.3 Compile and Run Anywhere!

Open a terminal in the directory which contains the source files and type the command:

```
> javac *.java
```

Once done, you can now run the Java application. Start the Server, for example say the TCP-Server, in the same terminal by calling the command:

```
> java ServerTCP <in-file-name>
```

Then, from a different terminal, run the corresponding Client as the following:

```
> java ClientTCP
```

Now, you can enjoy the game!

Note:
It is important to kill the Server process if you are done using it. To do so, you can type the following command:

```
ps -A     # it displays all the processes running
```

Then, you can kill the Server process by using the command:

```
kill <pid>
```

3 Project Source Files

- **Server**: an abstract class containing command protocols for communication between Client and Server among other common functions such as start and stop.
- **ServerTCP**: an implementation of the **Server** in TCP.
- **ServerUDP**: an implementation of the **Server** in UDP.
- **ServerClientConnection**: an abstract class on the Server-side to deal with a Client since both UDP/TCP Server and their corresponding Clients follow a protocol for communication. For example, the Client requests *PLAY* and then the Server responds with - - - - - - - - - -
- **Hangman**: an instance class to manage a word and its tries along with the required methods which do the checking and updating.
- **ClientGame**: an abstract class which both Clients implement. It contains methods such as asking for input and validating it.
- **ClientTCP**: an implementation of **ClientGame** where a `Socket` is utilized for connection and `BufferedReader` and `DataOutputStream` are for Input/Output with the Server.

- **ClientUDP**: an implementation of ClientGame where a DatagramSocket is used for connection and DatagramPackets for sending and receiving data.
- **FileUtils**: a utility class that reads the games and returns a `Map` from word to number of tries. This class also provides functions to write at the end of each game over.

4 Contribution

A good team collaborates efficiently by communicating and to solve the assignment. Since the first day, we agreed to divide the major tasks as follows:

- **Raafat Waheb**: writing the following classes: `ServerClientConnection, ServerTCP, ClientGame`. Also, writing this report and creating easy-installer.
- **Mohamad Zein El Deen**: writing the following classes: `Server, ClientTCP, Hangman. And creating the makefile`.
- **Fawzi Chwayfety**: writing the following classes: `ServerUDP, ClientUDP, FileUtils`.

As for the minor issues, each one who found a bug or possible improvement, he reported it to the other team members and fixed it! Note that not a single class was written solely by one member.

5 Known Issues

The UDP-Server currently doesn't support multiple Clients at the same time. In fact, it behaves incorrectly when two Clients connect at the same time!

6 References

In software development, reading sample programs after understanding the concept is your starting point:

1. Simple Java TCP Example
2. Simple Java UDP Example

We didn't need further help!

7 Project Description

This project creates a Client-Server application of the good old Hangman game that we used to play on paper and pen! A Hangman game is basically a word that the user has to try to guess! The protocol for this game is simple:

1. Show an encoded word such as: - - - - - - -
2. Ask the user for a letter as a guess such as: e
3. Update the encoded word accordingly if the word contains the user guess letter such as: - - - - e - -
4. Repeat until the word is all guessed or the user runs out of tries!

In the original paper and pen game, we used to draw a man being hanged! And each time the user fails to make a good guess, a part of the hanged man dies...

Now our version is slightly different: The Server has a list of Hangman games. The Client communicates with the Server through protocol commands such as *PLAY, STOP...* There are also two implementations of this game: TCP which is a reliable Server-Client communication and UDP which is not reliable.