Exercise01: Server/Client+Threads

Objectives:

- To learn to use Threads
- To learn to write server client code

Work with your group (or by yourself). Each group should upload only one submission.

First, open blackboard, go to Assignments link, and then download exercise01.zip file into your workspace (U:\workspace or something like that!). Then, unzip.

1 TUTORIAL SECTION: PLAYING WITH Threads & Server/Client

1.1 Threads

Go to ThreadExamples Directory.

Then, go over the threads pdf file.

Next, play with the programs in "threadExamples". Upload to Eclipse and run them.

There are four examples.

1. The first one shows how to create Threads by extending the Thread class.
2. The second example shows an alternative using implementations of the Runnable interface.
3. The third example shows problems when sharing modifiable data between threads.
4. The fourth example shows how such problems can be handled in Java by using the synchronized keyword that makes a thread that is currently doing the method to complete it before letting other methods start the method.
5. There is a lot more to threading – but that is outside the scope of this class. This info should be enough to get you started.

1.2 Server Client

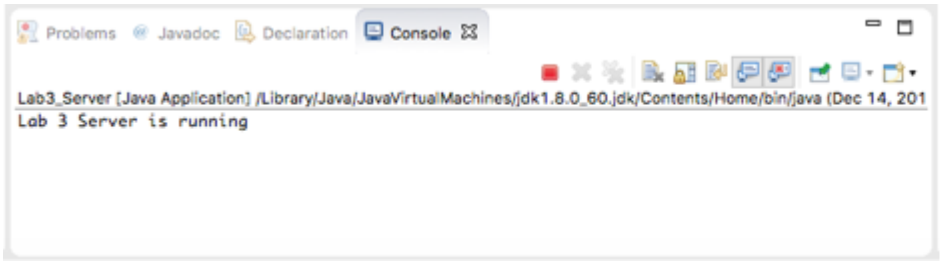
Go to ServerClientExamples folder.

Then, go over serverClient.pptx.pdf .

There are six programs which are to be run in pairs: MyServer and MyClient, ListServer and ListClient, and MyBrowser and MyWebServer.

1. "MyServer.java" shows code for a sample Server program. Run it on Eclipse. If you run it multiple times, the later runs will all fail because port 4444 is already being reserved by the first run. "MyClient.java" shows code for a sample client program. Run it several times. Each time a client program is run, it sends data to the server - which prints out the information on the screen.

2. Don't forget to kill the server program after you are done playing with the code. For killing the server you need to press the red square in the Eclipse.



```
Lab3_Server [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java (Dec 14, 201
Lab 3 Server is running
```

3. ListServer.java and ListClient.java have a more interactive communication between the server and client. Here the Client spawns a separate thread to handle the incoming messages from the server (so that the rest of the client does not block).

4. MyBrowser.java tries to give some idea of how a browser like Chrome would work. You can type an address (like www.google.com) and a port (80) and see it talk to google's server. It will basically send HTTP Requests.
MyWebServer.java tries to give some idea of how a server like Apache etc would work. Essentially, it will need to understand HTTP Requests and send back HTTP Responses. You can even connect to this server by using the MyBrowser.java client.

2 APPLICATION SECTION: Server Client/Thread

Create a chat application using Server Client. Here are some features that you should incorporate.

Note, we may have under-specified what you need to do. If so, make up your own rules on what to do for those situations. In other cases, follow the requirements carefully.

2.1 Connect to Server

a) When you start the client, it should come up with a prompt (NOTE: you can either make it entirely text based – or you can use the provided GUI template in exercise01.zip to create a GUI frontend).

Enter your Name: (Type in your name, then press Enter)

b) After entering name, client should be connected to server.

c) Now, client will be shown a menu:

1. Send a text message to the server

2. Send an image file to the server

What happens next depends on which of the menu items is selected.

2.2 Send text message to server

1. Send user name and message to server.
2. Messages should be encrypted before sending to the server.
3. Messages should be displayed in your UI and the server's UI after decryption.

You should use Base64 class included in Java API to encode and decode your message. See https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html. Send image file to server.

2.3 Send the image file to the server

1. Images should be encrypted using Base64 class before sending to the server. You should use Base64 class to encode and decode images. Also, send the image file name.

2. Use java's object streams (to send and receive the data)
   https://docs.oracle.com/javase/tutorial/essential/io/objectstreams.html

   https://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html

3. Image file name should be displayed in client and server UIs. No need to display the actual image.
4. The image should be stored on the server side as an image file in images folder with the file name

   <sender's name>+<received time>.

5. When receiving an image message, your UI should show the image file name (from the server).

6. Received images should be stored in the same directory as the client code being executed.

2.4 The server should keep history of chat in "chat.txt".

1. All messages are stored along with corresponding user name.

2. Store a message number along with each message.

3. For text messages, the message should be stored in the chat.txt file.

4. For images, the name of the image file should be stored in the chat.txt file.

5. On restarting the application, your program should not overwrite the chat.txt file.

2.5 Admin facilities

When you type in your name as "admin" (on the client side), you should provide the following menu items.

1. Broadcast message to all clients.

2. List messages so far (from chat.txt). Note that ALL clients messages are stored in the same chat.txt file.

3. Delete a selected message (from chat.txt) – give a message number.

HINT-1: In the implementation, to have the CLIENT not "hang" waiting for the server to send message, you will need to create a thread whose sole purpose is to listen for messages from the server.

3 Submission:

Zip your Eclipse project and submit on black board. Remember there is only one submission per group. Make sure to include all the files that are needed in order to run your program(s).