CS 421 – Computer Networks
Programming Assignment I
Instant Message App

Fall 2017

Due: Nov. 6, 2017 at 11:59PM

I. Introduction

In this programming assignment, you are asked to implement a program in Java. The program is an instant messaging application which uses a server to register and to find addresses of the other users, and a peer-to-peer connection to send and receive messages directly to other users. The server program will be provided to the students with the assignment.

The project has both a **design** and an **implementation** part. There will be parts where there are no implementation specifications, any implementation within the following constraints which satisfies the functional specifications will be accepted.

For the design part, you are asked to write a report in which you will explain your design and discuss the decisions you made.

Constraints For this programming assignment, you are not allowed to use any third party HTTP client libraries, or the HTTP specific core or non-core APIs supplied with the JDK including but not limited to *java.net.HttpURLConnection*.

Your project grade will be penalized by 50% if you use one of these libraries.

The goal of the assignment is to make you familiar with the HTTP, TCP and UDP protocols. You have to implement your program using the Java Socket API of the JDK. If you have any doubt about what to use or not to use, please contact us.

When preparing your project please mind the formatting as this project will be auto-graded by a computer program. Any problems in the formatting can create problems in the grading. While you will not get a zero from your project, you might need to have an appointment with us for a manual grading. Problems caused by Windows/Linux

incompatibility will have no penalty. However, errors caused by incorrectly naming the project files and folder structure might cause you to lose points.

II. Design Specifications

Your client program must be **a console application** (no graphical user interface(GUI) is allowed) and should be named as InstantMessenger (i.e., the name of the class that includes the main method should be InstantMessenger). Your program should be run with the

java InstantMessenger <username> <serverURL> <mode>

command where <username>, <serverURL> and <mode> are the command-line arguments. In detail, this command-line arguments are:

- <username>[Required] The username of the client where all clients have only one username. No spaces, colons ':' or at sign '@' is allowed inside the username.
- <serverURL> [Required] The URL of the BilkentMessageServer program.
- <mode> [Required] The mode is either "send" or "listen" (without the quotes). If "send" is selected the app is configured to send messages. If "listen" is selected then the app is configured to listen for any incoming messages from other clients.

When a user enters the command above and chooses **listen** as the mode, the program will send an HTTP POST request to the server to register its username, IP address and port number. When a user is in the **send** mode, the program will send an HTTP GET request to the server to get the list of currently registered users, along with their IP addresses and port numbers. Fortunately, the server we provide uses a simple protocol where there are two commands.

REGISTER <username>@<IP>:<Port> (e.g. "REGISTER yarkin@192.168.1.13:54321")

When the server receives an **HTTP POST** request with a body that contains a message formatted as above, it will create a registry for the user in its userlist.txt file. Note that the HTTP response from the server may contain a status code indicating an error (for example if username includes characters that are not allowed or a user with the given username already exists in the userlist). Therefore your client program needs to check the status code of the response and ask the user to enter a valid username until the POST request is accepted by the server.

The other command is accessed through sending a **GET** message to the serverURL/userlist.txt which returns a file where all registered users are written in a line-by-line fashion. E.g.

yarkin@192.168.1.13:54321
bulut@192.168.1.13:12345
deniz@192.168.1.41:12345

You will use this file to determine the other users in your network.

If the client program is run in **send** mode, it enters into a simple read-evaluate-print loop (REPL) where it waits for commands. For this project, you have to design and implement **5 commands** which are defined as functionalities. You can implement them freely, without use of the aforementioned libraries of course. Commands are:

a. *list*

   When list command is entered the program retrieves the current user list, stores it locally and prints the names of the users available.

b. *unicast <username> "<message>"*

   This command will send the <message> to the desired user IF the user is in the registry (Your program should automatically update the local registry before sending.). You do not have to check for acknowledgments if the desired user is registered; just print "message is sent to <username>" (without the quotes). IF NOT, i.e. the user is not in the list, simply print "user <username> is not found". Note that the messages should be entered between quotation marks ("").

c. *broadcast "<message>"*

   This command will send the <message> to all users (except itself) registered in the server after updating its local registry.

d. *multicast [<username1>, <username2>, ..., <usernameN>] "<message>"*

   This command will send the <message> to users registered in the server and given in the arguments after updating its local registry. It will also print whether the desired users are in the userlist or not, just like unicast command.

   e.g. where the user is veli and tries to send to yarkin, ali and bulut, while only the users yarkin and bulut exist in the server userlist.

```
>> multicast [yarkin, ali, bulut] "merhabalar"
>> message is sent to yarkin
>> user ali is not found
>> message is sent to bulut
```

e. *exit*

The program will terminate.

If the program is run in **listen** mode, it will enter into an infinite loop. Inside the loop the program will listen to the incoming messages and print them in the form of "<username>: <message>" (without the quotes). You should use the address information of the received packet to determine the username of the sender. **Also, the listening client should print who sent the message.**

For example, let us assume that your IP address is 192.168.1.33 and you also run the message server locally (at 192.168.1.33) and you chose the username rick. When you start the program with command:

>> java InstantMessenger rick 192.168.1.33 listen

The program will automatically bind to a port provided by the operating system. Let us assume it is bound to port 12345. The program then sends a POST message to the server URL which has the following body:

REGISTER rick@192.168.1.33:12345

And the server updates the userlist file with URL <serverURL>/userlist.txt.

Another user called morty wants to join the network as a sender, he runs the command

>> java InstantMessenger morty 192.168.1.33 send

Now the program will automatically bind to a port provided by the operating system, however this time it will not send its information as it is a sender and not a receiver client.

Now **morty's** program awaits user input. **Morty** wants to see who is online so he enters

>> list

The program outputs:

>>The online users are:
>>rick

Now morty wants to send a message to **rick**, so he enters the command:

>> unicast rick "what is my purpose?"

The program outputs:

>> message sent to rick

rick sees the following message in his console:

>> morty: what is my purpose?

A third user called **jerry** wants to join as a listener. Now morty wants to send messages to both **rick** and **jerry**. There are two ways he can achieve this:

First is by multicast

>> multicast [rick,jerry] “what is my purpose?”

The program outputs:

>> message sent to rick
>> message sent to jerry

Second is by broadcast which sends messages to all available users

>> broadcast “what is my purpose?”

rick and **jerry** sees the following message in their consoles:

>> morty: what is my purpose?

III. Implementation Specifications

While this is aimed to be a design oriented project, you still have some constraints on how to implement certain functionalities.

1. Use TCP connections for communicating with the message server and you have to use HTTP messages to communicate.
2. For peer-to-peer communication between the clients, you will use UDP as the transport layer service. Since UDP is unreliable, there might be packet losses, however you do not have to handle these losses.
3. Turn in the version which implements only the features listed in the assignment. If you’d like to improve your project further you can do so, however since these will cause compatibility issues with auto-grader, please turn in the original version.

Assumptions and Hints

- Please refer to W3Cs RFC 2616 for details of the HTTP messages.
- Please contact your TAs if you have any questions about the assignment.
- Since the messages that users send are written between quotation marks (“”), you can assume that the message itself does not contain quotation marks.
- You can modify source code of the server for experimental purposes. However, do not forget that your projects will be evaluated based on the version we provide.

Assignment Report

You need to submit a short and simple report **in pdf format (important) named as “**[CS421_PA1]AliVelioglu20111222.pdf” where you evaluate your results in three samples given. (Note that the program will be evaluated on other servers and files.)

Your report **must** include the following:

- A complete description of your protocol for peer-to-peer communication between the clients. (How clients understand who sent the message, etc.)
- How can you improve the protocol? List 3 things that would improve this messaging application.

Submission rules

You need to apply all the following rules in your submission. **You will lose points if you do not obey the submission rules below or your program does not run as described in the assignment above.**

- The assignment should be submitted as an e-mail attachment sent to yarkin.cetin[at]bilkent.edu.tr. Any other methods (Disk/CD/DVD) of submission will not be accepted.
- The subject of the e-mail should start with [CS421_PA1], and include your name and student ID. For example, the subject line must be

  [CS421_PA1]AliVelioglu20111222

  if your name and ID are Ali Velioglu and 20111222. If you are submitting an assignment done by two students, the subject line should include the names and IDs of both group members. The subject of the e-mail should be

  [CS421_PA1]AliVelioglu20111222AyseFatmaoglu20255666

  if group members are Ali Velioglu and Ayse Fatmaoglu with IDs 20111222 and 20255666, respectively.

- All the files must be submitted in a zip file whose name is the same as the subject line **except the [CS421_PA1] part**. The file must be a .zip file, not a .rar file or any other compressed file.
- All of the files must be in **the root of the zip file**; directory structures are not allowed. Please note that this also disallows organizing your code into Java packages. The archive should not contain:
  – Any class files or other executables,
  – Any third-party library archives (i.e. jar files),
  – Any text files **except the report in pdf format**,
  – Project files used by IDEs (e.g., JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans etc.). You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.
- The standard rules for plagiarism and academic honesty apply. Please refer to 'YOK Student Disciplinary Rules and Regulations', items 7.e and 7.f, and 'Bilkent University Undergraduate Education Regulations' item 4.9.